T.C.
Radyo ve Televizyon Üst Kurulu

WEB TABANLI UYGULAMALARDA GÜVENLİK AÇIKLARI VE ÖNLEMLER

UZMANLIK TEZİ

Necmettin ÜNAL

ANKARA

HAZİRAN / 2021

T.C.

Radyo ve Televizyon Üst Kurulu

WEB TABANLI UYGULAMALARDA GÜVENLİK AÇIKLARI VE ÖNLEMLER

UZMANLIK TEZİ

Necmettin ÜNAL

DANIŞMAN

Üst Kurul Uzmanı Cem ULUOĞLAKCI

ANKARA

HAZİRAN / 2021

Radyo ve Televizyon Üst Kurulu Başkanlığına,

Bu çalışma, Tez Değerlendirme Komisyonu tarafından oy birliği/oy çokluğu ile Uzmanlık Tezi olarak kabul edilmiştir.

	Adı Soyadı	İmza
Başkan :		
Üye :		
Üye :		
Üye :		
Üye :		

ONAY

…./…./2021

Ebubekir ŞAHİN

Üst Kurul Başkanı

TEZ TESLİM TUTANAĞI VE DOĞRULUK BEYANI

Radyo ve Televizyon Üst Kurulu'nda görev yapan Uzman Yardımcısı olarak, Radyo ve Televizyon Üst Kurulu Tez Hazırlama Yönergesi'ne uygun şekilde hazırlamış olduğum uzmanlık tezi ilişikte sunulmuştur.

Bu uzmanlık tezindeki bütün bilgileri akademik kurallara ve etik davranış ilkelerine uygun olarak toplayıp sunduğuma; ayrıca, bu kural ve ilkelerin gereği olarak, çalışmada bana ait olmayan tüm veri, düşünce ve sonuçları andığımı ve kaynağını gösterdiğimi beyan ederim.

Bilgilerinizi ve gereğini arz ederim.
…./…./2021

Necmettin ÜNAL
Üst Kurul Uzman Yardımcısı

Uzmanlık Tezinin Adı:
Web Tabanlı Uygulamalarda Güvenlik Açıkları ve Önlemler

EK:
1- TEZ (6 adet)
2- CD (1 adet)

Tezi Teslim Alan

Daire Başkanı
…./…./2021

TEŞEKKÜR

Bu tez çalışmasının hazırlanmasında desteklerini esirgemeyen başta Üst Kurul Başkanı Sayın Ebubekir ŞAHİN'e, Daire Başkanım Sayın Olcay ÇİÇEKLİTAŞ'a ve Daire Başkan Yardımcısı Sayın Murat İNCE'ye teşekkürlerimi sunarım.

Bu süreçte en büyük manevi destekçilerim olan sevgili eşim Wajd ve sevgili kızım İrem'e özellikle teşekkür ederim.

İÇİNDEKİLER

TEZ TESLİM TUTANAĞI VE DOĞRULUK BEYANI I
TEŞEKKÜR II
İÇİNDEKİLER III
KISALTMALAR LİSTESİ VI
ŞEKİLLER ve TABLOLAR LİSTESİ VIII
GİRİŞ 1
TEZİN KONUSU 5
TEZİN KAPSAMI 6
TEZİN AMACI 6
TEZİN ÖNEMİ 6
TEZİN YÖNTEMİ 7
TEZİN SINIRLILIKLARI 7
BİRİNCİ BÖLÜM 8
WEB UYGULAMA GÜVENLİĞİNİN TEMELLERİ 8
1.1 WEB UYGULAMA TEORİSİ 8
1.2. GÜVENLİK AÇIĞI KAVRAMI VE OLUŞTURDUĞU RİSKLER 13
1.2.1 Veri Kaybı 14
1.2.2 Veri Hırsızlığı 14
1.2.3 Yetkisiz Veri Girişi (Veri Enjeksiyonu) 15
1.2.4 Hizmet Reddi 15
1.3 TEMEL KAVRAMLAR VE ARAÇLAR 16
1.3.1 Web Sunucusu (Web Server) 16
1.3.2 HTTP 18
1.3.3 HTTPS 18
1.3.4 TLS/SSL Sertifikası 19
1.3.5 Veri Tabanı 20
1.3.6 SQL 21
1.3.7 HTML 21
1.3.8 Javascript 22
1.3.9 XML 23
1.3.10 Hash 24
1.3.11 Yanlış Alarm (False Positive) 25
1.3.12 DNS 25
1.3.13 HTTP Çerezi (Cookie) 26

İKİNCİ BÖLÜM 27
SALDIRI TÜRLERİ VE ÖNLEMLER 27
2.1 ENJEKSİYON (INJECTION) 28
2.1.1 Önlemler 29
2.2 OTURUM VE DOĞRULAMA HATALARI (BROKEN AUTHENTICATION) 32
2.2.1 Önlemler 33
2.3 HASSAS BİLGİ SIZINTISI (SENSİTİVE DATA EXPOSURE) 34
2.3.1 Önlemler 34
2.3.2 SSL/TLS'in Önemi 35
2.4 HARİCİ XML VARLIKLARI (XML EXTERNAL ENTITIES) 39
2.4.1 Önlemler 41
2.5 HATALI ERİŞİM KONTROLÜ (BROKEN ACCESS CONTROL) 41
2.5.1 Önlemler 42
2.6 YANLIŞ GÜVENLİK YAPILANDIRMASI (SECURITY MISCONFIGURATION) 42
2.6.1 Önlemler 43
2.7 SİTELER ARASI BETİK ÇALIŞTIRMA (CROSS-SITE SCRIPTING) 44
2.7.1 Reflected XSS 45
2.7.2 Stored XSS 47
2.7.3 Önlemler 51
2.8 GÜVENSİZ SERİLEŞTİRME (INSECURE DESERIALIZATION) 54
2.8.1 Önlemler 55
2.9 BİLİNEN ZAFİYETLERE SAHİP BİLEŞENLERİ KULLANMA 55
2.9.1 Önlemler 55
2.10 YETERSİZ LOGLAMA VE İZLEME 56
2.10.1 Önlemler 56
2.11 KABA KUVVET SALDIRISI (BRUTE FORCE ATTACK) 57
2.11.1 Önlemler 59
2.12 KISITSIZ DOSYA YÜKLEME (UNRESTRICTED FILE UPLOAD) 60
2.12.1 Önlemler 61
2.13 SİTELER ARASI İSTEK SAHTECİLİĞİ (CROSS SITE REQUEST FORGERY) 63
2.13.1 Önlemler 64
ÜÇÜNCÜ BÖLÜM 67
WEB UYGULAMALARINDA ZAFİYET TESPİT YÖNTEMLERİ 67
3.1 STATİK KOD ANALİZİ 67
3.1.1 Statik Kod Analizinde Kullanılan Araçlar 69

3.1.2 Statik Kod Analizinin Artı ve Eksileri 71
3.2 KOD GÖZDEN GEÇİRME 73
3.2.1 Kod Gözden Geçirme Çalışmasında Gerekli Denetimler 75
3.2.2 Kod Gözden Geçirme Çalışmalarının Artı ve Eksileri 78
3.3 SIZMA (PENETRASYON) TESTİ 79
3.3.1 Sızma Testi Türleri 81
3.3.2 Sızma Testinin İçermesi Gereken Metotlar 83
3.3.2 Sızma Testinde Risk Derecelendirme 86
3.3.3 Sızma Testinde Değerlendirme Raporu 89
3.3.4 Sızma Testlerinde Kullanılan Araçlar 90
3.3.5 Sızma Testi Artı ve Eksileri 95
3.4 ZAFİYET TARAMASI 96
3.4.1 Zafiyet Taraması'nda Kullanılan Araçlar 97
3.4.2 Zafiyet Taraması Artı ve Eksileri 97
DÖRDÜNCÜ BÖLÜM 99
WEB UYGULAMA GÜVENLİĞİ İLE İLGİLİ MEVZUAT 99
4.1 SİBER OLAYLARA MÜDAHALE EKİPLERİNİN KURULUŞ, GÖREV VE ÇALIŞMALARINA DAİR USUL VE ESASLAR HAKKINDA TEBLİĞ 100
4.2 KİŞİSEL VERİLERİN KORUNMASI KANUNU 102
BULGULAR 104
TARTIŞMA VE YORUM 107
SONUÇ 118
KAYNAKÇA 122
ÖZET 132
ABSTRACT 134
ÖZGEÇMİŞ 135

KISALTMALAR LİSTESİ

ASP	Etkin Sunusu Sayfaları (Active Server Page)
ASPX	Genişletilmiş Etkin Sunusu Sayfaları (Active Server Pages Extended)
CORS	Kökler Arası Kaynak Paylaşımı (Cross-Origin Resource Sharing)
CRUD	Ekle Oku Düzenle Sil (Create Read Update Delete)
CSRF	Siteler Arası İstek Sahteciliği (Cross-Site Request Forgery)
CSS	Basamaklı Stil Şablonları (Cascading Style Sheets)
CWE	Ortak Zayıflık Numaralandırması (Common Weakness Enumaration)
DB	Veri Tabanı (Database)
DDOS	Dağıtılmış Hizmet Reddi (Distributed Denial of Service)
DNS	Alan Adı Sistemi (Domain Name System)
DOM	Doküman Nesne Modeli (Document Object Model)
DOS	Hizmet Reddi (Denial of Service)
EF	Entity Framework
EXE	Çalıştırılabilir (Executable)
FTP	Dosya Transfer Protokolü (File Transfer Protocol)
GUI	Grafiksel Kullanıcı Arayüzü (Graphical User Interface)
GUID	Evrensel Benzersiz Tanımlayıcı (Universally Unique Identifier)
HSTS	HTTP Sıkı Taşıma Güvenliği (HTTP Strict Transport Security)
HTML	Hiper Metin İşaretleme Dili (HyperText Markup Language)
HTTP	Üstmetin Transfer Protokolü (Hyper-Text Transfer Protocol)
HTTPS	Güvenli Üstmetin Transfer Protokolü (HTTP Secure)
IBM	International Business Machines
ID	Kimlik (Identity)
IDE	Entegre Geliştirme Ortamı (Integrated Development Environment)
IIS	İnternet Bilgi Servisi (Internet Information Services)

IP	İnternet Protokolü (Internet Protocol)
JS	JavaScript
MD5	Mesaj Özeti Algoritması 5 (Message-Digest Algorithm 5)
MVC	Model-Görünüm-Denetleyici (Model-View-Controller)
OS	İştetim Sistemi (Operating System)
OWASP	Açık Web Uygulama Güvenliği Projesi
PHP	Hiper Metin Önişlemcisi (Hypertext Preprocessor)
PT	Sızma Testi (Penetration Test)
S	Sayfa
SHA	Güvenli Hash Algoritması (Secure Hash Algorithm)
SOME	Siber Olaylara Müdahale Ekibi
SQL	Yapılandırılmış Sorgu Dili (Structured Query Language)
SSL	Güvenli Yuva Katmanı (Secure Sockets Layer)
TCP	Geçiş Kontrol Protokolü (Transmission Control Protocol)
TLS	Taşıma Katmanı Güvenliği (Transport Layer Security)
USOM	Ulusal Siber Olaylara Müdahale Merkezi
VS	Visual Studio
WASC	Web Uygulama Güvenliği Konsorsiyumu
WWW	Dünya Çapında Ağ (World Wide Web)
XML	Genişletilebilir İşaretleme Dili (Extensible Markup Language)
XSS	Siteler Arası Betik Çalıştırma (Cross-Site Scripting)
XXE	XML Harici Varlık Saldırısı (XML External Entity Attack)

ŞEKİLLER ve TABLOLAR LİSTESİ

Şekil 0.1: 2019-2020 Web Saldırılarının Karşılaştırılması (CDNetworks) 3
Şekil 0.2: Tiplerine Göre Web Saldırılarının Karşılaştırılması (CDNetworks) 3
Şekil 1.1: Örnek Veri tabanı Tablosu 9
Şekil 1.2: Uygulamanın Veri tabanıyla İlişkisini Sağlayan Model Sınıfı 10
Şekil 1.3: Model katmanını karşılayan kod bloğu 11
Şekil 1.4: Tablodan Gelen Verilerin Görsel Olarak Listeleyen Kod Bloğu 11
Şekil 1.5: Microsoft IIS Web Sunucusu Genel Görünümü 17
Şekil 1.6: SSL/TLS Sertifikasına Sahip Web Sitesinin Görünümü 19
Şekil 1.7: Örnek SQL Sorgusu 21
Şekil 1.8: Örnek HTML Kodu 22
Şekil 1.9: Kullanıcı Listesini Taşıyan Bir XML Kodu 23
Şekil 1.10: SHA256 Hash Hesaplama Fonksiyonu 24
Şekil 1.11: Google.com DNS A Kaydı 26
Şekil 2.1: Kullanıcı Giriş Ekranına Yazılan SQL Enjeksiyon Kodu 29
Şekil 2.2: EF ile Çalışan Uygulamada SQL Enjeksiyonu Denemesi 32
Şekil 2.3: SHA256 Hash Algoritmasıyla Yapılan Şifreleme 35
Şekil 2.4: Uygulamanın HTTP Protokolüyle Çalıştırılması 36
Şekil 2.5: Güvensiz Siteye Ait Paketin İçeriği 37
Şekil 2.6: HTTPS ile Yapılan Bağlantının Görüntüsü 38
Şekil 2.7: HTTPS Protokolü Üzerinden Gönderilen Paketin İçeriği 39
Şekil 2.8: ASP.NET Varsayılan Hata Sayfasında Kritik Bilgilerin İfşa Olması 43
Şekil 2.9: Özel Oluşturulmuş Hata Sayfası 44
Şekil 2.10: XSS Uygulaması İsim Giriş Ekranı 45
Şekil 2.11: XSS Uygulaması Yönlendirilen Hoşgeldiniz Sayfası 46
Şekil 2.12: Input Alanına Girilmiş XSS Saldırısı Script Kodu 46
Şekil 2.13: Yönlendirilen Sayfada Alert Fonksiyonunun Çalışması 47
Şekil 2.14: XSS Uygulamasında Yorumların Listelendiği Sayfa 48
Şekil 2.15: XSS Uygulamasında Yorum Giriş Alanına Yazılmış Saldırı Kodu 49
Şekil 2.16: XSS Uygulamasında Yorumların Saklandığı Veri tabanı Tablosu 49
Şekil 2.17: Yorumların Listelendiği Sayfanın XSS Saldırısı Sonrası Görüntüsü 50
Şekil 2.18: XSS'i Engelleyen Öznitelik Eklenen Metot 51
Şekil 2.19: XSS Saldırı Kodu Sonucu Yönlendirilen Hata Sayfası 52
Şekil 2.20: Yorum Listeleme Sayfası Front-End Kodları 53
Şekil 2.21: Serialization ve Deserialization İşlemlerini Gösteren Grafik 54
Şekil 2.22: Brute Force Denemelerinin WireShark ile Tespiti 59
Şekil 2.23: Örnek Captcha Uygulaması 60
Şekil 2.24: Shell Yazılımının Web Tarayıcıda Çalıştırılması 61
Şekil 2.25: Dosya Uzantısını Kontrol Eden Kod Blokları 62
Şekil 2.26: CSRF Saldırısı Sonrası Oluşan Hata Sayfası 66
Şekil 3.1: VS Üzerinde Çalışan CodeRush Kod Analiz Aracı 68
Şekil 3.2: Zenmap GUI Ekran Görüntüsü 91
Şekil 3.3: WireShark TLS Trafiği İncelemesi 92
Şekil 3.4: Metasploit Kullanıcı Arayüzü 93

Şekil 3.4: Nessus Kullanıcı Arayüzü 94
Şekil 3.5: Burp Suite Kullanıcı Arayüzü 95
Şekil 4.1: Kurumsal SOME'lerin Görevlerini Gösteren Şema 101

Tablo 1: Zafiyet Önlem Etkinliği Bulguları..... 104
Tablo 2: Zafiyet Tespit Çalışmaları Karşılaştırması 105

GİRİŞ

Günümüzde 7.8 milyarı bulan dünya nüfusunun 4.6 milyarı aktif internet kullanıcısıdır[1] ve dünyanın hemen her yerinde insanlar, şirketler ve devletler arasındaki bilgi akışının büyük kısmı internet üzerinden sağlanmaktadır. İnternet adı verilen devasa ağın gerçek kapasitesinin anlaşılmasında ve dünya genelinde tetiklediği değişimde web uygulamalarının ortaya çıkışı çok önemli bir yer tutmaktadır.[2]

Web uygulamaları ilk ortaya çıktıkları dönemden günümüze kadar çok ciddi gelişmeler göstermiş ve insan hayatında vazgeçilmez bir yere sahip olmuşlardır. Finans ve alışveriş gibi sektörlerde her gün web uygulamaları üzerinden milyarlarca dolarlık işlem gerçekleştirilmektedir. Medya sektörünün önemli bir kısmı günümüzde internet üzerinden hizmet vermeye başlamıştır. Bu durum web uygulamalarının ne denli yaygın ve vazgeçilmez hâle geldiğini göstermektedir.

Bir web sunucusunda işletilen ve bu sunucunun da bağlı olduğu bir ağda hizmet veren yazılım uygulamaları "Web Uygulaması" olarak adlandırılmaktadır. Web uygulamaları istemciler tarafından "Web Tarayıcı" (Browser) aracılığıyla kullanılabilmektedir. Web tarayıcılar, sunucuda bulunan uygulamanın gönderdiği bir yazılım kodunu okumakta ve bunu istemcinin anlayabileceği bir içeriğe dönüştürmektedir. Bu işlem günlük hayatta internet kullanırken sıkça karşılaşılan HTTP veya onun şifrelenmiş versiyonu olan HTTPS protokolü tarafından sağlanmaktadır.

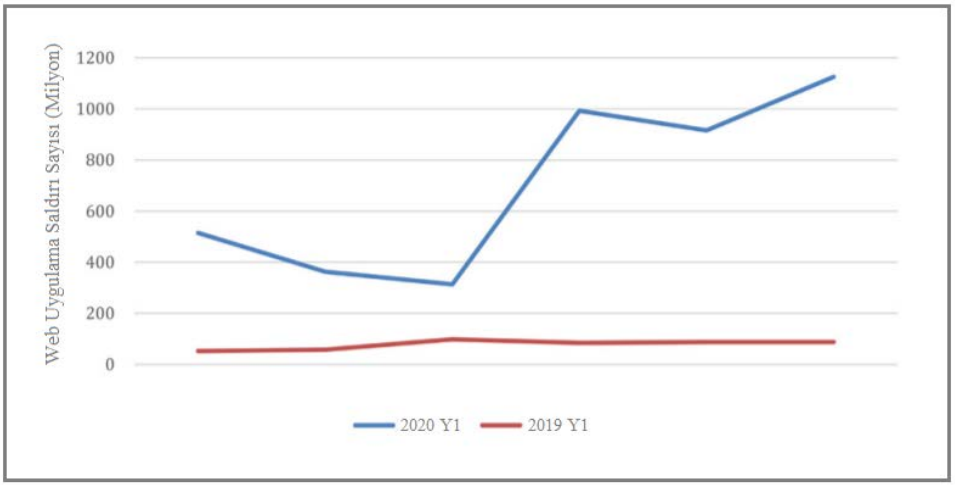
[1] Global Digital Population as of January 2021, https.//www.statista.com/statistics/617136/digital-population-worldwide (28.01.2021)
[2] What Is Web Applıcatıon Securıty And Why Is It Important?, https.//blog.eccouncil.org/what-is-web-application-security-and-why-is-it-important (28.01.2021)

Sunucuda çalışması, internet veya intranet gibi bir ağ aracılığıyla çok sayıda kullanıcıya hizmet verebilmesi, istemcinin cihazına herhangi bir kurulum gerektirmemesi; web uygulamalarını, geleneksel masaüstü uygulamalardan daha kullanışlı kılmaktadır. Bu durum da web uygulamalarının giderek daha fazla tercih edilmesini sağlamaktadır. Hemen hemen tüm iş sektörleri tarafından kullanılan web uygulamaları, bunun haricinde sosyalleşme ve bilgi paylaşımının da en önemli merkezi olmuştur ve milyarlarca insanın kişisel verileri bu uygulamalarda saklanmakta ve işlenmektedir.

Masaüstü ve konsol uygulamalarından (Örneğin Windows tabanlı .exe uygulamaları) farklı olarak; web tabanlı uygulamaların özellikle internet ağında yayımlandıkları taktirde tüm dünya tarafından erişilebilir olması, ilgili uygulamanın maruz kalabileceği saldırıların sayısının da artması anlamına gelmektedir. Bu saldırılar sonucu maddi ve manevi pek çok zarar meydana gelebilmektedir.

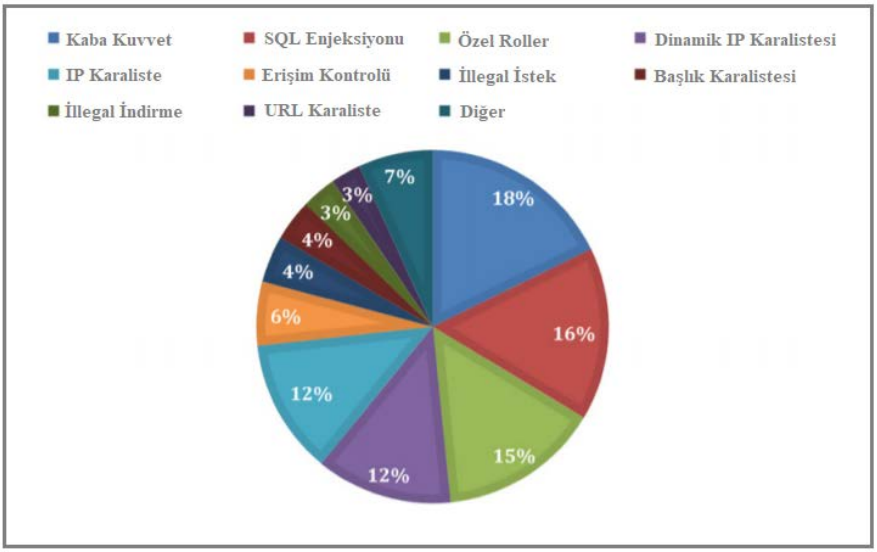
İnternet içerik hizmetleri alanında önemli bir yeri bulunan CDNetworks şirketi tarafından yayınlanan "State of the Web Security for H1 2020" raporuna[3] göre 2020 yılının ilk yarısında 4.2 milyar web uygulama saldırısı meydana gelmiştir ve bu rakam, önceki yılın aynı dönemine göre saldırılarda %800 artış anlamına gelmektedir. Bu artışta 2020 yılında meydana gelen salgının da etkili olduğu düşünülmektedir. Bu da olağanüstü dönemlerde saldırı sayısının arttığına işaret etmektedir. Şekil 1'deki grafikte 2019 ve 2020 yıllarının ilk yarılarında meydana gelen saldılar gösterilmiştir.

[3] CDNetworks Inc., State of the Web Security for H1 2020, https.//www.cdnetworks.com/download-assets/?CDNetworks-State-Of-web-Security-H1-2020 (29.01.2021)



Şekil 0.1: 2019-2020 Web Saldırılarının Karşılaştırılması (CDNetworks)

Güvenlik açıklarının ve saldırıların türü web uygulamasının içeriğine ve özelliklerine göre değişebilmektedir. Örneğin bir finans kuruluşunun web sitesinde veya kişisel hesapların değerli olduğu bir web sitesinde en çok karşılaşılan saldırı türü yetersiz yetkilendirme ayarlarından faydalanılarak yapılan saldırılardır[4]. Şekil 2'deki grafikte saldırı türleri ve saldırı sayıları arasındaki ilişki gösterilmiştir.



Şekil 0.2: Tiplerine Göre Web Saldırılarının Karşılaştırılması (CDNetworks)

[4] Financial Application Vulnerabilities, https.//www.ptsecurity.com/ww-en/analytics/financial-application-vulnerabilities (29.01.2021)

CDNetworks'ün raporunda dikkat çeken bir diğer analiz ise, bilindik saldırı taktiklerine karşı alınan önlemler sonrası saldırganların yapay zekâ gibi en yeni teknolojileri de kendi saldırı yöntemleri arasına dahil etmesidir. Makine öğrenmesi metodunun yardımıyla saldırgan, bir veri tabanının kurallarını ve hedef uygulamanın koruma stratejilerini kendi yazılımına öğretip yeni saldırı yöntemleri geliştirebilmektedir.

Diğer disiplinlerle karşılaştırıldığında web uygulama güvenliğinin nispeten yeni bir alan olduğu söylenebilir. Bu alanda ilk sistematik çalışmaları başlatmış topluluk olan Open Web Application Security Project (OWASP) 2001 yılında kurulmuştur[5]. OWASP web uygulama güvenliği ile ilgili raporlar yayınlamakta, saldırı türleri üzerine analizler yapmakta ve saldırılara karşı alınacak önlemler konusunda dokümanlar üretmektedir. OWASP hâlihazırda web uygulama güvenliği alanında en önemli referans olmayı sürdürmektedir.

İnternetin gelişim süreci içerisinde siber saldırılar her düzeyden uygulamayı tehdit etmektedir. 2014 yılında meydana gelen ve dünya üzerindeki en büyük teknoloji firmalarından olan Apple'ı hedef alan saldırganlar iCloud uygulamasının güvenliğini aşmayı başarmış ve çok sayıda kullanıcının kişisel verilerini ele geçirmişlerdir.[6] Bu durum en büyük bütçeli firmaların bile tehdit altında olduğuna bir örnek teşkil etmektedir. Bunun yanında 2007 yılında Estonya'nın maruz kaldığı siber saldırılar ise birçok kamu kurumunda ve özel sektörde hayatı durma noktasına getirmiştir. Estonya bu saldırılardan dolayı Rusya'yı sorumlu tutmuş ve olay bir devletin diğerine yönelik siber saldırısı kavramını ortaya çıkarmıştır.[7]

[5] About the OWASP Foundation, https.//owasp.org/about (29.01.2021)
[6] ICloud Leaks of Celebrity Photos, https.//en.wikipedia.org/wiki/ICloud_leaks_of_celebrity_photos (29.01.2021)
[7] Russia accused of unleashing cyberwar to disable Estonia (2007), https.//www.theguardian.com/world/2007/may/17/topstories3.russia (29.01.2021)

Tez çalışmasının birinci bölümünde, web uygulama teorisi ile giriş yapılmış ve web uygulamasının bileşenleri ile çalışma prensipleri açıklanmıştır. Ardından web uygulamalarındaki güvenlik açıklarının oluşturduğu potansiyel riskler ele alınmış ve son olarak web uygulama güvenliği ile ilgili temel kavramlar açıklanmıştır.

İkinci bölümde, web uygulama güvenliği alanında çalışmalar yapan OWASP isimli kuruluşun belirlediği en riskli 10 adet güvenlik açığı ve buna ek olarak yine yüksek risk taşıdığı değerlendirilen 3 adet güvenlik açığı ele alınmıştır. Bu açıkların oluşturduğu riskler ve bu açıklara karşı alınacak önlemlerin etkinliği tez çalışmasına özgü olarak yazılan web uygulamaları üzerinden incelenmiştir.

Üçüncü bölümde, güvenlik açıklarının tespitinin sağlanması için geliştirilen zafiyet tespit ve analiz çalışmalarının türleri ve bunların kapsamları incelenmiştir. Bu çalışmaların olumlu ve olumsuz yanları belirtilmiştir.

Dördüncü bölümde, web uygulama güvenliğini ilgilendiren mevzuat maddeleri taranmıştır. Bir zafiyetin ortaya çıkması veya bir saldırıya maruz kalınması durumunda kurum yetkililerinin yapması gerekenler belirtilmiştir. Bir saldırı sonucu oluşacak hasarın, yetkililerle ilgili ne gibi hukuki sorumluluk doğurduğu incelenmiştir.

Bulgular kısmında, ikinci ve üçüncü bölümlerde elde edilen bulgular tablolarla gösterilmiştir. Tartışma ve yorum kısmında, elde edilen bulguların değerlendirilmesi yapılmış ve kurumlar için yol haritaları sunulmuştur. Sonuç ve öneriler kısmında, tez çalışmasının genel değerlendirilmesi ve ulaşılan netice açıklanmıştır.

TEZİN KONUSU

Bu tez çalışmasında web sunucusu üzerinden hizmet veren yazılım uygulamalarında bulunan güvenlik zafiyeti türlerinin belirlenmesi ve sınıflandırılması,

bu zafiyet türlerinin önlenmesi için çözüm yöntemleri ve uygulamaların zafiyetlerinin tespitine yönelik çalışmalar işlenecektir. Kamu kurumlarının mevzuattan doğan yükümlülükleri ve web uygulama güvenliğinin sağlanması için Radyo ve Televizyon Üst Kurulu ve diğer kamu kurumlarının, yazılım birimi personelleri tarafından izlenmesi gereken yol haritası önerisi ile konu sınırlandırılacaktır.

TEZİN KAPSAMI

Web uygulama güvenliği üzerine uzmanlaşmış kuruluşlar tarafından en ciddi tehlikeyi arz ettiği iddia edilen 13 güvenlik açığının çalışma prensibi ve bu güvenlik açıklarını önlemek için literatürde önerilen yöntemlerin etkinliği, tez çalışmasına özgü olarak yazılmış web uygulamaları üzerinden incelenmiştir. Güvenlik açıklarının tespitinde kullanılan 4 farklı zafiyet tespit çalışması, hangi çalışmaların kurum içi kaynaklarca, hangilerinin de kurum dışı hizmet alımlarıyla yapılabileceği değerlendirilmiştir. Bir siber saldırı sonucu web uygulamada bulunan kişisel verilerde oluşan sızıntıda, ilgili web uygulama sorumlusuna mevzuatta ne gibi yaptırımların öngörüldüğü araştırılmıştır.

TEZİN AMACI

Bu tez çalışmasının amacı web uygulama güvenliğini tehdit eden durumların araştırılması, çözüm yöntemlerinin etkinliğinin incelenmesi, zafiyet test yöntemlerinin karşılaştırılmasının yapılması, mevzuatta web uygulama işleticilerine yüklenen sorumluluğun araştırılması ve kurumların, web uygulama güvenliğini sağlamak için takip etmesi gereken en ideal olduğu değerlendirilen yolun önerilmesidir.

TEZİN ÖNEMİ

Web Uygulama zafiyetlerinin tespit yöntemlerinin ve alınması gereken önlemlerin araştırılması kurumların web uygulama güvenliği yol haritasının

belirlenmesinde büyük öneme sahip. Bu konuda doğru hizmet alımı kararlarının verilebilmesi için hangi zafiyet tespit çalışmalarının ilgili kuruma uygun olduğu araştırılarak belirlenmelidir. Bu tez web uygulama işleticilerinin, özellikle Radyo ve Televizyon Üst Kurulunun ve diğer kamu kurumlarının bu kararlarını kolaylaştıracak bir yol haritası sunacaktır.

TEZİN YÖNTEMİ

Tez çalışması hazırlanırken, her türlü Web Uygulama için güvenliğin sağlanmasının çeşitli çalışmalar gerektirdiği varsayılarak, web uygulamalarında en yüksek risk arz eden güvenlik açıklarının neler olduğu, bu açıklara ne gibi çözüm yöntemlerinin önerildiği ve web uygulama güvenliğini sağlama yolunda hangi zafiyet tespit yöntemlerinin kullanıldığı ile ilgili "literatür taraması" yapılmıştır. Zafiyet tespit yöntemleri üzerinde bir "karşılaştırmalı analiz" çalışması yürütülmüştür. Tez çalışması için yazılan web uygulamaları ile "deney yöntemi" kullanılarak güvenlik zafiyetlerinin ne gibi sonuçlar doğurduğu ve alınan önlemlerin sorunu nasıl çözdüğünü gösteren pratik uygulamalarla veri toplanmıştır. Web uygulama güvenliğini ilgilendiren mevzuat incelenmiş ve web uygulama yöneticilerinin hukuki sorumluluğu belirlenmiştir.

TEZİN SINIRLILIKLARI

Tez çalışmasında güvenlik açıkları ve önlemler incelenirken yapılan pratik uygulama çalışması, teze özgü olarak test amaçlı geliştirilen uygulamalar ile sınırlandırılmıştır. İnternet ortamında canlı yayında bulunan kurumsal uygulamalara ait güvenlik seviyesinin kamuya açık bir çalışmada belirtilmesinin uygun olmayacağı değerlendirilmiştir.

BİRİNCİ BÖLÜM

WEB UYGULAMA GÜVENLİĞİNİN TEMELLERİ

Bilgi güvenliğinin bir alt branşı olarak değerlendirilen web uygulama güvenliği her türlü web tabanlı uygulamanın temel bileşenlerinden birisidir. Kurum içi intranet ağlarına kıyasla internet ağı denilen ortam doğası gereği web uygulamaları için çok daha tehlikeli bir ortamdır ve saldırıların çok büyük bir kısmı bu ortamda gerçekleşmektedir. Web uygulaması olarak genelleyebileceğimiz web siteleri, web servisleri ve web tabanlı işlemlerin yapıldığı her türlü yazılımı güvenlik açıklarından ve saldırılardan korumak için geliştirilen disipline web uygulama güvenliği denmektedir.

Tez çalışmasının bu bölümünde öncelikle web uygulamalarının ne olduğu, nasıl çalıştığı, hangi yeteneklere sahip olduğu detaylı olarak incelenecek ve web uygulamalarında güvenlik açıklarının türleri açıklanacaktır.

1.1 WEB UYGULAMA TEORİSİ

Web uygulaması en kısa tanımıyla bir ağ üzerinde çalışan ve bir istemcinin ilgili ağ üzerinden erişim sağlayarak işlemini gerçekleştirebildiği yazılımdır. Web uygulama yazılımı diğer tüm yazılımlar gibi bir programla dili ile kodlanır. Web Uygulamalarında kullanılan en popüler programlama dilleri arasında PHP, Javascript, C#, Java, Python gibi diller yer almaktadır. Bu dillerle oluşturulmuş çeşitli Framework (Yazılım İskeleti) ve Library (Yazılım Kütüphanesi) türleri de yaygınca kullanılmakta olup, bunların en popüler olanları arasında ASP.NET, ReactJS, Django, Laravel, Angular gibi teknolojiler yer almaktadır.

Bir web uygulama yazılımı genellikle arka yüz (back-end), ön yüz (front-end) ve veri tabanı olarak üç yapıdan oluşmaktadır. İstemcinin yani son kullanıcının görmediği ve çekirdek yazılım diyebileceğimiz yapıya arka yüz denmektedir. Ön yüz ise web uygulamalarında, özellikle de web sitelerinde son kullanıcının internet tarayıcısı (browser) ile görüntüleyebildiği kısmın tasarlandığı görsel yapıdır. ön yüz programlanırken HTML, CSS ve Javascript dilleri kullanılmaktadır. Bu dillerle istemciye sunulan ve istemcinin etkileşime girebileceği kısımlar kodlanır. Veri tabanı ise uygulamanın mevcut verilerinin depolandığı ortam olarak anlaşılabilir. Arka yüz yapısı veri tabanı ile iletişime geçmekte ve veri tabanındaki kayıtlı bilgileri okuyup ön yüz yapısına iletmektedir ve ön yüz üzerinde bu bilgiler kullanıcıya sunulmaktadır. Kullanıcının etkileşimi sonucu yeni bir veri gelmişse, bu veriler ön yüz yapısı üzerinden arka yüz yapısına aktarılmakta ve arka yüz yeniden veri tabanı ile iletişim kurarak bu bilgileri veri tabanına yazmaktadır. Yani arka yüz bileşeni; veri tabanı ile ön yüz arasında bir köprü vazifesi görmektedir.

	ID	Adi	Silindi
1	1	Kitap	0
2	2	Dergi	0
3	6	Makale	0
4	9	Gazete	0
5	10	Rapor	0
6	11	Sözlük	0

Şekil 1.1: Örnek Veri tabanı Tablosu

Şekil 1.1’de örnek bir e-kitap web uygulamasının veri tabanında kitap türlerinin tutulduğu veri tabanı tablosu görülmektedir. “ID”, “Adi” ve “Silindi” alanları kolonları temsil eden isimlerdir ve veri tabanındaki toplam 6 adet kayıt gözükmektedir. Bu kayıtlar web uygulamasının arka yüz kodları tarafından veri tabanına bağlantı yapılarak okunur ve uygulama tarafından tablo üzerinde düzenle,

silme ve yeni kayıt ekleme gibi işlemler yapılabilir. Bu işlemler yazılım dilinde CRUD (Create Read Update Delete) işlemleri olarak isimlendirilir[8].

```
[Table("Tur")]
public partial class Tur
{

    public Tur()
    {
        Yayin = new HashSet<Yayin>();
    }

    public int ID { get; set; }

    [Required]
    [StringLength(50)]
    public string Adi { get; set; }

    public bool Silindi { get; set; }


    public virtual ICollection<Yayin> Yayin { get; set; }
}
```

Şekil 1.2: Uygulamanın Veri tabanıyla İlişkisini Sağlayan Model Sınıfı

Şekil 1.2'de bir önceki şekilde gösterilen veri tabanı tablosuyla ilişkinin sağlanabilmesi için Entity Framework teknolojisi kullanılarak C# dilinde yazılmış arka yüz kod bloğu gösterilmektedir. Bu kod bloğunda veri tabanındaki "ID", "Adi" ve "Silindi" kolonlarının karşılıkları bulunmakta ve bu sınıf uygulamamızın veri tabanıyla bağlantısında bir harita (mapping) görevi görmektedir. Hangi veri tabanı ile ilişkili olduğu kod bloğunun yukarısındaki "Table" isimli bir öznitelik (attribute) ile belirtilmektedir.

[8] What are CRUD Operations. How CRUD Operations Work, Examples, Tutorials & More, https.//stackify.com/what-are-crud-operations/ (02.03.2021)

```
public ActionResult KitapTurleri()
{
    return View(db.Tur.Where(x => x.Silindi == false).ToList());
}
```

Şekil 1.3: Model katmanını karşılayan kod bloğu

Şekil 1.3'te model katmanı vasıtasıyla veri tabanına bağlantının yapılıp kitap türlerinin çekilmesini sağlayan arka yüz kod bloğu gösterilmektedir. İlgili kod bloğu "Tur" isimli model sınıfının içinde "Silindi" değeri "false" olan tüm verileri çekmekte ve "return View" komutuyla ilgili verileri HTML kodunun oluşturulduğu "KitapTurleri View" isimli ön yüz bileşenine aktarmaktadır.

```
@model IEnumerable<Ekitap.Models.Tur>

<div class="container">
    <h3 class="float-left">Tür Listesi</h3>
    <br /><br />
    <table class="table table-borderless table-bordered table-hover">
        <thead>
            <tr>
                <th>Tür Adı</th>
                <th></th>
                <th></th>
            </tr>
        </thead>

        <tbody>
            @foreach (var item in Model.OrderBy(x => x.Adi))
            {
                <tr>
                    <td>
                        @item.Adi
                    </td>
```

Şekil 1.4: Tablodan Gelen Verilerin Görsel Olarak Listeleyen Kod Bloğu

Şekil 1.4'te veri tabanındaki ilgili tablodan gelen verilerin kullanıcıya göstermek üzere basıldığı arka yüz kod bloğu görülmektedir. Bu kısım HTML kod üreterek Web Tarayıcının gösterebileceği bir görselin üretilmesini sağlar. İlgili kod

bloğunda HTML kodların için arka yüz bileşeninden gelen veriler basmak için Razor Kütüphanesi kullanılmıştır. Razor kodları şekilde arka planı gri renkte gösterilmiştir.

Hangi programlama diliyle yazılırsa yazılsın ön yüz kısmında HTML çıktısını manipüle etmek için bu tip bir kodlama metodu kullanılmaktadır. Bu işlem ön yüzün içinde arka yüz ile haberleşen ara bileşen olarak tanımlanabilir.

Günümüzde web uygulamalarında kullanılan popüler 10 yazılım iskeleti (framework) ve kullandıkları programlama dilleri aşağıdaki gibidir[9]:

- Ruby On Rails (Java)
- Django (Python)
- Angular-AngularJS (Javascript)
- ASP.NET (C#)
- METEOR (Javascript)
- Laravel (PHP)
- Express (Javascript)
- Spring (Java)
- PLAY (Scala ve Java)
- CodeIgniter (PHP)

Web uygulama teorisindeki önemli yer tutan bazı kavramlardan aşağıda maddeler hâlinde bahsedilecektir[10]:

[9] Top Ten Framework for Web Applications, https.//www.geeksforgeeks.org/top-10-frameworks-for-web-applications (23.02.2021)
[10] Doaa Nabil, Abeer Mosad, Hesham A. Hefny, *Web-Based Applications quality factors. A survey and a proposed conceptual model, Egyptian Informatics Journal*, Volume 12, Issue 3, 2011, s. 211-217

- Kalite: Uygulamanın hızlı, güvenilir, hatasız ve kolay kullanılabilir olmasını ifade eder.
- Sürdürülebilirlik: Uygulamanın testlerinin yapılmış olmasını, uygulama tasarımının ve uygulama bileşenlerinin dokümante edilmiş olmasını, uygulamanın modüler ve kurallı bir yapıya sahip olmasını ifade eder.
- Sınırlamalar: Uygulamanın maliyetini, geliştirme süresini ve risk öngörüsünü ifade eder.

1.2. GÜVENLİK AÇIĞI KAVRAMI VE OLUŞTURDUĞU RİSKLER

Web uygulama teorisine göre kalite bileşeni içerisinde uygulama güvenliği kavramı önemli bir yer tutmaktadır. Web uygulamalarında güvenlik şu şekilde tanımlanmaktadır: Uygulamaya ait verilerin, bilgilerin ve işlemlerin; yetkisiz kişilerin erişimine ve müdahalesine kapalı olması durumu[11]. Güvenlik açığı ise bu verilerin, bilgilerin ve işlemlerin korunmasında bir zafiyetin meydana gelmesi sonucu yetkisiz kişiler tarafından erişilebilme tehlikesini ifade eder.

Bu bölümde web uygulamalarında bulunan güvenlik açıklarından yararlanarak yapılacak saldırıların uygulamada ne gibi hasarlara yol açacağı üzerinde durulacaktır. Bir güvenlik açığı saldırganlar tarafından fark edildiğinde ve siber saldırı gerçekleştirildiğinde, web uygulamasının dolayısıyla ilgili kurumun ve bunun yanında uygulamaya kullanan istemciler bundan çeşitli şekillerde etkilenmesi kaçınılmazdır. Saldırı sonucu web uygulamasında meydana gelebilecek hasarları dört başlıkta inceleyeceğiz.

[11] D. Nabil, A. Mosad (2011), s. 211-217

1.2.1 Veri Kaybı

Bir güvenlik açığının sonucu gerçekleşen siber saldırı web uygulamasında veri kaybına sebebiyet verebilir. Bir tetikleyici vasıtasıyla küçük bir veri kaybı yaşanabileceği gibi, güvenlik açığı saldırganın sunucunun dosya sistemine veya uygulamanın veri tabanına ulaşabilmesini (SQL Injection ve benzeri saldırı metotlarıyla) sağlayacak kadar kritik ise çok büyük miktarda verinin kaybıyla da sonuçlanabilir. Veri kaybının telafi edilmesi uygulama kullanıcısı veya firma için hem yüksek maliyetli hem de zahmetli olabileceği gibi kaybın telafisinin mümkün olmadığı durumlar da yaşanabilir. Verilerin sıkça yedeğinin alınması ve saldırının gerçekleştiği an ile son yedeğin alındığı anın arasının dar tutulması veri kaybının boyutunu azaltacaktır.

1.2.2 Veri Hırsızlığı

Veri kaybından daha ciddi sonuçlar doğurabilen veri hırsızlığı, bir siber saldırı sonucu kullanıcının veya şirketin verilerinin ele geçirilmesi olarak tanımlanabilir. Bir web uygulamasında tutulan bilgilerin, örneğin müşteri veri tabanındaki kişisel verilerin ya da daha tehlikelisi mali verilerin ele geçirilmesi çok ciddi sonuçlar doğurabilecektir. Uygulamaya ait veriler yetkisiz kişiler tarafından kopyalandığı taktirde bu durum veri sızıntısı olarak tanımlanır[12].

İnternet tarihinde kayda geçen en büyük veri hırsızlığı 2013 yılında Yahoo'nun uğradığı siber saldırı sonucunda gerçekleşmiştir. Yahoo yönetiminin "devlet destekli aktörler" tarafından yapıldığını iddia ettiği saldırılar sonucunda yaklaşık 3 milyar kullanıcı hesabının bilgileri veri hırsızlığı kurbanı olmuştur[13].

[12] Deshpande Varun M, Nair Mydhili K., ShahDhrumil (2017), Major Web Application Threats for Data Privacy & SecurityDetection, Analysis and Mitigation Strategies, *International Journal of Scientific Research in Science and Technology, Sayı: 3, No:7, s. 186*

[13] The 15 biggest data breaches of the 21st century (2021), https.//www.csoonline.com/article/2130877/the-biggest-data-breaches-of-the-21st-century.html (23.01.2021)

1.2.3 Yetkisiz Veri Girişi (Veri Enjeksiyonu)

Siber saldırı sonucu yaşanabilecek bir diğer istenmeyen durum ise web uygulamaya yetkisiz veri girişi yapılmasıdır. Uygulama sunucusunun dosya sistemini veya uygulamanın yönetici panelini ele geçiren saldırgan uygulamaya istediği bir içeriği enjekte edebilir. Örneğin bir kamu kurumunun web sitesine yasadışı bir örgütün posterlerini ekleyip o kurumun ve devletin itibarını sarsabilir. Bununla birlikte siteye eklediği virüs ve benzeri yazılımlarla o web uygulamasına bağlanan çok sayıda internet kullanıcısına da zarar verebilir.

Veri enjeksiyonuna sebep olan saldırılardan SQL Enjeksiyonu, Siteler Arası Betik Çalıştırma Enjeksiyonu (XSS), Kod Enjeksiyonu ve benzeri bazı saldırılar çoğu web uygulaması için temel güvenlik problemleri arasındadır. [14] Ayrıca SQL ve XSS enjeksiyonları, veri enjeksiyonun daha büyük risklere de neden olmaktadır.

1.2.4 Hizmet Reddi

Hizmet reddi, web uygulamasının gelen isteklere cevap verememesi durumudur. İstemciler ilgili web sitesine bağlanmak istediklerinde "Sayfa görüntülenemiyor" benzeri uyarılarla karşılaşır. Genellikle DoS/DDoS saldırısı olarak anılan bir siber saldırı türüne maruz kalındıktan sonra yaşanan bir durumdur. Saldırı, web uygulamasının çalıştığı sunucunun kapasite limitlerinden faydalanarak gerçekleştirilir. İlgili web uygulamasına aynı anda çok sayıda istek gönderen saldırganlar, sunucunun sağlıklı hizmet vermesini engellemeye çalışır. Özellikle canlı olarak yapılan işlemlerden para kazanan web uygulamaları, örneğin alışveriş siteleri DDoS saldırılarının en çok hedef aldığı uygulamalardır.

[14] Z. Liu, J. Pen, "The Analysis of HTTPS Privacy Protection Based on Malicious Code Injection," *2020 IEEE 20th International Conference on Communication Technology (ICCT)*, Nanning, China, 2020, s. 1075-1082, doi: 10.1109/ICCT50939.2020.9295688.

Saldırının etkinliğini artırmak için saldırganlar kendileri dışındaki internet kullanıcılarını da bu saldırıya dahil etmek için çeşitli yöntemler geliştirmişlerdir. Bunların en ünlüsü "zombi ağı" olarak anılan yöntemdir. Zombi Ağı ünlü güvenlik ve bilişim kuruluşu Kaspersky tarafından şöyle tanımlanmaktadır[15]:

> Siber suçlu, hedeflenen kaynağa olağanüstü sayıda istek göndermek için genellikle virüs yaydığı bilgisayarlardan oluşan bir zombi ağı kurar. Suçlunun zombi ağındaki tüm virüslü bilgisayarların eylemlerini kontrol edebilmesi nedeniyle saldırının boyutu kurbanın web kaynaklarına aşırı yüklenilmesine yol açabilir.

1.3 TEMEL KAVRAMLAR VE ARAÇLAR

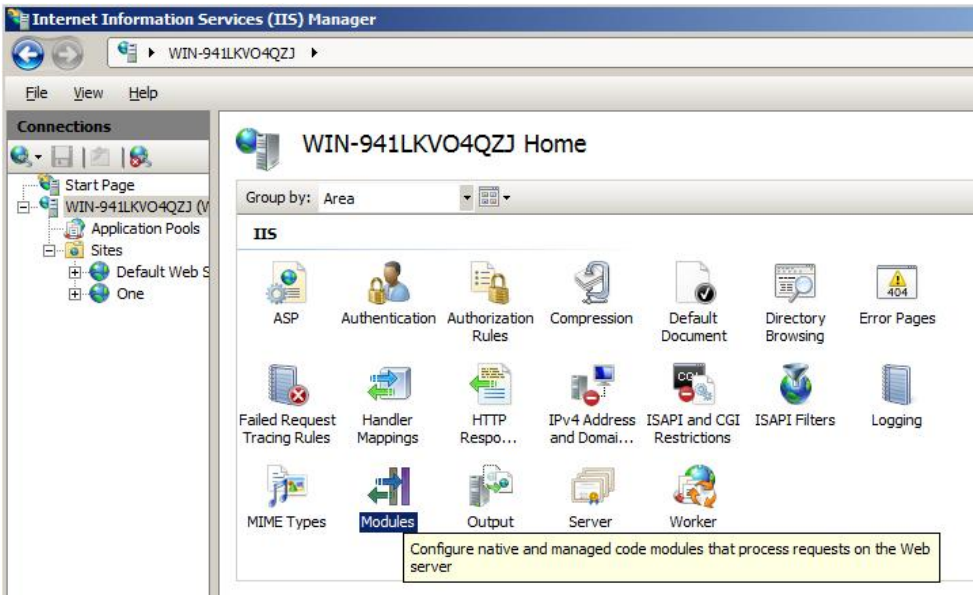
Web uygulama güvenliği söz konusu olduğunda en sık kullanılan temel kavramlardan ve bazı araçlardan bahsedeceğiz. Web uygulama güvenliğinin daha iyi anlaşılabilmesi için ön bilgi olarak bu kavramların anlaşılması faydalı olacaktır. Burada anlatılan kavram ve araçlar tez çalışmasının geri kalan bölümünde de sıkça anılacak ve üzerlerinde daha detaylı çalışmalar yapılacaktır.

1.3.1 Web Sunucusu (Web Server)

Web sunucusu, üzerinde web uygulamalarının ve web servislerinin çalıştığı bir yazılımdır ve bir bilgisayar üzerinde hizmet verir.[16] Bu bir kişisel bilgisayar olabileceği gibi, genellikle işlem gücü ve disk büyüklüğü açısından çok daha güçlü ve bu iş için özel üretilmiş bilgisayarlar web sunucusu olarak tercih edilmektedir. Web sunucusu bir iç ağ veya internet üzerinde web uygulamasının Barındırma (Hosting) hizmetini sunar.

[15] DDoS saldırısı nedir, www.kaspersky.com.tr/resource-center/threats/ddos-attacks (23.02.2021)
[16] Gray, Neil, *Web Server Programming*, Wiley, 2003, s.2

Windows, Linux, MacOS gibi işletim sistemleri üzerinde çalışan farklı web sunucuları destekledikleri programlama dillerine göre de çeşitlilik göstermektedir. En popüler web sunucularına Apache (açık kaynak kodlu) ve Şekil 1.5'te genel görünümü bulunan Microsoft Internet Information Services(IIS) örnek verilebilir[17].



Şekil 1.5: Microsoft IIS Web Sunucusu Genel Görünümü [18]

Günümüzde en popüler web sunucularından bazıları şu şekildedir:

- Apache (Açık kaynak kodlu bir web sunucusu)
- Microsoft Internet Information Services (IIS): Genellikle ASP.NET gibi Microsoft tabanlı web uygulamalarını çalıştırmak için kullanılır. Bunun yanında PHP tabanlı uygulamalar için de sıkça kullanılmaktadır.
- Tomcat JSP (Java programla dilinde yazılmış web uygulamaları için tasarlanmış bir web sunucusudur.)

[17] Robbins, Jennifer Niederst, *Learning Web Design*, O'REILLY, 2012, s.22
[18] IIS Modules Overview, https://docs.microsoft.com/en-us/iis/get-started/introduction-to-iis/iis-modules-overview (23.02.2021)

- LiteSpeed (Apache'ye alternatif olarak geliştirilmiş ve daha yüksek performanslı bir web sunucusudur.)
- Nginx (Düşük hafıza kullanımına odaklanmış bir web sunucusudur.)

1.3.2 HTTP

Açılımı Hyper Text Transfer Protocol olan HTTP, ağ katmanlarından "uygulama katmanı" seviyesinde çalışan bir protokoldür.[19] TCP/IP protokolünü baz alarak çalışan HTTP, Web'deki bir noktadan bir diğer noktaya verilerin (HTML kodu, resim, video, dosya vb.) dağıtılmasını sağlayarak, sunucu ile istemci arasında bir kurye görevi görür. Varsayılan olarak TCP 80 portunda çalışır.

HTTP 1990 yılından beri kullanılmaktadır. HTTP/0.9 sürümüyle ile başlamış ve son olarak HTTP/1.1 versiyonuna yükseltilmiştir[20].

1.3.3 HTTPS

HTTP'nin şifrelenmiş versiyonu olarak tanımlayabileceğimiz HTTPS, sunucu ve istemci arasında giden veri paketlerinin üçüncü bir taraf tarafından ele geçirilse bile okunamamasını sağlar. [21]

Örneğin bir internet kafeden bir bankanın web sitesine kullanıcı adı ve şifre yazarak bağlantı yapıldığında, bu veriyi taşıyan paketlerin internet kafenin modem/router cihazının yöneticisi tarafından, internet servis sağlayıcısı tarafından

[19] Thomas, Stephan. A. HTTP Essentials: Protocols For Secure, Scaleable, Web Sites, Wiley, 2001, s. 13
[20] Çontar, Furkan, Ağ ve Yazılım Güvenliği, KODLAB Yayınları, İstanbul 2017, s. 5
[21] F. Wibowo, H. H. Nuha and S. Wibowo, "Network Security Analysis Using HTTPS with SSL On General Election Quick Count Website," 2020 IEEE International Conference on Communication, Networks and Satellite (Comnetsat), Batam, Indonesia, 2020, s. 204-207, doi: 10.1109/Comnetsat50391.2020.9328940.

veya paketin geçtiği ağ yolundaki farklı bir saldırgan tarafından rahatlıkla ele geçirilebileceğini düşünebiliriz. Bu veri taşıma işlemi HTTP ile yapıldığında içinde bulunan şifre verisinin açık metin şeklinde okunabilmesi mümkündür ancak HTTPS bu veriyi şifreleyerek gönderdiğinden tehlike önlenmiş olur.

Bu sebeple günümüzde web sitelerinin çok büyük kısmı HTTPS protokolünü kullanır ve bu siteler TLS/SSL (Transport Layer Security / Secure Socket Layer) sertifikaları kullanarak şifrelemeyi sağlar. HTTPS varsayılan olarak TCP 443 portunda çalışır. Günümüzde internet ağında hizmet veren web sitelerinin %70.4'ü varsayılan olarak HTTPS üzerinden hizmet vermektedir.[22]

1.3.4 TLS/SSL Sertifikası

SSL (Secure Socket Layer) ve TLS (Transport Layer Security) en popüler HTTPS şifreleme protokolleridir. TLS protokolü SSL'in yeni versiyonu olup 1999 yılında kullanıma başlamasıyla SSL'in yerini almıştır.[23] Ancak günümüzde halen SSL ifadesi iki protokolü de tanımlamak için kullanılmaya devam etmektedir[24].



Şekil 1.6: SSL/TLS Sertifikasına Sahip Web Sitesinin Görünümü [25]

[22] Usage statistics of Default protocol https for websites, Erişim Tarihi: 23.02.2021. https.//w3techs.com/technologies/details/ce-httpsdefault
[23] Oppliger R. SSL and TLS : Theory and Practice. Vol Second edition. Artech House; 2016, s.21
[24] What is SSL, TLS and HTTPS, https.//www.websecurity.digicert.com/security-topics/what-is-ssl-tls-https (23.02.2021)
[25] What is an SSL, https.//www.znetlive.com/blog/what-is-an-ssl-certificate-and-why-is-it-a-must-have-for-every-website (13.03.2021)

Bir SSL/TLS sertifikasına sahip olan web siteleri, web tarayıcısında (Chrome, Firefox vb.) açıldığında link çubuğunun solunda Şekil 1.6'daki gibi bir anahtar işareti görülecektir. Bu işaret bağlantının HTTPS üzerinden sağlandığının sembolü olarak kullanılmaktadır.

1.3.5 Veri Tabanı

Veri tabanları, web uygulamalarının çok büyük kısmının kullandığı bir araçtır. Veri tabanlarında ilgili uygulamaya ait veriler bir sistematik düzen içerisinde saklanır ve ihtiyaç halinde uygulama tarafından okunur. Statik web siteleri veri tabanına ihtiyaç duymadan istemciye sabit bir içerik sunabilir ancak web sitesi dinamik hale gelip bir yönetim paneli aracılığıyla değiştirilebilecekse veya istemci tarafından web sitesine bazı girdiler gönderilecekse, veri tabanı bunları karşılamak için en sık kullanılan yöntemdir.[26]

Günümüzde en sık kullanılan veri tabanı uygulamaları sırasıyla aşağıdaki gibidir[27]:

- MySQL
- PostgreSQL
- Microsoft SQL Server
- SQLite
- MongoDB
- Redis
- MariaDB
- Oracle
- Elasticsearch
- Firebase

[26] Gehani, N., & Annamalai, M., The Database Book : Principles & Practice Using the Oracle Database: Vol. First edition. Silicon Press. 2011, s. 2
[27] Most Popular Databases in 2020 and New Trends, https.//www.eversql.com/most-popular-databases-in-2020 (24.02.2021)

- DynamoDB
- Cassandra
- IBM DB2
- Couchbase

1.3.6 SQL

Açılımı Structured Query Language olan SQL, veri tabanlarını düzenlemek için kullanılan bir sorgu dilidir. İlk olarak 1974 yılında IBM araştırmacıları tarafından geliştirilmiş eski bir dildir[28]. SQL vasıtasıyla veri tabanları üzerinde CRUD (ekleme, okuma, değiştirme ve silme) işlemleri gerçekleştirilebilir. SQL sorgusu, veri tabanı üzerinde yapılmak istenilen işlemin belirtildiği kod bloğudur.

```
SELECT * FROM Personeller WHERE Yas > 40
```

Şekil 1.7: Örnek SQL Sorgusu

Şekil 1.7'de gösterilen örnek SQL sorgusu ile veri tabanında bulunan "Personeller" isimli tablodaki kayıtlardan "Yaş" verisi 40'ın üzerinde olan tüm(*) kayıtlar seçilir.

1.3.7 HTML

Açılımı Hyper Text Markup Language olan HTML bir işaretleme dilidir. Bir web sitesinin web tarayıcıda nasıl sunulacağını gösteren en temel belirteçtir.[29] Çoğu

[28] Debarros, Anthony, *Practical SQL*, William Pollock, 2018, s. 6
[29] Cox, Vicki ve Reding, Eisner Elizabeth ve Wermers, Lynn, *HTML Illustrated Introductory*, Cengage Learning, 2006, s. 11

web sayfaları için bir standart dildir. Tüm web sayfalarının %86'sı HTML kullanır[30]. Çeşitli elementlerden ve etiketlerden meydana gelir ve genellikle CSS (Cascading Style Sheets) işaretleme diliyle birlikte kullanılır. CSS dili görsel tasarımı şekillendirir. Şekil 1.8'de örnek bir HTML kod bloğu gösterilmiştir.

Örnek.html

```
1  <!DOCTYPE html>
2  <html lang="tr">
3
4  <meta charset="utf-8">
5  <title>Sayfa Başlığı</title>
6
7  <body>
8    <h1>Metin Başlığı</h1>
9    <p>Birinci paragraf.</p>
10   <p>İkinci Paragraf.</p>
11 </body>
12
13 </html>
```

Şekil 1.8: Örnek HTML Kodu

1.3.8 Javascript

Web tarayıcıları tarafından okunabilen ve hemen hemen tüm web sitelerinde kullanılan bir betik (scripting) dilidir. Front-end dili olduğu için istemci tarafından tüm Javascript kodları görülebilir. Bu kodlar HTML dosyalarının içerisine yazılabileceği gibi düzenli olması açısından ".js" uzantılı dosyaların içine yazılması daha sık tercih edilir. Javascript diliyle yazılmış bir betik, diğer programlama dillerinde olduğu gibi

[30] Usage Statistics of HTML5 for Websites, https.//w3techs.com/technologies/details/ml-html5 (23.02.2021)

mantıksal ve matematiksel işlemleri yerine getirebilir, fonksiyonlar tanımlayabilir ve kullanıcıyla etkileşime geçebilir.[31]

1.3.9 XML

Açılımı Extensive Markup Language olan XML, verilerin saklanmasını ve taşınmasını sağlayan bir işaretleme dilidir ve yapı olarak HTML ile benzerlik göstermektedir.[32] XML ile yazılımcılar kendi etiketlerini oluşturabilir ve bu etiketlerin arasına eklenen veriler bir düzen içerisinde karşı tarafa aktarılabilir. XML dosyaları Şekil 1.9'da görüldüğü gibi. herhangi bir text editör ile düzenlenebilen basit metin dosyalarıdır. [33]

ornek.xml*

```xml
<?xml version="1.0" encoding="utf-8" ?>
<KullaniciListesi>

  <kullanici id="0">
    <kullaniciadi>KullaniciBir</kullaniciadi>
    <eposta>bir@ornek.com</eposta>
  </kullanici>

  <kullanici id="1">
    <kullaniciadi>KullaniciİKi</kullaniciadi>
    <eposta>iki@ornek.com</eposta>
  </kullanici>

  <kullanici id="2">
    <kullaniciadi>KullaniciUc</kullaniciadi>
    <eposta>uc@ornek.com</eposta>
  </kullanici>

</KullaniciListesi>
```

Şekil 1.9: Kullanıcı Listesini Taşıyan Bir XML Kodu

[31] *What is Javascript*, https.//developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript (02.03.2021)
[32] McKinnon, Linda, *XML in 60 Minutes a Day*, WILEY, 2003, s.14
[33] XML Overview, https.//www.tutorialspoint.com/xml/xml_overview.htm (03.03.2021)

Web Tarayıcıları XML Parser adı verilen bir araca sahiptirler ve bu sayede XML dosyalarını okuyabilir, ayrıştırabilir ve üzerinde işlem yapabilirler. XML DOM (Document Object Model) adı verilen Belge Nesne Modeli XML dosyasına erişimin ve üzerinde yapılacak değişikliklerin metotlarını belirler.

1.3.10 Hash

Hash, tek gidişli yazılım fonksiyonu olarak tanımlanmaktadır. Hash fonksiyonuna giren değer her zaman aynı boyuttaki bir değere dönüşür. Her çıkış değerinin kendine özgü (Unique) bir değer olması hedeflenir. Verilen veri hangi boyutta olursa olsun (1 MB veya 100 MB) çıkış uzunluğu sabittir. Hash fonksiyonlarının başarısı farklı giriş değerleri sonucunda oluşan çıkış değerlerinin çakışma oranı ile ölçülmektedir. Farklı giriş verileri fonksiyon sonucunda aynı çıkış değerine ulaşıyorsa bu durum bir çakışmadır ve fonksiyonun zayıflığına işarettir. En çok kullanılan hash fonksiyonları MD5, SHA256, SHA512, LM, NTLM gibi algoritmalardır. [34] Şekil 1.10'da C# diliyle yazılmış bir SHA256 hash hesaplama kod bloğu gösterilmektedir.

```
public static string Sha256HashHesapla(string rawData)
{
    // SHA256 Oluştur
    using (SHA256 sha256Hash = SHA256.Create())
    {
        // ComputeHash
        byte[] bytes = sha256Hash.ComputeHash(Encoding.UTF8.GetBytes(rawData));

        // Byte Array'i String'e dönüştür
        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < bytes.Length; i++)
        {
            builder.Append(bytes[i].ToString("x2"));
        }
        return builder.ToString();
    }
}
```

Şekil 1.10: SHA256 Hash Hesaplama Fonksiyonu

[34] Altınkaynak, Mustafa, *Uygulamalı Siber Güvenlik ve Hacking*, Abaküs Yayınları, İstanbul, 2017, s. 11

1.3.11 Yanlış Alarm (False Positive)

False Positive kavramı yazılım dilindeki "yanlış alarm" olarak tanımlanmaktadır. Uygulamalar üzerinde yapılan güvenlik testlerinin çıktısında hata veya zafiyet olarak görülen ama aslında bir hata veya zafiyet olmayan durumdur.[35]

False Positive sonuçlar güvenlik testleri için olumsuz durumlardan birisidir çünkü gerçek hataların da gözden kaçmasına sebep olabilir. False Positive sorunu Netsparker tarafından şu şekilde ifade edilmiştir[36]:

> Örneğin, bir web uygulaması güvenliği tarayıcısı 200 adet Cross Site Scripting açığı bulsa ve ilk 20 çeşidi false positive olsa, sızma testini gerçekleştiren uzman diğerlerinin de false positive olduğunu varsayar ve geri kalanları göz ardı eder. Bundan dolayı, aslında var olan web güvenlik açıklarının tespit edilememesi gibi bir sorun ortaya çıkar.

1.3.12 DNS

Açılımı Domain Name System yani Alan İsimlendirme Sistemi olan DNS, web üzerinde (Intranet, Internet gibi ağlar üzerinde) bulunan domain isimlerini IP adresine çevirmek için kullanılır. Domain isimleri (örneğin www.google.com) kolay akılda kalabilen isimler olduğu için tercih edilmektedir. Ancak bu web siteleri bir IP adresi üzerinden hizmet vermekte ve bunun istemciler tarafından öğrenilmesi için DNS'e ihtiyaç duyulmaktadır. Web tarayıcıları ile ziyaret edilen her web sitesi için ilgili DNS sunucusuna bir istek gönderilmektedir. Şekil 1.11'de örnek bir DNS A kaydı gösterilmektedir.

---

[35] Jianjun Huang, Bin Liang, Jiagui Zhong, Qianqian Wang and Jingjing Cai, "Vulnerabilities static detection for Web applications with false positive suppression," *2010 IEEE International Conference on Information Theory and Information Security*, Beijing, China, 2010, s.. 574-577, doi: 10.1109/ICITIS.2010.5689529.

[36] Web Uygulama Güvenliğinde False Positive Sorunu, https.//www.netsparker.com.tr/blog/web-guvenligi/web-uygulama-guvenliginde-false-positive-sorunu (25.02.2021)

```
id 47889
opcode QUERY
rcode NOERROR
flags QR RD RA
;QUESTION
www.google.com. IN A
;ANSWER
www.google.com. 299 IN A 64.233.162.106
www.google.com. 299 IN A 64.233.162.99
www.google.com. 299 IN A 64.233.162.147
www.google.com. 299 IN A 64.233.162.105
www.google.com. 299 IN A 64.233.162.103
www.google.com. 299 IN A 64.233.162.104
;AUTHORITY
;ADDITIONAL
```

Şekil 1.11: Google.com DNS A Kaydı

1.3.13 HTTP Çerezi (Cookie)

Çerez anlamına gelen Cookie, ziyaret edilen bir web uygulaması tarafından istemcinin cihazına kaydedilen küçük bir text dosyasıdır ve bazı tanımlama bilgileri içerir. Web Cookie, Internet Cookie gibi isimlerle de anılır. Cookie içerisinde ilgili web uygulaması ile ilgili oturum (session) bilgileri, ziyaret edilen ilgili uygulamaya ait sayfaların bilgileri, input ve arama bilgileri, alışveriş sepeti bilgileri ve buna benzer bilgiler tutulur. Cookie dosyaları sayesinde söz konusu web uygulamaya tekrar bağlanıldığında ilgili kullanıcının işlemleri kaldığı yerden devam edebilir.

Web Uygulama Güvenliği açısından da Cookie dosyalarının önemi bulunmaktadır. Cookie dosyaları saldırganlar tarafından ele geçirilebilmekte ve bu şekilde kullanıcı oturumları çalınabilmektedir. Bunun yanında Cookie dosyasının içeriğini değiştirme ve sunucuya farklı Cookie dosyasını yönlendirme gibi saldırı türleri mevcuttur. [37]

[37] Jussila, Jusha, *Http Cookıe Weaknesses, Attack Methods And Defense Mechanısms. A Systematıc Lıterature Revıew*, Unıversıty Of Jyväskylä, s. 23

İKİNCİ BÖLÜM

SALDIRI TÜRLERİ VE ÖNLEMLER

Web uygulamalarına yönelik saldırılar çok çeşitli olmakla birlikte en sık maruz kalınanları bu bölümde inceleyeceğiz. Öncelikle başarılı olan tüm saldırıların bir güvenlik açığından kaynaklandığını bilmemiz gerekli. Bu güvenlik açıkları bazen kodlayıcının hatasından kaynaklanırken, bazen de kodlama ortamındaki veya kullanılan hazır araçların zafiyetlerinden kaynaklanabilmektedir.

Öncelikle OWASP kuruluşunun web uygulamalarındaki güvenlik açıklarında bulunan risklere dair her yıl yayınladığı İlk 10 (TOP 10) listesi [38] üzerinde çalışmalar yapılacaktır. OWASP'ın 2020 yılında web uygulamaları için en kritik güvenlik açıkları listesi şu şekildedir:

1. Enjeksiyon (Injection)
2. Oturum ve Doğrulama Hataları (Broken Authentication)
3. Hassas Bilgi Sızıntısı (Sensitive Data Exposure)
4. Harici XML Varlıkları (XML External Entitites)
5. Hatalı Erişim Kontrolü (Broken Access Control)
6. Yanlış Güvenlik Yapılandırması (Security Misconfiguration)
7. Siteler Arası Betik Çalıştırma (Cross-Site Scripting)
8. Güvensiz Serileştirme (Insecure Deserilization)
9. Bilinen Zafiyetlere Sahip Bileşenleri Kullanma
10. Yetersiz Loglama ve İzleme

[38] *OWASP Top Ten*, https.//owasp.org/www-project-top-ten (25.02.2021)

Bu zafiyet türleri hakkında önce açıklamalar yapılacak, ardından bazılarını uygulama üzerinden örneklendirilecek ve son olarak her zafiyet türü için alınması gereken önlemlerden bahsedilecektir.

2.1 ENJEKSİYON (INJECTION)

OWASP'ın yayınladığı listeye göre en çok tehlike arz eden saldırı türüdür. Enjeksiyon saldırılarında saldırgan, uygulamadaki bir güvenlik açığından faydalanarak zararlı bir kod yazar ve bu şekilde uygulamaya yetkisiz şekilde müdahale edebilir.

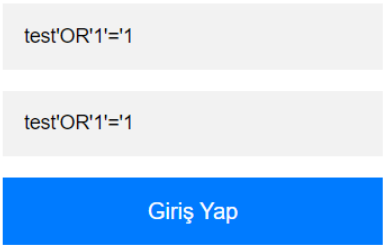
En yaygın olanı SQL Injection adı verilen saldırıdır. Ponemon Institude tarafından 2015 yılında Web Uygulama sahibi firmalarla yaptığı anket çalışmasına[39] göre SQL Injection saldırıları tüm saldırılarında %56'ünü oluşturmuştur. Bu saldırıda genellikle web sitesinde kullanıcıdan text inputu alınan alınan kısımlara SQL sorgu kodları yazılarak veri tabanına müdahale edilir.

Örneğin bir kullanıcı giriş ekranında Kullanıcı Adı "rtuk" ve Parola "bilkent" olarak girildiğinde; SQL Injection'a karşı zafiyeti bulunan bir web uygulaması, girilen bilgileri doğrulamak için veri tabanına bir sorgu gönderecektir. Bu örnekte oluşacak SQL sorgu kodu aşağıdakine benzer olacaktır:

```
select * from Kullanicilar where name='rtuk' and password='bilkent'
```

[39] SQL injection has surfaced as the no. 1 attack in 2015, https.//www.helpnetsecurity.com/2015/12/11/sql-injection-has-surfaced-as-the-no-1-attack-in-2015 (25.02.2021)

Bu aşamada SQL Injection saldırısının nasıl gerçekleştiği incelenecektir:



Şekil 2.1: Kullanıcı Giriş Ekranına Yazılan SQL Enjeksiyon Kodu

Şekil 2.1'de Input alanına bir SQL Injection saldırı kodu yazılmıştır. Kullanıcı Adı ve Parola yukarıdaki gibi girildiğinde SQL sorgusu aşağıdakine dönüşecektir:

select * from Kullanicilar where name= 'test'OR'1'='1' and password= 'test'OR'1'='1'

SQL dilinde [where name='test'OR'1'='1'] ifadesi [name değeri "test"tir veya bir eşittir birdir] anlamına gelir ve [bir eşittir bir] ifadesi her zaman True (doğru) sonuç vereceği için ve [password= 'test'OR'1'='1'] ifadesinden de yine True sonuç geleceği için Uygulama tarafından kullanıcı adı ve şifre doğru girilmiş farz edilecektir. Böylece yetkisiz bir kişi SQL Injection ile web uygulamasına giriş yapabilecektir.

2.1.1 Önlemler

SQL Injection saldırısını gerçekleştirmek için uygulamanın yapısını bilmek çok önemlidir. Bu yapıyı çözmek için genellikle deneme yanılma yöntemi kullanılır ve yüzlerce farklı saldırı kodu yazılır. Başarılı olan her deneme, veri tabanına zarar vermeyi başaramasa da uygulamanın ve veri tabanının iç yapısı hakkında saldırgana fikir verir, böylece istediği sonuca bir adım daha yaklaşmış olur.

Uygulamanın verdiği hata mesajları da saldırgan için yol göstericidir. Saldırgan hata oluşması için uğraşacaktır ve özellikle hata mesajlarında veri tabanıyla ilgili bir bilgi bulmak isteyecektir. Dolayısıyla uygulamada bir hata oluştuğunda ayrıntılı hata mesajı vermesi yerine elle oluşturulmuş bir hata sayfasına veya Anasayfa gibi farklı bir sayfaya yönlendirme yapılması Web Uygulama Güvenliği açısından önem arz etmektedir.

SQL Injection'dan korunmak için kullanılabilecek yöntemlerden birisi inputlarda, SQL'e özel karakterlerin (virgül, parantez, tırnak gibi) ve özel ifadelerin (drop, delete, update gibi) kullanımının kısıtlanmasıdır. Bu karakterlerin bulunduğu bir input, web uygulamanın yazılım kodu içerisinde henüz SQL sorgusuna çevrilmeden engellenebilir. Ancak bu durum web uygulamayı kullanışlı olmaktan uzaklaştıracak, yetkili kullanıcıların da inputlara girebileceği ifadeleri sınırlandıracaktır. Dolayısıyla etkili bir çözüm olduğu söylenemez.

Daha etkili korunma yöntemi ise input alanlarına girilen text ifadelerinin bir parametre olarak kabul edilmesi ve tüm textin bir String (metin) olarak SQL sorgusuna yazılmasıdır[40]. Örneğin C# dilinde yazılmış bir ASP.NET uygulamasında, veri tabanına ADO.NET teknolojisi ile bağlanırken kodun içerisinde SQL Injection zaafiyeti olan ve olmayan kodları inceleyelim:

```
String sorgu = "select * from Kullanicilar where name='" +input1.Text +"' and password='" + input2.Text +"' ";
SqlCommand sqlsorgu = new SqlCommand(sorgu,connection);
```

[40] Spett, Kevin, *SQL Injection: Are Your Web Applications Vulnerable*, SPI DYNAMICS, 2005, s.28

Yukarıdaki C# kodunda input1 ve input2 isimli input alanlarından gelen metinler direkt olarak SQL Sorgu Stringi'ne eklendiği için bu kod SQL Injection zaafiyeti bulunan bir koddur.

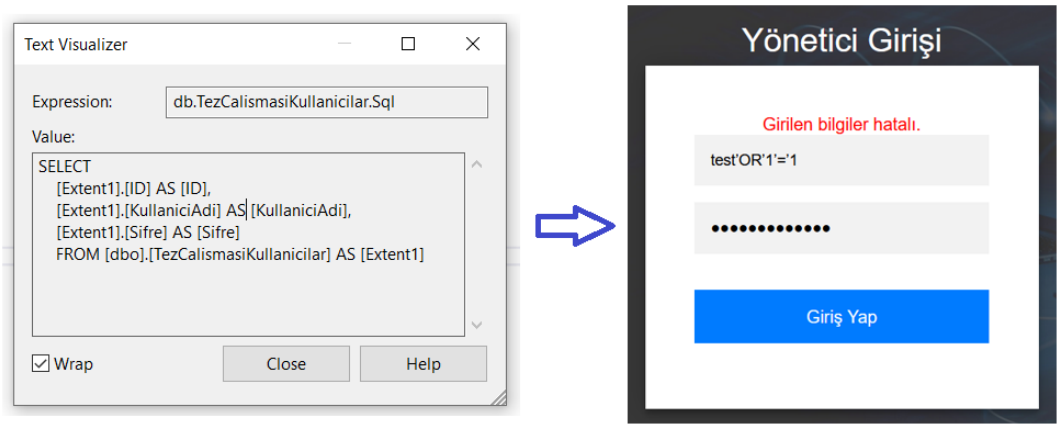
```
String sorgu = “select * from Kullanicilar where name=@KULLANICIADI and password=@SIFRE”;
SqlCommand sqlsorgu = new SqlCommand(sorgu,connection);
sqlsorgu.Parameters.AddWithValue(“@KULLANICIADI”,input1.Text);
sqlsorgu.Parameters.AddWithValue(“@SIFRE”,input2.Text);
```

Yukarıdaki C# kodunda ise input1 ve input2 alanlarından gelen değerler AddWithValue fonksiyonu kullanılarak parametre şeklinde SQL Sorgusuna eklendiği için bu kod SQL Injection zafiyetinden arınmıştır çünkü input olarak gönderilecek her türlü saldırı kodu bir metin ifade olarak algılanacak ve veri tabanına bir müdahalede bulunamayacaktır.

SQL Injection’dan korunmak için bir diğer yöntem ise veri tabanı bağlantılarında etkinliği ispatlanmış hazır yazılım iskeletleri(framework) kullanmaktır[41]. Örneğin Microsoft tarafından geliştirilen ve .Net Framework veya .Net Core ile beraber kullanılan Entity Framework(EF) veya JAVA teknolojilerinde kullanılan Hibernate gibi yazılım iskeletleri SQL Injection’a karşı etkili korunma sunmaktadır. Bu teknolojiler ile yazılan uygulamalarda kodlayıcının ek bir çaba harcamadan SQL Injection’dan korunması mümkündür ancak bu teknolojiler, veri tabanı isteklerinin ve yanıtlarının süresini uzattığı ve böylece uygulamanın hızını düşürdüğü için her zaman tercih edilmemektedir.

[41] Security Considerations (Entity Framework), https.//docs.microsoft.com/en-us/previous-versions/dotnet/netframework-4.0/cc716760(v=vs.100) (02.03.2021)

Bu aşamada tez çalışması için bir web uygulaması oluşturulmuş ve Entity Framework'ün SQL Injection saldırılarını önleyip önlemediği test edilmiştir.



Şekil 2.2: EF ile Çalışan Uygulamada SQL Enjeksiyonu Denemesi

Veri tabanı bağlantısı Entity Framework ile sağlanan bir ASP.NET MVC uygulamasında, oluşan SQL sorgusunda input alanlarından gelen değerlerin bir bütün halinde parametreye dönüştürmesiyle SQL Enjeksiyonunun engellendiği ve "test'OR'1'='1" ifadesi ile kullanıcı girişinin sağlanamadığı görülmüştür.

2.2 OTURUM VE DOĞRULAMA HATALARI (BROKEN AUTHENTICATION)

OWASP'ın listesinde ikinci sırada yer alan Broken Authentication zaafiyetleri Web Uygulamalarında kullanıcı oturumlarının yönetimindeki zayıflıklarından kaynaklanmaktadır. Bu zayıflıklar web uygulamasındaki kullanıcı giriş yöntemlerinin çözülmesi ve baypas edilmesi için saldırganlara fırsatlar sunar.

Örneğin saldırgan kimlik bilgisi doldurma alanlarına daha önceden hazırlanmış bir listedeki kullanıcı adı ve şifre bilgilerini otomatize bir yöntemle girebilir ve saniyeler içerisinde binlerce deneme yapabilir.

Bunun yanında oturumla ilgili bazı bilgiler, mesela Session ID bilgisi eğer URL içinde bulunuyorsa saldırgan bu bilgileri kopyalayıp başka bir kullanıcının aynı URL'ye tıklamasını sağlayarak onun oturumunu çalabilir. Bununla ilgili Mustafa Altınkaynak'ın Uygulamalı Siber Güvenlik ve Hacking (2017:35) isimli eserinde şu örnek bulunmaktadır:

> Saldırgan web sitesine ziyaret gerçekleştirir. Adres çubuğunda www.site.com/index.php/PHPSESSID=1556587294 şeklinde uzunca bir değer yer almaktadır. Saldırgan bu bağlantıyı kopyalar ve kopyaladığı bağlantıyı kurbana gönderir. Kurban bu bağlantıya tıklar ve siteye üye girişi gerçekleştirir. Artık 1556587294 değeri bir üyenin oturumunu temsil etmektedir. Bu sayede saldırgan oturumu bu değer üzerinden sabitlemiştir. Sayfa yenilendiği taktirde aynı oturuma sahip olacaktır.

2.2.1 Önlemler

Broken Authentication saldırılarından korunmak için Web Uygulama tasarlanırken ve kodlanırken dikkat edilmesi gereken bazı hususlar vardır. Yukarıda da belirtildiği gibi Session ID gibi hassas bilgilerin URL'de gösterilmemesi bu hususların en önemlilerindendir. Ayrıca Session ID bilgisinin tek seferlik olması ve kullanıcı oturumunu kapattıktan sonra saklanıp tekrar kullanılmamasının sağlanması önem arz etmektedir.

Bunun dışında kullanıcı şifrelerinin belirli bir karmaşıklık düzeyinin üzerinde olmaya zorlanması, yine kullanıcı şifrelerinin açık olarak değil Hash algoritmaları ile kodlanarak veri tabanında tutulması ve bu Hash algoritmalarından da güvenli olanların seçilmesi, oturum açma denemelerine çeşitli kıstaslar koyarak belli bir deneme sayısından sonrasına izin verilmemesi, birkaç denemeden sonra "Ben robot değilim

testi" olarak bilinen Captcha testlerinin yapılması gibi yöntemler de Broken Authentication saldırılarından korunmak için göz önünde bulundurulması gereken yöntemlerdir.[42]

2.3 HASSAS BİLGİ SIZINTISI (SENSİTİVE DATA EXPOSURE)

Hassas bilgilerin korunaksız şekilde bulundurulması durumudur. Web uygulamalarında diğer yazılım uygulamalarına göre daha fazla tehlike arz etmektedir. Özellikle Google gibi arama motorlarının bu hassas bilgileri indekslemesi ve ilgili aramalarda tüm internet kullanıcılarının görebilmesi mümkündür.

Hassas bilgi denildiğinde anlaşılan kişisel veriler, kredi kartı bilgileri gibi ticari veriler, sağlık verileri, şifre bilgileri gibi verilerdir[43]. Web uygulamasının bu tür verileri çok dikkatli ve güvenli bir şekilde saklaması gerekmektedir.

Data Breach olarak adlandırılan ve hassas bilgilerin çalınması olarak tanımlanabilecek durumda ise finansal kayıplar, kişisel verilerin ihlali sonucu hukuki problemler ve web uygulamasına duyulan itibarın kaybı gibi problemler yaşanabilir.

2.3.1 Önlemler

Hassas bilgilerin korunması için alınması gereken önlemlerin başında Web Uygulaması'nın SSL/TLS sertifikası ile çalıştırılması gelmektedir. SSL/TLS ile

[42] Sharma Rupal, Sheth Ravi, "Discover Broken Authentication and Session Management Vulnerabilities in ASP.NET Web Application", *Oriental Journal Of Computer Scıence & Technology* Sayı: 10/2, 2017, s. 361-362

[43] Sensitive Data Exposure, https.//www.contrastsecurity.com/knowledge-hub/glossary/sensitive-data-exposure (03.03.2021)

uygulama HTTPS üzerinden hizmet verdiğinde verilerin alt ağ katmanlarında korunması sağlanmaktadır.

Oturum ve Doğrulama Hataları (2.2) konusunda bahsedilen önlemlerden; şifrelerin Hash işleminden geçirilerek veri tabanında saklanması, şifre politikasının sıkı tutulması bu konuda da önem arz etmektedir.



Şekil 2.3: SHA256 Hash Algoritmasıyla Yapılan Şifreleme [44]

Hassas veri içeren dosyalar URL üzerinden herkesin ulaşımına açıksa ve dosya sadece sayılı kullanıcıya özgü ise, yetkisiz kişiler tarafından dosya isimlerinin tahmin edilememesi için ilk eklenme anında dosya isminin bir GUID veya benzeri metinle değiştirilmesi gerekmektedir.

Kullanıcının giriş yaptığı input alanlarının, kod tarafında güçlü doğrulama metotlarından geçirilerek okunması, sunucu tarafında güvenlik duvarı (firewall) kullanımı, uygulamanın belli aralıklarla sızma testlerine tabi tutulması da alınması gereken diğer önlemlerdendir.

2.3.2 SSL/TLS'in Önemi

HTTPS üzerinden veri transferi yapıldığında web uygulamasının bulunduğu web sunucusunda bir SSL/TLS sertifikasının yüklü olması ve bu sertifika ile bağlantının şifrelenmesi gerekmektedir. Günümüzde güvenlik açısında son derece önemli olan bu teknik modern web siteleri için vazgeçilmez hale gelmiştir.

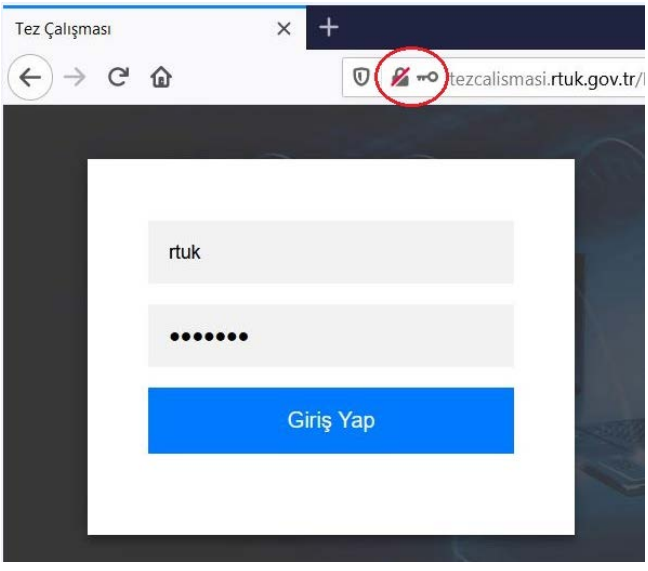
[44] SHA 256 Encrypt & Decrypt, https.//md5decrypt.net/en/Sha256 (02.03.2021)

Bu kısımda bir uygulamanın HTTP üzerinden çalıştırılması ve SSL/TLS sertifikası kullanılarak HTTPS üzerinden çalıştırılması arasındaki fark üzerine bir test çalışması yapılacaktır

Bu çalışma için kullanılan bileşenler aşağıdaki gibidir:

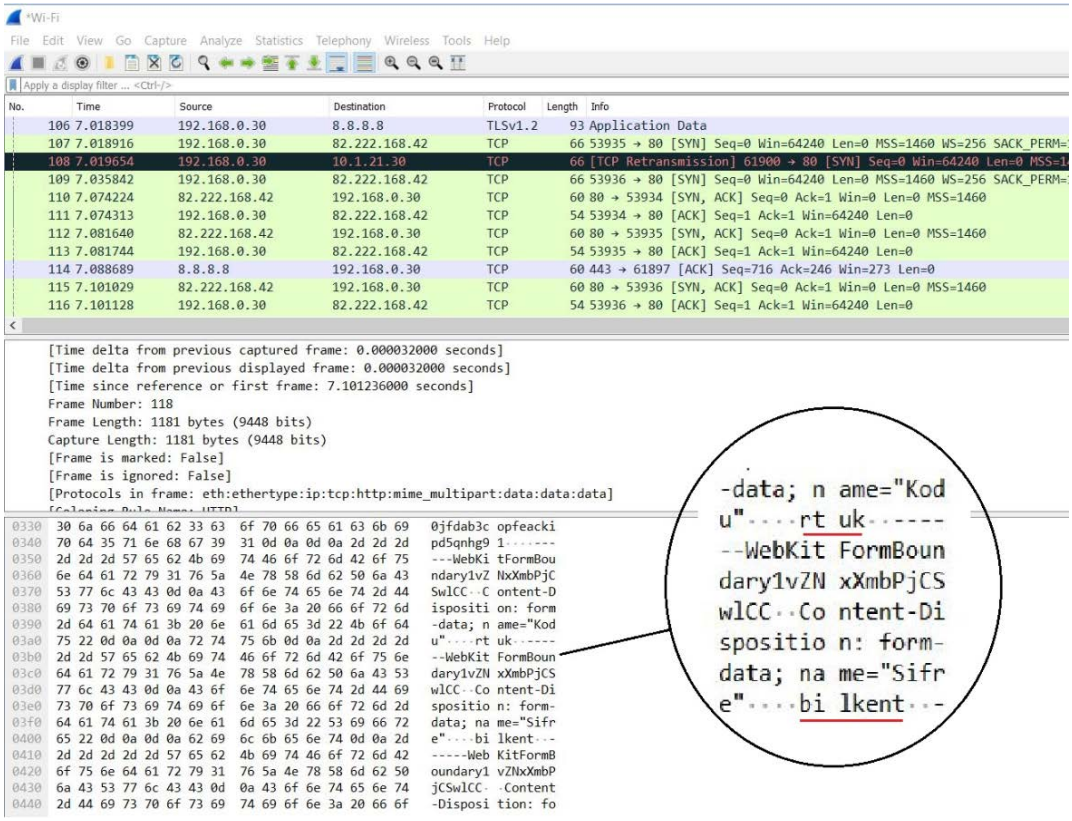
- Örnek amaçlı oluşturulmuş bir ASP.NET web uygulaması,
- Web uygulamasının çalışması için bir Windows Server sunucuda hizmet veren bir IIS Web Sunucusu,
- Herhangi bir web tarayıcısı
- Wireshark isimli Ağ İzleme Yazılımı

Çalışmanın ilk etabına web uygulamasındaki input alanlarına giriş yaparak başlanacaktır.



Şekil 2.4: Uygulamanın HTTP Protokolüyle Çalıştırılması

Şekil 2.4'te "güvenli değil" ifadesinden de anlaşılacağı üzere uygulama SSL/TLS sertifikası olmadan http protokolü ile çalıştırılmıştır. Kullanıcı giriş ekranına [Kullanıcı Adı: rtuk, Şifre: bilkent] şeklinde iki değer girilmiştir. Bir sonraki aşamada; bu bağlantı üzerinden iletilen paketlerin içeriklerinin, örneğin kullanıcı adı ve şifre bilgilerinin üçüncü bir taraf tarafından okunup okunmadığını denetlemek için ilgili paketler, ağ paketlerini ve network protokollerini izlemek için geliştirilmiş Wireshark yazılımı[45] ile incelenecektir. Bu inceleme HTTP protokolünün güvenlik seviyesi hakkında fikir veren bir gösterge olacaktır.

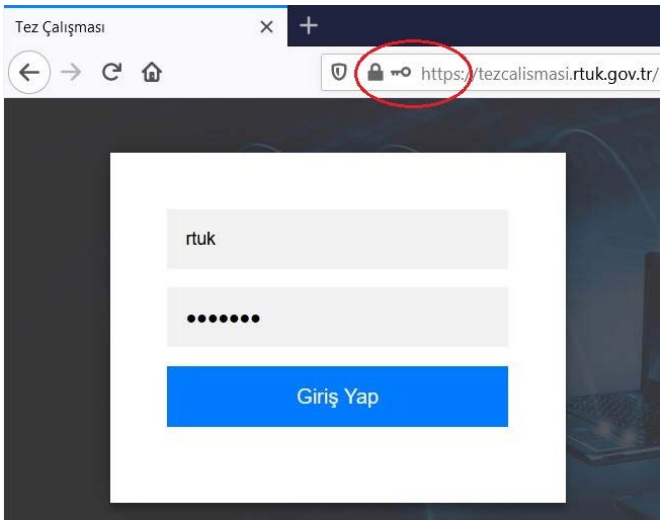


Şekil 2.5: Güvensiz Siteye Ait Paketin İçeriği

[45] What is Wireshark? What this essential troubleshooting tool does and how to use it, https.//www.csoonline.com/article/3305805/what-is-wireshark-what-this-essential-troubleshooting-tool-does-and-how-to-use-it.html (05.03.2021)

Şekil 2.5'te görüldüğü gibi, ilgili paket incelendiğinde giriş ekranına girilen "rtuk" ve "bilkent" ifadeleri ağdaki paket içerisinde açık şekilde görünmektedir. Bu da paketin içeriğinin şifrelenmeden gönderildiğini göstermektedir.

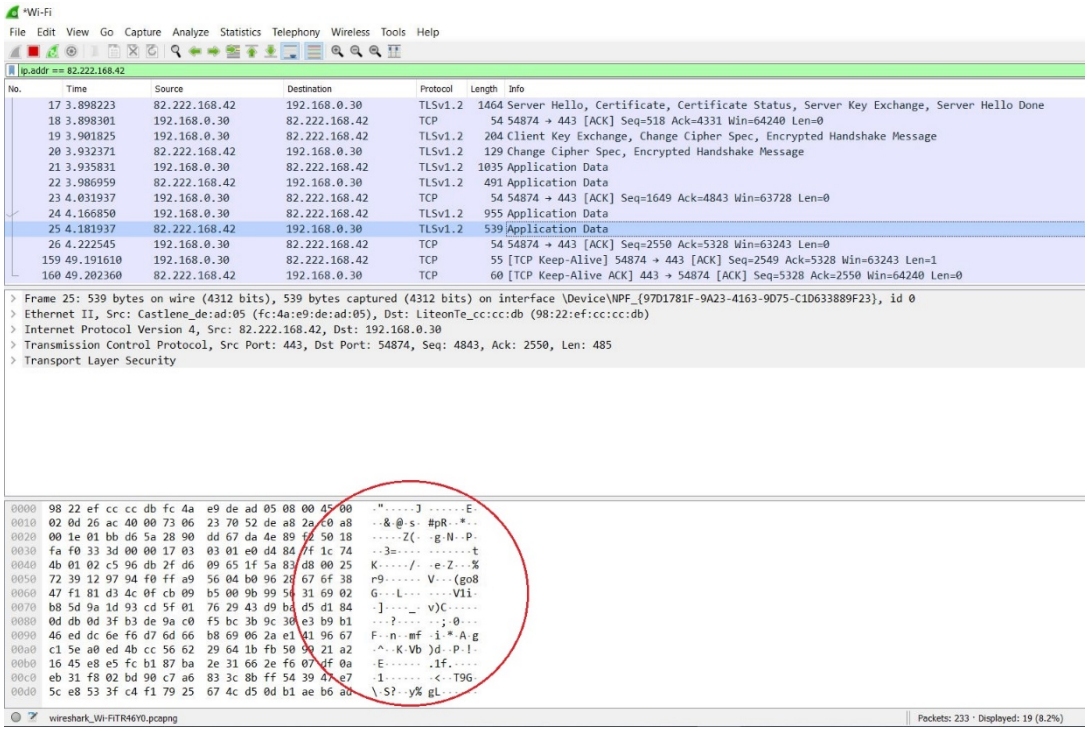
Bu aşamada SSL/TLS sertifikası ile HTTPS üzerinden yapılan bağlantı incelendiğinde aşağıdaki görünüm ortaya çıkacaktır.



Şekil 2.6: HTTPS ile Yapılan Bağlantının Görüntüsü

Şekil 2.6'da ekranın sol üstündeki kilit simgesinden de anlaşılacağı üzere HTTPS bağlantısı yapılmıştır ve yine aynı şekilde giriş ekranında kullanıcı adı ve şifreler "rtuk" ve "bilkent" şeklinde girilmiştir.

Bu sefer uygulamamız ile istemci olarak giriş yaptığımız bilgisayar SSL/TLS sertifikası üzerinden şifreli haberleşeceği için güvenli bir veri transferi olmasını bekleyebiliriz. Şimdi bu bağlantıyla üretilmiş paketleri yine Wireshark yazılımı ile inceleyeceğiz ve paketlerin içeriğindeki hassas verilerin üçüncü bir tarafa ifşa olup olmadığını göreceğiz.



Şekil 2.7: HTTPS Protokolü Üzerinden Gönderilen Paketin İçeriği

Şekil 2.7'de HTTPS protokolü ile güvenli hâle getirilmiş web sitesine ait paket yine Wireshark yazılımı ile incelenmiştir. Görüldüğü gibi paketin içeriği SSL/TLS protokolleri ile şifrelenmiştir ve ekranda çözülmesi mümkün olmayan karmaşık karakterler gözükmektedir. Bu da veri transferinin güvenli yoldan yapıldığına işaret etmektedir.

2.4 HARİCİ XML VARLIKLARI (XML EXTERNAL ENTITIES)

Kısa adı XXE'dir. Saldırganların XML (eXtensive Markup Language) verilerinin işlenmesi sürecinde müdahale etmesine olanak verilmesi sonucu ortaya çıkan bir zafiyettir. Saldırının başarılı olması; uygulamanın servis dışı kalmasına veya uygulamanın kritik bilgilerinin, uygulama sunucusunun dosya sisteminin saldırgan tarafından görülmesine yol açabilir hatta uygulamanın kod satırlarının değiştirilmesini bile mümkün kılabilir. Bu durum da çok kritik bir güvenlik açığına işaret etmektedir.

2013 yılında dünyaca ünlü sosyal medya uygulaması Facebook'ta bulunun bir zafiyetin ortaya çıkması harici XML varlıkları güvenlik açığını daha bilinir kılmıştır. Mısırlı Mohamed Ramadan isimli bir araştırmacı bu açığı ortaya çıkarmış ve Facebook tarafından 6.000 dolar ile ödüllendirilmiştir. Ramadan'ın aktardığında göre [46] bu açık sayesinde Facebook'un kariyer bölümünü kullanan üyelerin sayfalarına ".docx" formatında dosyalar eklenebilmekteydi.

Uygulama güvenliği üzerinde çalışmalar yapan OWASP topluluğunun bu konudaki bir çalışması şu şekildedir: [47]

Örnek 1: Saldırgan uygulamadaki hassas bir şifre verisini ele geçirmek ister.

```
<?xml version="1.0" encoding="ISO-8859-1"?> <!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]> <foo>&xxe;</foo>
```

Örnek 2: Saldırgan, Örnek 1'deki Entity satırını alttaki şekilde değiştirerek sunucunun özel ağını araştırır.

```
<!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>
```

Örnek 3: Saldırgan sunucuyu hizmet dışı bırakmak için sonu olmayan bir dosyayı XML'e dahil eder.

```
<!ENTITY xxe SYSTEM "file:///dev/random" >]>
```

[46] XXE Bug Patched in Facebook Careers Third-Party Service, https.//threatpost.com/xxe-bug-patched-in-facebook-careers-third-party-service/110151 (08.03.2021)
[47] XML External Entities(XXE) Processing, https.//owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing (09.03.2021)

2.4.1 Önlemler

Harici XML varlıkları güvenlik açıkları uygulamada XML dosyalarını işleyen araçların veya kod satırlarının yanlış veya eksik yapılandırılmasından kaynaklanır. Doğru yapılandırma için bazı kriterlere uymak gerekmektedir. Bunlardan en önemlisi uygulamamızda üçüncü parti olarak kullandığımız araçların XXE zafiyeti içermediğinden emin olarak bu araçları kullanmaktır.

2.5 HATALI ERİŞİM KONTROLÜ (BROKEN ACCESS CONTROL)

Bu güvenlik zafiyeti, uygulamanın kullanıcılar için yaptığı kimlik doğrulama işleminden sonra kullanıcının erişim yapabileceği ve yapamayacağı kısımların doğru konfigüre edilmemesinden kaynaklanır.[48] Bu durumda kullanıcılar yetkisiz işlemler yapabilir, başka bir kullanıcının alanına veya profiline müdahale edebilir. Saldırganlar da haliyle bu durumu kötü amaçla kullanıp uygulamaya ciddi zararlar verebilirler, açığın boyutuna göre verileri okuma, ekleme, güncelleme ve silme işlemlerinden bazılarını gerçekleştirebilirler.

Örneğin kullanıcı detaylarını görmek için kullanıcı adı bilgisinin GET metodu ile web tarayıcının link butonuna girildiği bir uygulama düşünelim:

```
www.ornek.com/kullanicidetay?kullaniciadi=ahmt
```

[48] V. Dehalwar, A. Kalam, M. L. Kolhe and A. Zayegh, "Review of web-based information security threats in smart grid," *2017 7th International Conference on Power Systems (ICPS)*, Pune, India, 2017, pp. 849-853, doi: 10.1109/ICPES.2017.8387407. s. 852

Eğer kaynak kod tarafında herhangi bir doğrulama yapılmadıysa herhangi bir kullanıcı adını giren saldırgan o kullanıcının detaylarına ulaşabilecek ve devamında farklı işlemleri de gerçekleştirebilecektir.

2.5.1 Önlemler

Erişim Kontrolünün doğru çalışmasının tek yolu sunucu tarafında bulunan uygulama yazılımının doğru kodlanmasıdır ve kuralların doğru belirlenmesidir. Kurallar belirlenirken, bir kullanıcıya ve kullanıcı grubuna verilen yetkiler için bir "izin verilenler" dairesi oluşturulmalı ve daire dışında kalan yetkiler kısıtlanmalıdır. Tersi yapıldığında bir "yasak dairesi" güvenlik açıklarına çok daha müsait olacaktır.

Eğer kullanıcı yönetimi için bazı üçüncü parti hazır araçlar kullanılıyorsa, bu araçların ihtiyaçları karşıladığından ve güvenilir olduğundan emin olunmalıdır. Uygulama sunucuda canlı yayına alınmadan önce kullanıcı yönetiminin ve erişim kontrollerinin testleri detaylı şekilde yapılmalıdır.

2.6 YANLIŞ GÜVENLİK YAPILANDIRMASI (SECURITY MISCONFIGURATION)

Yanlış Güvenlik Yapılandırması en sık görülen zafiyetlerden bir tanesidir. Uygulamanın her aşamasında karşılaşılabilecek bir problemdir. Genellikle uygulamaların yapılandırma ayalarının varsayılan (default) şekilde bırakılmasından veya elle yapılan ayarlardaki hatalı yapılandırmadan kaynaklanır. Yazılımın kullandığı çeşitli framework, yazılım kütüphaneleri ve araçların da güncelliğini yitirmesi zafiyetlere sebep olabilmektedir.[49]

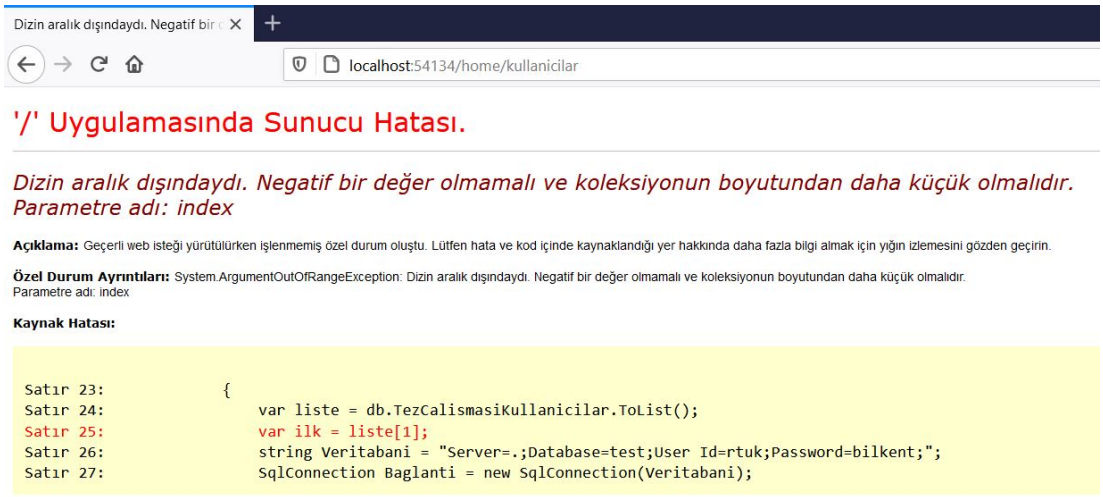
[49] Hoffman, Andrew, *Web Application Security Exploitation and Countermeasures for Modern Web Applications*, O'REILLY, ABD, 2020, s. 217

Bu zafiyet türü saldırganların yetkisiz erişim elde etmelerine yol açabilir ve uygulamaya ait veriler çalınabilir, değiştirilebilir veya silinebilir. Dolayısıyla kritik seviyede bir zafiyet türüdür. Saldırgan genellikle saldırı girişiminde boş sayfaları, ön tanımlı kullanıcı hesap adlarını, varsayılan dosya, dizin ve araç isimlerini kullanır.

2.6.1 Önlemler

Security Misconfiguration zafiyetlerini önlemenin en önemli yollarından birisi uygulamada gereksiz özelliklerden kaçınmaktır. Sade yapı saldırganın elindeki imkanları kısıtlayacaktır. Güvenlik Yapılandırmasındaki hataları tespit etmek için bazı otomatik tespit araçları vardır ve bu araçlar hemen her uygulama için faydalı olmaktadır. Bunun dışında zafiyetlere karşı uygulamanın elle test edilmesi de kaçınılmazdır.

Bu aşamada Güvenlik Yapılandırma açıkları için literatürde önerilen yöntemlerin etkinliğinin test edilmesi için bir web uygulaması yazılmıştır.

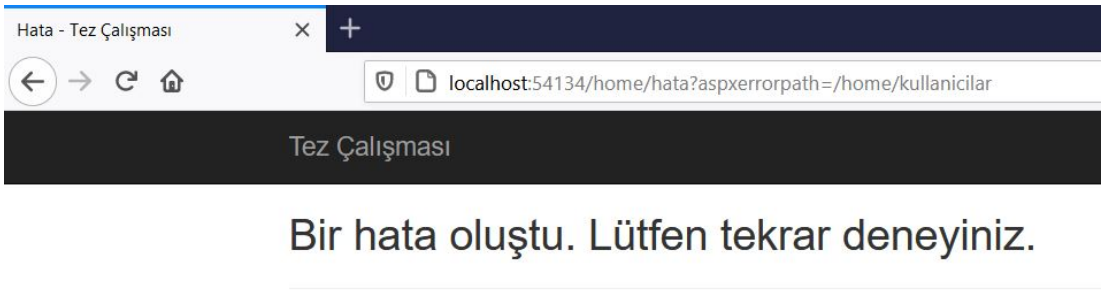


Şekil 2.8: ASP.NET Varsayılan Hata Sayfasında Kritik Bilgilerin İfşa Olması

ASP.NET MVC teknolojisi kullanılarak yazılan uygulamada, yapılandırma dosyası olan “web.config” içerisinde “customErrors mode="On" etiketi eklenmiş ve

istemci tarafından yapılan hatalı bir işlem sonucu oluşan hata sayfasında, ilgili kodlarda bulunan hassas bilgilerin ifşa olduğu görülmüştür. Şekil 2.8'de uygulamanın varsayılan hata sayfasında kaynak hatası bölümünde 26. Satırda bulunan ve gizli kalması gereken veri tabanı şifresi görünmektedir.

Ardından bu güvenlik açığını gidermek için önerilen yapılandırma ayarı olan "customErrors mode="Off" etiketi eklenerek, uygulama özel geliştirilmiş bir hata sayfasına yönlendirilmiştir.



Şekil 2.9: Özel Oluşturulmuş Hata Sayfası

2.7 SİTELER ARASI BETİK ÇALIŞTIRMA (CROSS-SITE SCRIPTING)

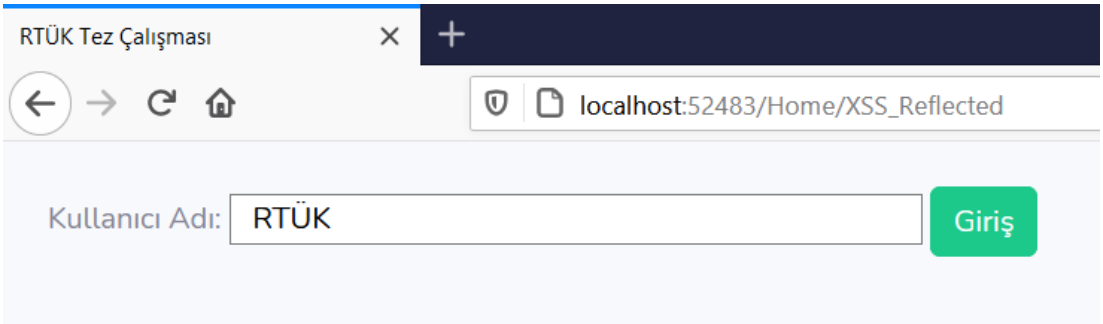
Kısaca XSS olarak anılan Cross-Site Scripting, web uygulamasında bulunmayan ve saldırgan tarafından haricen yazılan zararlı Script kodlarıyla yapılan bir saldırı türüdür. Web sitesini zararlı şekilde çalışmaya zorlayan bir atak şeklinde tanımlanabilir[50]. Scripting dili olarak en çok Javascript kullanılır. Uygulamada Script kodları bir filtreden geçirilmeden çalıştırıldığında ortaya çıkan bir zafiyettir. SQL Injection ile benzerlik gösterir ve genellikle uygulamadaki text input alanlarını kullanır. XSS saldırılarını iki tür olarak incelenmektedir.:

[50] Grossman, Jeremiah, *XSS Attacks Cross Site Scripting Exploits and Defence*, SYNGRESS, 2007, s. 67

2.7.1 Reflected XSS

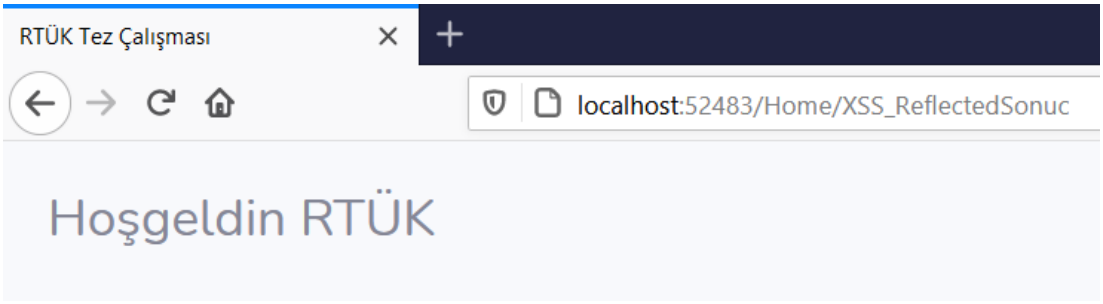
Uygulamada, kullanıcının text girmesi için hazırlanmış input alanlarında zararlı Javascript kodları, giriş sonrası yine aynı web tarayıcısı (browser) ekranında çalıştırılır.

Bu aşamada Reflected XSS konusunda bir ASP.NET MVC uygulaması üzerinden örneklendirme yapılacaktır:



Şekil 2.10: XSS Uygulaması İsim Giriş Ekranı

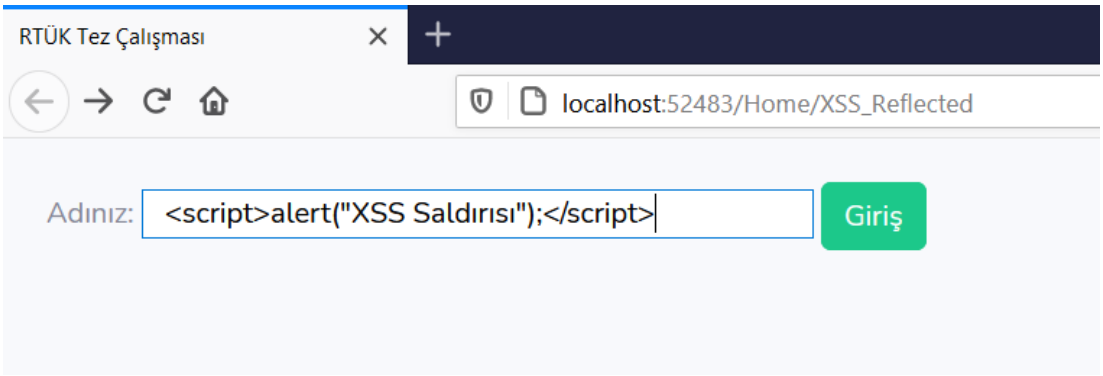
Örnek uygulamada bir input alanına kullanıcı tarafından isim girilip "Giriş" butonuna basıldığında, input olarak girilen değer uygulama tarafından kaydedilmekte ve ilgili inputun içeriğindeki değer bir başka değişkene basılarak diğer sayfaya yönlendirilmektedir. Yeni açılan sayfada da ilgili değişkene eklenen değer herhangi bir kontrol geçirilmeden ekrana basılmaktadır.



Şekil 2.11: XSS Uygulaması Yönlendirilen Hoşgeldiniz Sayfası

Yönlendirilen sayfada "Hoşgeldiniz" ifadesinden sonra ilk sayfada girilen isim basılmaktadır ve herhangi bir girdi kontrolünün olmaması XSS saldırısına müsait bir yapı ortaya çıkarmaktadır.

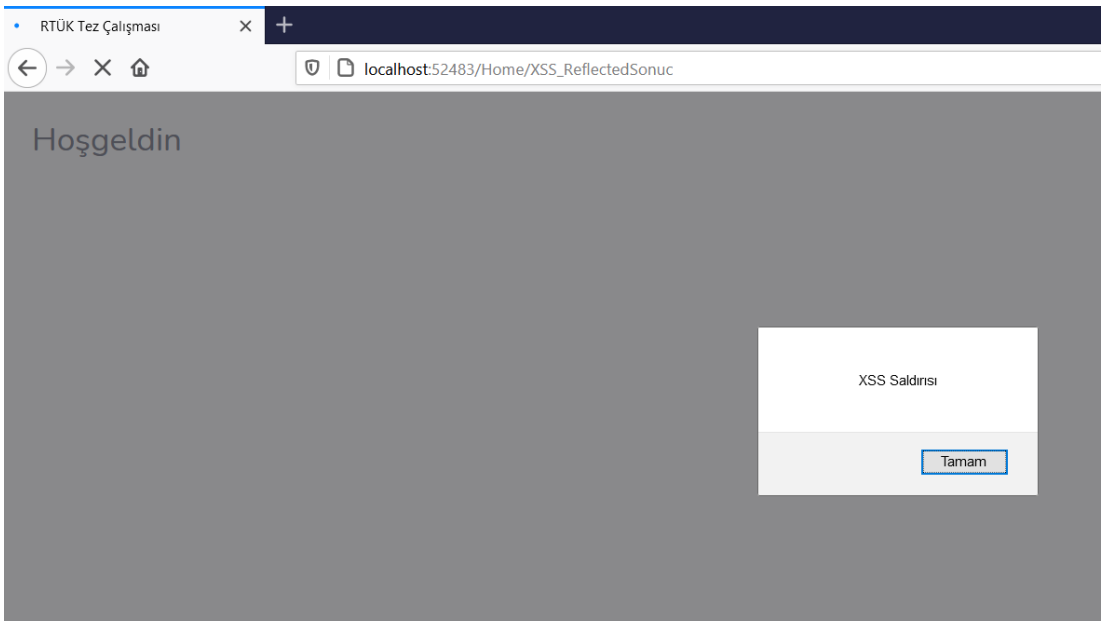
Bu aşamada, güvenlik açığı bulunan bir uygulamada Cross Site Scripting saldırısının nasıl yapıldığını incelenecektir. Aynı isim giriş sayfasına giderek XSS içerikli saldırı kodu girildiğinde XSS saldırısının ne olduğu daha iyi anlaşılacaktır.



Şekil 2.12: Input Alanına Girilmiş XSS Saldırısı Script Kodu

Şekil 2.12'de görüldüğü gibi input alanına Javascript'in "Alert" fonksiyonunu içeren bir script kodu yazılmış ve giriş butonuna basılmıştır. Bu durumda input alanına girilen değer uygulama tarafından yine kaydedilip bir sonraki sayfada gönderilmek üzere bir değişkene atanacak ve herhangi bir kontrol işleminden geçirilmeden ilgili sayfaya yönlendirilecektir. XSS içerikli saldırı kodu da yeni açılan sayfada çalışacak

ve sayfanın amacının dışında bir fonksiyonu çalıştıracaktır. Bu durum da bir güvenlik açığının kötü amaçla kullanımıma örnek teşkil etmektedir.



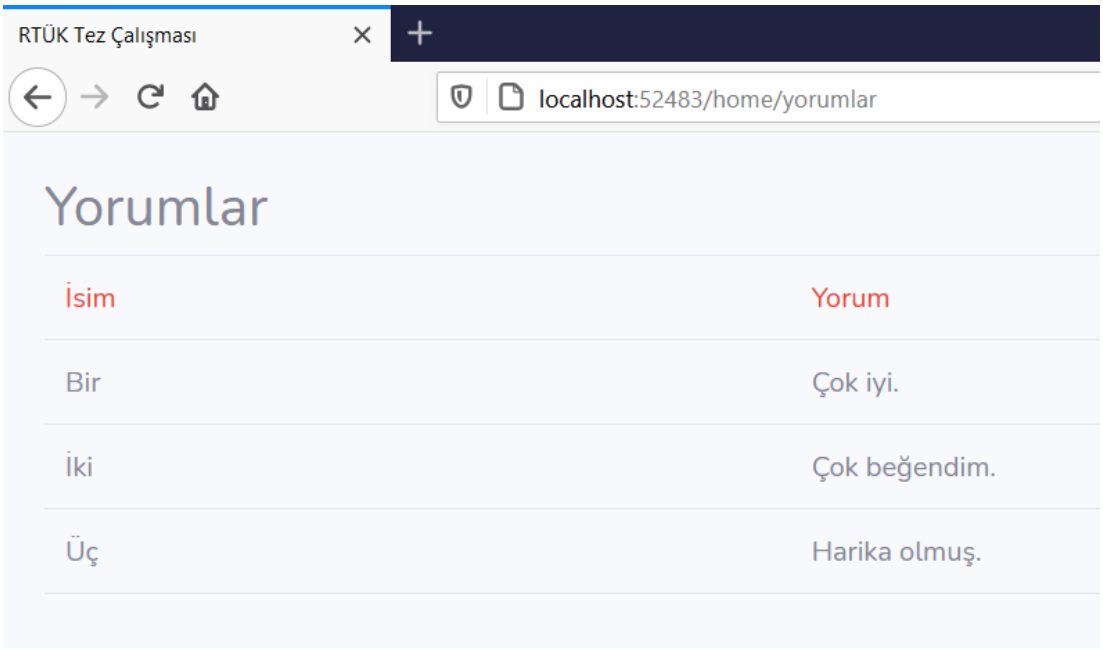
Şekil 2.13: Yönlendirilen Sayfada Alert Fonksiyonunun Çalışması

Yönlendirmenin ardından isim alanına girilen metnin "Hoşgeldin" ifadesinden sonra yazılması gerekirken, ilgili metin bir script kodu olduğu için Javascript'in "Alarm" fonksiyonunu tetiklemiş ve bir alarm ekranını açılmıştır ve uygulamanın amacı dışında bir eylemi gerçekleştirilmiştir. XSS saldırısı başarılı olmuş ve güvenlik zafiyetinin kötü amaçlı kullanımı simüle edilmiştir.

2.7.2 Stored XSS

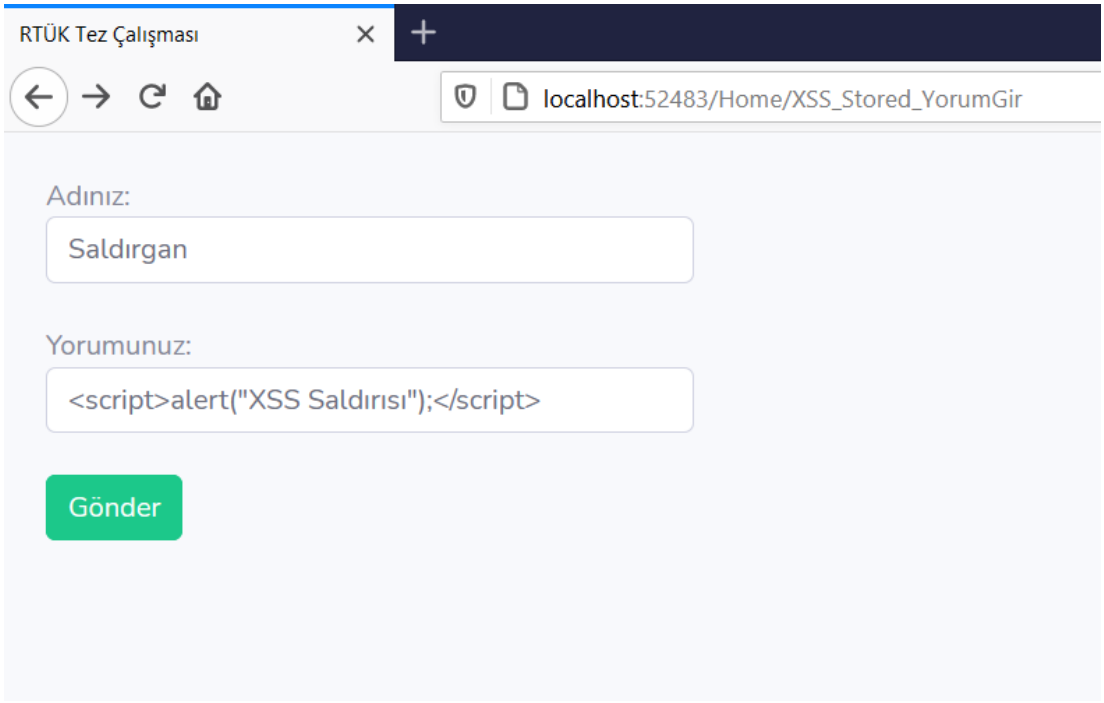
En tehlikeli XSS türüdür. Saldırgan tarafından girilen XSS içerikli script kodunun uygulamada veya veri tabanında saklandığı durumlarda gerçekleşir. Bu durumda XSS zafiyetli kalıcı hale gelir ve ilgili sayfayı ziyaret eden her kullanıcı saldırıdan/zafiyetten etkilenir. Bu sebeplerden dolayı saldırganların en çok tercih ettiği XSS türüdür.

Bu aşamada Stored XSS'in ne demek olduğu bir örnekle açıklanacaktır. XSS saldırısında örnek teşkil etmesi için yazılmış bir web uygulamasında, ilk ekrandan İsim ve Yorum bilgileri girilmekte ve ardından yorumların listelendiği bir sayfaya yönlendirme yapılarak yorumların listelenmesi yapılmaktadır.



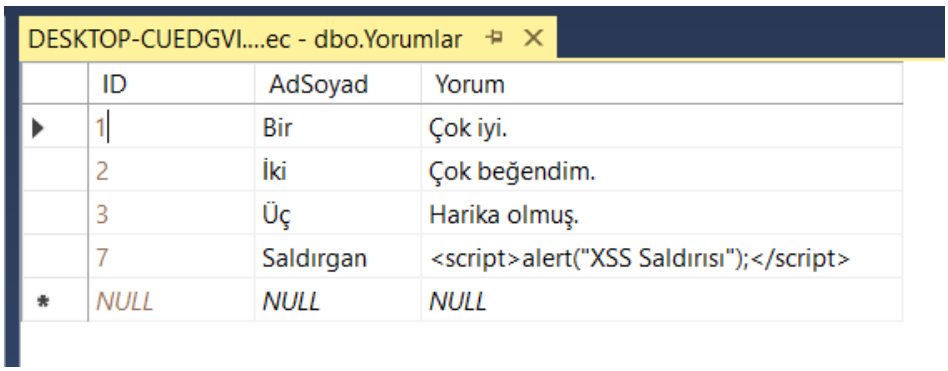
Şekil 2.14: XSS Uygulamasında Yorumların Listelendiği Sayfa

Şekil 2.14'te önceki sayfada girilen bilgilere göre eklenen yorumların listelendiği sayfa görülmektedir. Input alanına girilen değer web tarayıcı tarafından POST metoduyla web uygulamaya gönderilmekte ve arka yüz kodları tarafından input değeri veri tabanına yazılmaktadır. Yorumların listelendiği sayfada ise veri tabanından çekilen isim ve yorum değerleri HTML olarak basılmaktadır. İşte bu noktada devreye XSS tehlikesi girmektedir. Eğer input değeri olarak sakıncalı bir içerik girilirse bu durum o değer veri tabanına yazılacak ve listeleme sayfasında ilgili değer veri tabanından çekilip çalıştırıldığı anda XSS saldırısı gerçekleşecektir.



Şekil 2.15: XSS Uygulamasında Yorum Giriş Alanına Yazılmış Saldırı Kodu

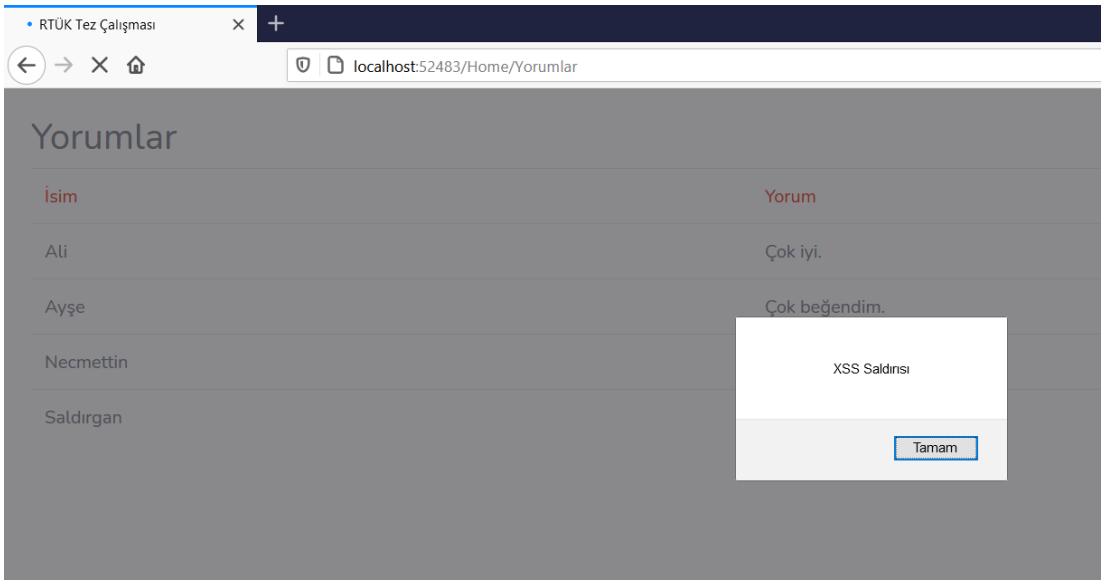
Yorum bölümüne şekil 2.15'te görülen saldırı kodu yazıldığında bu kod web uygulamasının arka planda çalışan kodları tarafından yorum olarak veri tabanına kaydedilecektir çünkü uygulama kodlarında henüz herhangi bir XSS karşıtı önlemi alınmamıştır ve girilen değerin zararlı bir içerik olup olduğu sorgulanmamaktadır.



	ID	AdSoyad	Yorum
▶	1	Bir	Çok iyi.
	2	İki	Çok beğendim.
	3	Üç	Harika olmuş.
	7	Saldırgan	<script>alert("XSS Saldırısı");</script>
*	NULL	NULL	NULL

Şekil 2.16: XSS Uygulamasında Yorumların Saklandığı Veri tabanı Tablosu

Şekil 2.16'da görüldüğü gibi veri tabanına XSS saldırı kodu kaydedilmiş ve artık kalıcı hale gelmiştir. XSS kodunun bulunduğu yorum veri tabanından çekildiği aşamada herhangi bir XSS karşıtı kontrol bulunmadığı için web uygulama tarafından ilgili yorum aynen bastırılacaktır, bu da bir güvenlik zafiyeti olarak karşımıza çıkacaktır.



Şekil 2.17: Yorumların Listelendiği Sayfanın XSS Saldırısı Sonrası Görüntüsü

Şekil 2.17'de görüldüğü gibi saldırı sonrasında, yorumların listelendiği sayfada yine bir Javascript "Alert" fonksiyonu çalışmış ve sayfayı ziyaret eden kişi istenmeyen bir görüntüyü görmüştür. Yorumlar veri tabanına kalıcı olarak kaydedildiği için bu saldırı kodu da her sayfa yenilenmesinde çalışacak ve sayfayı her ziyaret eden bu saldırıdan etkilenecektir.

Alert penceresinin içerisine yasadışı bir ifadenin yazılması veya JQuery gibi bazı Javascript kütüphanelerinin Alert fonksiyonuna kazandırdığı bazı özellikler sayesinde Alert penceresi içerisinde yasadışı bir resmin eklenmesi de mümkündür. Bu da tehlikenin boyutunu artırmaktadır.

2.7.3 Önlemler

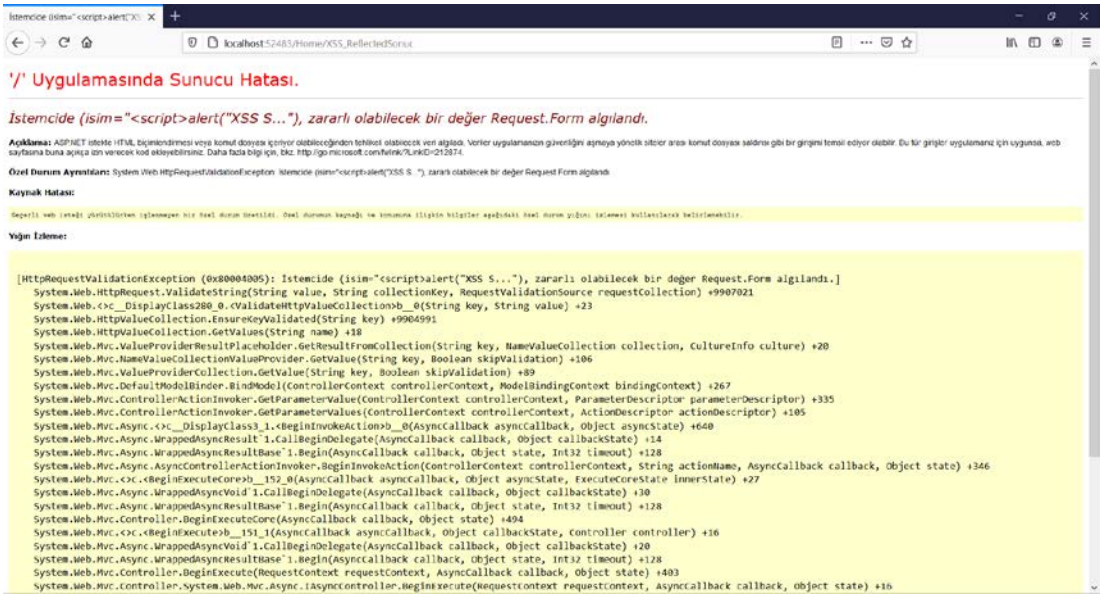
XSS saldırılarını önlemek için yapılması gereken şey, Input alanlarına yazılan script kodlarının içeriğinin kontrol edilerek çalıştırılmasıdır. Bunun için yine bir ASP.NET MVC uygulamasında nelere dikkat edilmesi gerektiğini inceleyelim:

```
[HttpPost]
[ValidateInput(true)]
public ActionResult XSS_ReflectedSonuc(string isim)
{
    ViewBag.isim = isim;
    return View();
}
```

Şekil 2.18: XSS'i Engelleyen Öznitelik Eklenen Metot

Şekil 2.18'de görüldüğü gibi ilgili metot üzerine "ValidateInput(true)"[51] isimli attribute (öznitelik) eklenmiş ve input alanların script kodları yazıldığında bunun çalıştırılmasının önüne geçilmiştir. ValidateInput özniteliği "true" değer aldığında XSS saldırısını önleyecek bir yapıya sahiptir. Bu öznitelik bir fonksiyonu tetiklemekte ve ilgili fonksiyon input alanlarından gelen değerlerin HTML ve Javascript'e ait herhangi bir tag (etiket) içerip içermediğini kontrol etmekte ve içerdiği durumlarda uygulamayı varsayılan bir hata sayfasına yönlendirmektedir. Bu durumda yukarıda bulunan söz konusu uygulama az önce girilen Javascript kodlarına izin vermeyecek ve XSS saldırısı engellenecektir.

[51] ValidateInputAttribute Class, https.//docs.microsoft.com/en-us/dotnet/api/system.web.mvc.validateinputattribute (05.03.2021)



Şekil 2.19: XSS Saldırı Kodu Sonucu Yönlendirilen Hata Sayfası

Şekil 2.19'da görüldüğü gibi "ValidateInput" özniteliğinin eklenmesi ve "true" değeri verilmesi sonrası XSS Saldırısı engellenmiş ve zararlı bir script kodunun yazıldığı ASP.NET'in ilgili teknolojisi tarafından tespit edilerek uygulama hata sayfasına yönlendirilmiştir.

Bu kadar küçük bir değişiklikle XSS saldırısının engellenmesi mümkündür. Ancak bazı durumlarda, örneğin text editörle input alanına düzenli ve paragrafla içeren bir metnin girilmesi halinde "ValidateInput" özniteliğini "false" değer ile eklemek zorunlu olabilmektedir. Aksi halde text-editörden gelen değer kabul edilmeyecektir. Bu durumda ilgili input sonraki sayfada bastırılırken kullanılan "Html.Raw" fonksiyonu da XSS saldırısında müsait bir yapı sunduğundan, saldırı karşıtı önlem olarak elle (manuel) bazı kontroller eklemek gerekmektedir.

```
<h2>Yorumlar</h2>
<table class="table">
    <thead>
        <td class="text-danger">İsim</td>
        <td class="text-danger">Yorum</td>
    </thead>
        <tbody>
            @foreach (var item in Model)
            {
                <tr>
                    <td>@item.AdSoyad</td>
                    <td>@Html.Raw(item.Yorum)</td>
                </tr>
            }
        </tbody>
</table>
```

Şekil 2.20: Yorum Listeleme Sayfası Front-End Kodları

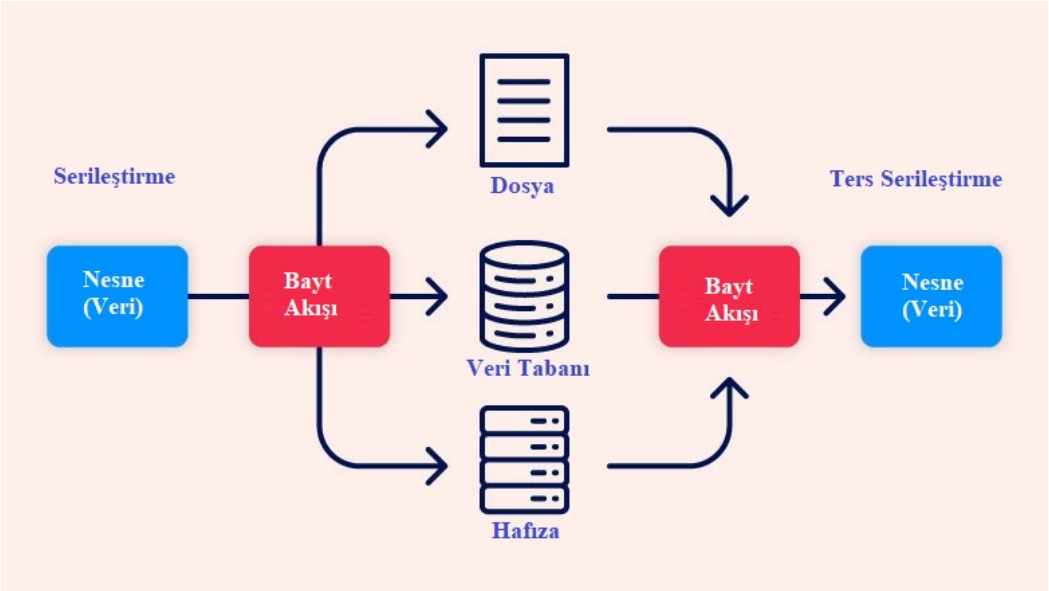
Şekil 2.20'de görüldüğü gibi Front-End kodlarına XSS saldırı kodu içerikli ifadenin çalışabilmesi için "Html.Raw"[52] fonksiyonunun içerisinde çalıştırılması gerekmektedir. XSS zafiyetlerinden korunabilmek için bu ve benzeri fonksiyonların gerekmedikçe kullanılmaması da önem taşımaktadır. Ancak bu fonksiyon çok satırlı uzun paragrafların ve metinlerin düzgün görünmesi için gerekli olduğundan bazı durumlarda kullanılması zorunludur. Bu sebeple XSS zafiyetinden korunmak için en doğru yok input içeriğinin arka yüz kodu içerisinde kontrolünün yapılmasıdır.

Diğer programlama dilleri ve teknolojilerde de yukarıda bahsettiğimiz "ValidateInput" benzeri input kontrol etiketleri veya fonksiyonları mevcut olduğu gibi, bunların yerine yazılımcı tarafından uygulamanın kendi ihtiyaçlarına özgü kontroller de yazılabilmektedir. Örneğin girilen input metninin içerisinde "<script>" ifadesi ve buna benzer anahtar ifadelerin bulunup bulunmadığı kontrol edilerek XSS zafiyetinin önüne geçilebilmektedir.

[52] HtmlHelper.Raw Method, https://docs.microsoft.com/en-us/dotnet/api/system.web.mvc.htmlhelper.raw (05.03.2021)

2.8 GÜVENSİZ SERİLEŞTİRME (INSECURE DESERIALIZATION)

Insecure Deserialization verilerin taşınması sırasında içeriğin bilgisayar diline çevrilmesi (serialization) ve ardından tersi işlemle (deserialization) orijinal haline tekrar restore edilmesi işlemleri sırasında ortaya çıkan bir güvenlik açığı türüdür. Nadir rastlanan ve saldırgan tarafından zor keşfedilen bir zafiyet olmakla birlikte tespit edildiğinde ciddi saldırılara yol açabilir.



Şekil 2.21: Serialization ve Deserialization İşlemlerini Gösteren Grafik [53]

Zafiyet, kullanıcı tarafından serialization işlemi ile gönderilen verinin uygulama tarafından deserialization işlemi yapılması esnasında gerçekleşir. Bu iki işlem arasında saldırgan araya girerek serialization işleminden geçmiş veriler üzerinde değişiklik yapar ve zararlı bir verinin uygulamaya ulaşmasına sebep olur. Hatta tamamıyla farklı bir verinin uygulamaya ulaşmasını sağlayabilir.

[53] Insecure Deserialization, https.//portswigger.net/web-security/deserialization (09.03.2021)

2.8.1 Önlemler

Bu zafiyet için alınması gereken en önemli önlem kullanıcı inputlarının çok gerekmedikçe deserialization işleminden geçirilmemesidir. Eğer bu işlemin yapılması gerekiyorsa verilerin tahrif edilmediğinden emin olmak için gerekli önlemler alınmalıdır. Örnek olarak veri bütünlüğünü kontrol etmek için dijital imzalar kullanılabilir. Ancak burada önemli olan deserialization işlemine başlamadan önce bu kontrolü yapmaktır. [54]

2.9 BİLİNEN ZAFİYETLERE SAHİP BİLEŞENLERİ KULLANMA

OWASP tarafından "Using Components with Known Vulnerabilities" olarak anılan güvenlik açığı, web uygulamalarında kullanılan üçüncü parti modüllerin sahip olduğu zafiyetler sebebiyle ortaya çıkar. Bu modüller farklı programlama dillerine göre çeşitlenmekte olup; yazılım kütüphaneleri, framework ve diğer benzeri paket araçları içerir.

Çok yaygın bir güvenlik açığı türü olup, bir bileşendeki güvenlik açığı keşfedildiği anda o bileşeni kullanan tüm web uygulamaları risk altına girer. Uygulamadaki zafiyetin ne gibi sonuçlar doğuracağı da bileşenin barındırdığı zafiyete bağlıdır. Bazen hafif sonuçları olabileceği gibi; veri kaybı, hatta sunucunun ele geçirilmesi gibi çok ağır sonuçlar da doğurabilir.

2.9.1 Önlemler

Bilinen zafiyetlere sahip bileşenleri kullanma zafiyetinden korunmak için yapılması gereken şeylerin başında gereksiz bağımlılıkları (dependency) kaldırmak gelmektedir. Kullanılması gereken bileşenlerin ise imzalı olmasına, güvenilir bir

[54] Insecure Deserialization, https.//portswigger.net/web-security/deserialization (09.03.2021)

üretici tarafından geliştirilmiş olmasına ve bileşenin indirildiği sistemin de resmi kaynaklar olmasına dikkat edilmesi gerekmektedir. Yazılım yaşam döngüsü içerisinde bu bileşenlerin bakımlarının yapılması, gerektiğinde güncellenmesi önem arz etmektedir.

2.10 YETERSİZ LOGLAMA VE İZLEME

OWASP tarafından "Insufficient Logging & Monitoring" olarak anılan güvenlik açığı, saldırganların web uygulamasına saldırma riskini artıran zafiyetlerden bir tanesidir. OWASP İLK 10 listesinde son sırada bulunan bu zafiyet türü özellikle saldırı sonrasını ilgilendiren bir durumdur.

Loglama ve İzleme yeterli olmadığında saldırıların kalıcı hale gelmesi de söz konusudur. Bu konuda IBM ve Ponemon Institiute tarafından yapılmış ve OWASP'ın da referans aldığı bir araştırma web uygulamalarındaki ihlallerin tespit süresinin ortalama 191 gün sürdüğünü göstermektedir.[55] Daha da ilginci tespitler genellikle dahili değil harici kaynaklar tarafından yapılmaktadır.

2.10.1 Önlemler

Loglama ve izleme yönetiminde uyulması gereken bazı kurallar bulunmaktadır. Tüm kullanıcı giriş ve çıkışları, erişim kontrol ihlalleri, ihlal teşebbüsleri veya erişim kontrolünde yaşanan hataların ve benzeri işlemler; daha sonra yapılması muhtemel bir adli analize sunulabileceği düşünülerek kayıt altına alınmalıdır. Log Kayıtları dünya genelinde kabul görmüş standartlara uygun formatta tutulmalı ve bu konudaki profesyonel yazılım çözümleri tarafından okunabilmelidir.[56]

[55] 2017 Cost of Data Breach Study, https.//www.ibm.com/downloads/cas/ZYKLN2E3 (10.03.2021)
[56] A10:2017-Insufficient Logging & Monitoring, https.//owasp.org/www-project-top-ten/2017/A10_2017-Insufficient_Logging%2526Monitoring.html (10.03.2021)

Şimdiye kadar OWASP Top 10 listesinde bulunan web uygulama güvenlik açıklarını inceledik. Bu listede bulunmadığı halde sıkça saldırılara konu olan saldırı türlerinden de bazılarını incelemeye devam edeceğiz.

2.11 KABA KUVVET SALDIRISI (BRUTE FORCE ATTACK)

Kaba Kuvvet Saldırısı olarak çevrilen Brute Force Saldırıları özellikle kullanıcı giriş-çıkış bileşenlerine sahip web uygulamalarının muhatap olduğu bir tehlikedir. Brute Force saldırısında, kullanıcı şifreleri veya pin kodları gibi şifreler deneme-yanılma yöntemiyle çözülmeye çalışılır.

Brute Force saldırılarının nasıl çalıştığını anlamak zor değildir. Çeşitli algoritmalar ve yardımcı programlar vasıtasıyla, saldırgan binlerce hatta milyonlarca deneme yaparak şifreyi kırmaya çalışır.

Kaba Kuvvet saldırıları aşağıdaki türler altında incelenir[57]:

- Basit Kaba Kuvvet Saldırıları (Simple Brute Force Attacks): Saldırgan sistematik bir yaklaşımla, bir dış mantığı dayanmadan, şifreyi tahmin etmek için çok sayıda deneme yapar. Bu saldırı, deneme sayısında sınır bulunmayan durumlarda kullanılır.
- Hibrid Kaba Kuvvet Saldırıları (Hybrid Brrute Force Attacks): Bu saldırıda tüm şifreler yerine bir dış mantık çerçevesinde olasılığı en yüksek şifreler denenir. Sözlükteki kelimelere sayı ekleme veya harflerin yerini değiştirme gibi küçük değişiklikler denenir.

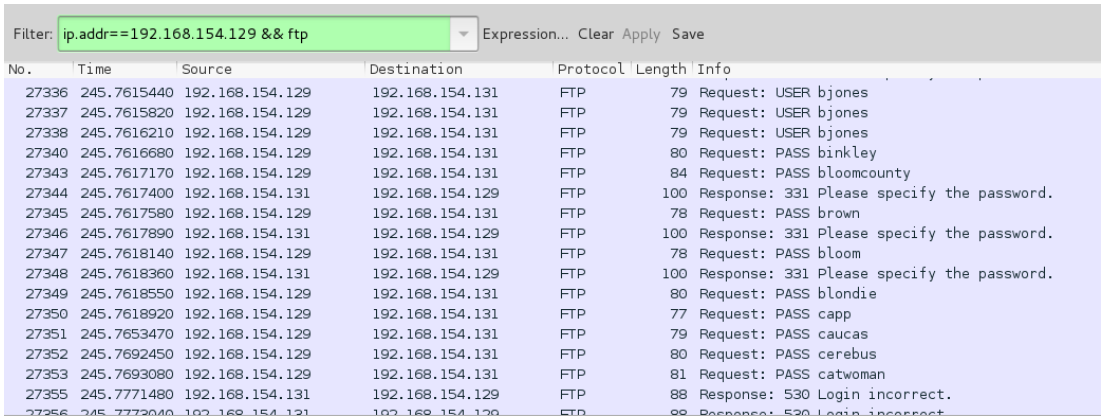
[57] Brute Force Attack, https.//www.imperva.com/learn/application-security/brute-force-attack/ (12.03.2021)

- Sözlük Saldırıları (Dictionary Attacks): Olasılık üzerine kurulu bir saldırıdır. Bir mantık çerçevesinde şifre deneme sözlüğü oluşturulur ve bunlar denenerek şifre kırılmaya çalışılır.
- Gökkuşağı Tablosu Saldırıları (Rainbow Table Attacks): Bir gökkuşağı tablosu kullanılır. Bu Hash fonksiyonlarını tersine çeviren önceden hesaplanmış bir tablodur.
- Ters Kaba Kuvvet Saldırısı (Revers Brute Force Attacks): Saldırgan, ele geçirdiği bir parola veya parola listesini olası tüm kullanıcı adları için dener.
- Kimlik Bilgisi Doldurma (Credential Stuffing): Saldırgan, önceden ele geçirdiği kullanıcı adı ve şifre çiftlerini birçok web sitesinde dener. Kullanıcıların farklı web uygulamalarında aynı şifreyi kullanması fenomeninden faydalanır.

Brute Force saldırılarını gerçekleştirmek için saldırganların yararlandığı bazı yazılımlar ve araçlar vardır. Bunların en bilinenlerinin isimleri şu şekilde sıralanabilir[58]:

- THC Hydra (En yaygın Brute Force saldırı aracıdır.)
- Aircrack-ng
- John the Ripper
- L0pthCrack
- HashCat
- DaveGrohl
- Ncrack

[58] Popular Tools for Brute-force Attacks, https.//resources.infosecinstitute.com/topic/popular-tools-for-brute-force-attacks (12.03.2021)



Şekil 2.22: Brute Force Denemelerinin WireShark ile Tespiti [59]

2.11.1 Önlemler

Brute Force saldırıları sayısal olarak en yayın saldırı türü olmasına rağmen, günümüzde etkinliği sınırlıdır. Web uygulamasını bu tür saldırılardan korumak için uyulması gereken bazı kurallar vardır:

- Kullanıcıların şifreleri belirlenirken güçlü standartlar getirilmeli bu standartlara uymayan şifreler kabul edilmemelidir.
- Kullanıcı giriş ekranında mutlaka güvenilir bir Captcha (İnsan ve Bilgisayar Ayrımı Amaçlı Tam Otomatik Genel Turing Testi[60]) aracı bulunmalıdır.
- Bir istemci arka arkaya çok sayıda şifre denemesi yaptığında belli bir süreyle bloke edilmelidir. Her denemeden sonra bloke edilme süresi uzatılmalı ve bir limite ulaştığında tamamen bloke edilmelidir. Uygulama yöneticisi yaptığı kontrolden sonra ancak istemcinin kötü niyetli olmadığından emin olursa kilidini açmalıdır.
- Riskin çok yüksek olduğu uygulamalarda iki faktörlü kimlik doğrulama yöntemi uygulanmalı, bankaların web uygulamalarında olduğu gibi kullanıcı girişi için telefon veya e-posta gibi kullanıcıya ait diğer iletişim araçlarıyla kimlik teyidi yapılmalıdır.

[59] Online Brute Force Attacks, https.//godlikesecurity.com/index.php/tag/brute-force (11.03.2021)
[60] CAPTCHA Nedir, https.//support.google.com/a/answer/1217728?hl=tr (12.03.2021)

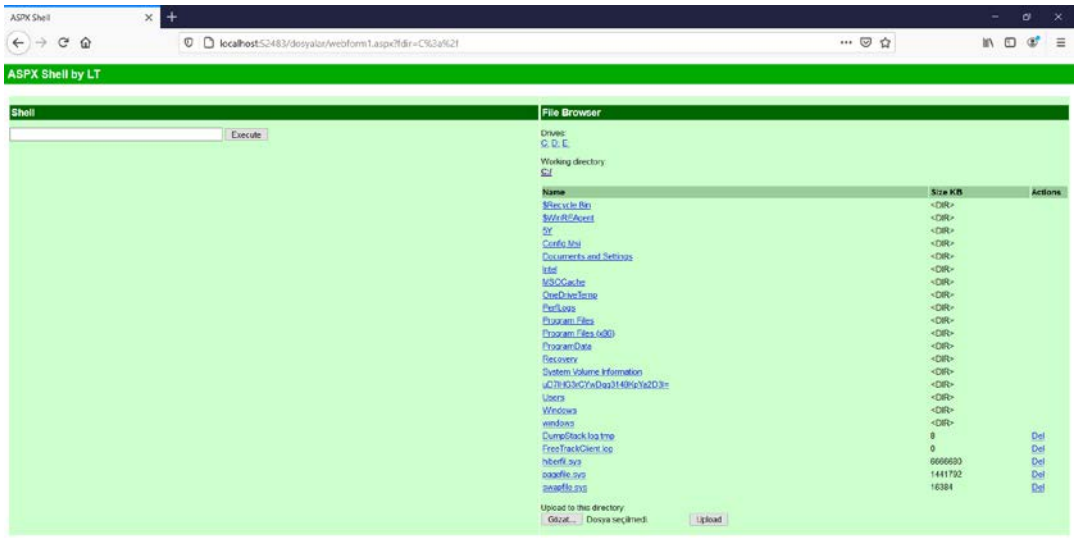


Şekil 2.23: Örnek Captcha Uygulaması

2.12 KISITSIZ DOSYA YÜKLEME (UNRESTRICTED FILE UPLOAD)

Web Uygulamalarında sık sık dosya yükleme bileşenlerine ihtiyaç duyulmaktadır. Örneğin bir web sitesinin yönetici panelinde text-editör ile de eşgüdümlü çalışabilen çeşitli dosya yöneticileri (File Manager) araçları söz konusudur. Bunun yanı sıra Web Tarayıcılarının entegre dosya yükleme (upload) bileşeni de HTML işaretleme dilinin "Input File" fonksiyonuyla kullanılabilmektedir. Bu şekilde web uygulamasına dosya ekleme işlemlerinde eğer dosya uzantıları kontrol edilmezse ve bunun yanında eklenen dosya web tarayıcıya yazılan bir link ile çalıştırılabiliyorsa bu durum çok ciddi bir güvenlik açığına neden olmaktadır.

Bu güvenlik açığını daha detaylı olarak bir örnek üzerinden açıklayalım. Bir ASP.NET web uygulamasında dosya upload bileşenine herhangi bir uzantıya sahip dosyanın kod tarafında bir kontrol yapılmadan yüklendiğini ve bu yüklenen dosyanın da "www.siteadi.com/dosyalar/" klasörü altına eklendiğini varsayalım. Bu durumda saldırgan Windows'un "Komut İstemi" programına benzer özelliklere sahip ".aspx" uzantılı bir Shell yazılımını uygulamaya upload edebilecektir. Ardından ilgili linkten dosyayı çalıştırıp web uygulamasının çalıştığı web sunucusunun tüm dosya sistemini ele geçirebilecektir.



Şekil 2.24: Shell Yazılımının Web Tarayıcıda Çalıştırılması

Şekil 2.24'te görüldüğü gibi web uygulamasına yüklenen ".aspx" uzantılı Shell yazılımı ilgili sunucunun dosya sistemine erişebilmekte ve en kritik dosyalar (örneğine Windows klasörü içindeki dosyalar) bili silinebilmekte veya değiştirilebilmektedir. Bu da en yüksek risk seviyesine mensup bir güvenlik açığı türüdür.

2.12.1 Önlemler

Kısıtlanmamış dosya yükleme veya uzaktan çalıştırılabilir dosya yükleme olarak anılan bu zafiyet durumuna karşı alınması gereken önlemler için Kaynak Kodunda bazı kontroller yazılmalıdır. Yüklenen dosyanın uzantı bilgisi ve bunun yanında Content bilgisi kontrol edilmeli ve ancak bu kontrolden sonra dosyanın yüklenmesine izin verilmelidir.

Bu aşamada yapılacak kontrollerle ilgili dikkat edilmesi gereken bazı noktalar vardır[61]:

- Yüklenecek dosya uzantıları belirlenirken, izin verilmeyenler listesi değil, izin verilenler listesi yapılmalı ve kontrol bu şekilde sağlanmalıdır.
- Sadece uygulamada ihtiyaç duyulan dosya uzantılarına izin verilmelidir.
- Dosya uzantısının yanında dosya içeriği (Content) de kontrol edilmelidir. Content Check denilen bu işlemde yine sadece ihtiyaç duyulan Content tiplerine izin verilmelidir.
- Dosya isimlerindeki bazı özel karaktere izin verilmemelidir. Bu özel karakterler ile sunucuda yeni klasör açma gibi bazı işlemler tetiklenebilmektedir.
- Dosya isminin uzunluğu kısıtlanmalıdır.

Bu aşamada dosya uzantılarını sınırlandırma yönteminin etkinliği incelemek için bir ASP.NET MVC tabanlı web uygulaması yazılmış ve aşağıda görülen, dosya uzantısını kontrol eden kod bloklarının izin verilen dosya türleri haricindeki dosyalara izin vermediği görülmüştür.

```
public static bool OnaylananFormatMi(string uzanti)
{
    List<string> dizi = new List<string>();
    dizi.Add(".png");
    dizi.Add(".jpg");
    dizi.Add(".jpeg");
    dizi.Add(".gif");
    dizi.Add(".pdf");
    dizi.Add(".doc");
    dizi.Add(".docx");
    dizi.Add(".xlsx");
    dizi.Add(".xls");
    dizi.Add(".ppt");
    dizi.Add(".pptx");

    if (dizi.Contains(uzanti))
    {
        return true;
    }
```

```
if (Araclar.OnaylananFormatMi(ext))
{
    string yol = "/files/" + fileUpload.FileName;
    var d = ctx.Dosya.FirstOrDefault(x => x.DosyaYolu == yol);
    if (d == null)
    {
        var fileName = Path.GetFileName(fileUpload.FileName);
        var path = Path.Combine(Server.MapPath("/files/"), fileName);
        fileUpload.SaveAs(path);

        Dosya dosya = new Dosya();
        dosya.Aciklama = Aciklama;
        dosya.DosyaYolu = Path.Combine(("/files/"), fileName);
        ctx.Dosya.InsertOnSubmit(dosya);
        ctx.SubmitChanges();
    }
```

Şekil 2.25: Dosya Uzantısını Kontrol Eden Kod Blokları

[61] Unrestricted File Upload, https.//owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload (12.03.2021)

2.13 SİTELER ARASI İSTEK SAHTECİLİĞİ (CROSS SITE REQUEST FORGERY)

Siteler Arası İstek Sahteciliği olarak çevrilen Cross Site Request Forgery, CSRF veya XSRF kısaltmalarıyla anılmaktadır. Web uygulamasında bulunan bir zafiyetten faydalanan saldırgan, kendisini uygulamaya giriş yapmış bir kullanıcı olarak gösterme yoluyla web uygulamasına sahte istek göndererek uygulamaya sızma gerçekleştirir. Çoğu web sitesi bu saldırılara karşı korunaksız[62] olduğu için, web uygulama zafiyetleri içerisindeki "uyuyan dev"[63] olarak nitelenmiştir.

CSRF saldırılarında, eğer saldırıdan etkilenen bir uygulama kullanıcısı ise saldırının etkisi kullanıcının hesap bilgilerinin değiştirilmesi veya bir finans uygulaması söz konusu ise hesabından para çekilmesi şeklinde gerçekleşebilir. Eğer saldırıdan etkilenen bir yönetici hesabı ise bu durum saldırı sonucu tüm uygulamanın güvenliği tehlikeye girebilir[64].

CSRF saldırılarının nasıl gerçekleştiğini birkaç örnek üzerinden inceleyeceğiz. Eğer web uygulaması GET metotları ile verilerde bir değişiklik yapılmasına izin veriyorsa bu ciddi bir güvenlik açığı oluşturur. Bu durumda bir kullanıcının hesap bilgileri ile CSRF saldırıları gerçekleştirilebilir. (OWASP)

http://www.ornek.com/telefon_no_degistir?KullaniciID=1023459966&Telefon=05551112233

Yukarıdaki gibi bir GET komutu yazılarak "1023459966" ID numarası önceden öğrenilen bir kullanıcının telefon numarası bir CSRF saldırısı ile

[62] William Zeller ve Edward W. Felten, Cross-Site Request Forgeries. Exploitation and Prevention, *International Journal of Computer Applications,* 106(11):20-25, s. 20
[63] J. Grossman. CSRF, The Sleeping Giant, 2006, s. 19
[64] Cross Site Request Forgery (CSRF), https.//owasp.org/www-community/attacks/csrf (13.03.2021)

değiştirilebilir. Benzer bir durum POST metodu için de geçerlidir. Kullanıcıyla ilgili bazı bilgiler ele geçirilir ve bir POST metodu oluşturulur. Kullanıcı uygulamaya giriş yapıp daha sonra başka bir sayfaya yönelmiş ama uygulamadan çıkış yapmamıştır. Yöneldiği diğer sayfada aşağıdaki gibi bir HTML formuna tıklaması sağlanarak saldırı gerçekleştirilmiş olur.

```
<form action="http://ornek.com/telefon_no_degistir" method="POST">
    <input type="hidden" name="KullaniciID" value="1023459966"/>
    <input type="hidden" name="Telefon" value="05551112233"/>
    <input type="submit" value="Resmi görmek için tıklayınız"/>
</form>
```

Yukarıda yazılan HTML kodunda POST metoduyla çalışan bir Form bulunmaktadır. Ancak ilgili kullanıcı, kendi ekranında sadece “Resmi görmek için tıklayınız” şeklinde bir metin görecektir. Metne tıkladığında form, ilgili bilgileri POST metodu aracılığıyla saldırılmak istenen web uygulamasına iletecek ve saldırganın istediği doğrultuda kullanıcı bilgileri değiştirilecektir. Bu da uygulama açısından çok ciddi bir güvenlik açığı bulunduğunu göstermektedir.

2.13.1 Önlemler

Cross Site Request Forgery saldırılarını engellemek için dikkat edilmesi gereken bazı kurallar ve uygulamaya eklenmesi gereken bazı kontroller bulunmaktadır Bunları maddeler halinde öyle sıralayabiliriz:

- Uygulama verilerinde herhangi bir değişiklikler asla GET metoduyla yapılmamalıdır. Bu tip durumlarda her zaman POST metot kullanılmalıdır.
- POST metodunun güvenliğini sağlamak için ilgili HTML formunun içerisine bir Token anahtar değeri eklenmeli ve kod tarafında kontrolü yapılmalıdır. Bu

anahtar değerine sahip olmayan formlar kabul edilmemelidir. Aşağıda ASP.NET teknolojisi ile oluşturulmuş örnek bir Token anahtarı görülmektedir.

- Kullanıcıların oturumlarını kapatmadan sayfayı kapatmaları ihtimaline karşılık, hareketsiz kalınan belli bir süre sonunda uygulama tarafından kullanıcı hesabına otomatik çıkış yaptırılmalıdır.

Bu aşamada token anahtarını doğrulama yöntemiyle CSRF saldırısını önleme metodunun etkinliğini araştırmak için bir çalışma yapılmıştır. ASP.NET MVC teknolojisiyle hazırlanan bir uygulamada, arka yüz kodu içerisinde yazılan bir HTML formunun içerisinde "@Html.AntiFergeryToken()" kodu yazıldığında aşağıdaki gibi bir Token oluşmuştur.

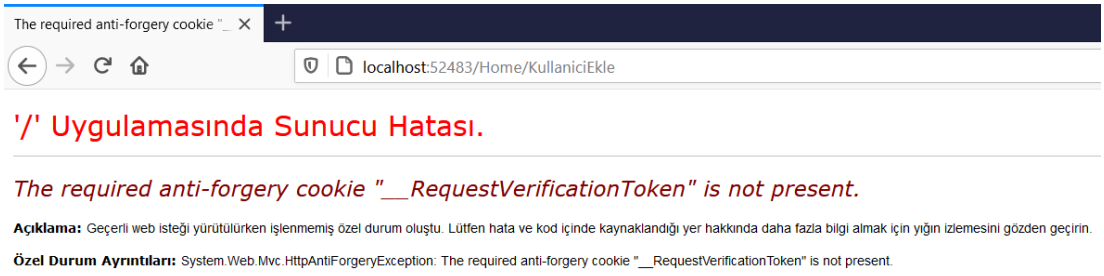
```
<input name="__RequestVerificationToken" type="hidden"
value="n9oI_ZOzB8bLiwocMfbne0v8wwyNmuUReI4OlfXpMd6eFBsHDB0ft
7feQ0bXtqPep3oaED5579rboOaXQg1z1pFIfBI-EcNTf72Ns44j0Ig1" />
```

Uygulamanın Back-End kodlarında Control bölümünde ilgili metodun üzerinde "ValidateAntiForgeryToken" özniteliği eklenmiştir ve Token doğrulamasının yapılması sağlanmıştır.

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult KullanıcıEkle(string kullaniciadi, string sifre, string email)...
```

Uygulama HTML formunun içerisinde "@Html.AntiFergeryToken()" kodunun olduğu durumda başarıyla çalışmıştır. Ancak formunda içerisinde bu kodun olmadığı durumda "ValidateAntiForgeryToken" özniteliğinin tetiklediği fonksiyon

kendisine ulaşan form inputları içerisinde, uygulama tarafından üretilmiş bir token bulunmadığı için uygulama hata sayfası üretmiş ve CSRF saldırısını engellemiştir.



Şekil 2.26: CSRF Saldırısı Sonrası Oluşan Hata Sayfası

Tez çalışmasının ikinci bölümünde 13 adet web uygulama güvenlik açığı incelenmiş, oluşturdukları riskler ve önleme yöntemleri üzerinde deneyler yapılarak veri toplanmıştır. Bir sonraki aşamada bu güvenlik açıklarının analiz ve tespit yöntemleri incelenecektir.

ÜÇÜNCÜ BÖLÜM

WEB UYGULAMALARINDA ZAFİYET TESPİT YÖNTEMLERİ

Günümüzde web uygulaması yöneten kişiler ve kuruluşlar yönettikleri uygulamaların tam olarak güvende olmadığının ve çeşitli saldırılara açık olabileceğinin farkında varmışlardır. Bu nedenle uygulamaların çeşitli test ve analizlerle mümkün olan en az güvenlik zafiyetine sahip olması hedeflenmektedir[65].

Web uygulamalarında bulunan güvenlik açıklarını tespit ve analiz etmek, bununla birlikte zafiyetlerin giderilmesi için gerekli önlemli belirlemek için statik kod analizi, kod gözden geçirme, sızma (penetrasyon) testi ve zafiyet taraması gibi bir takım genel kabul görmüş test yöntemleri vardır.

3.1 STATİK KOD ANALİZİ

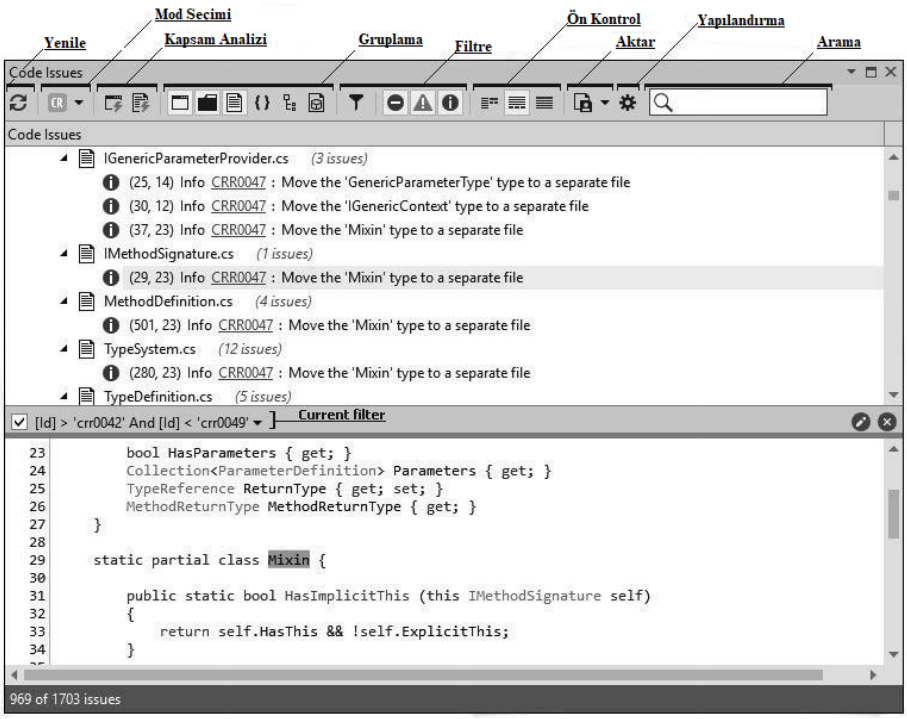
Statik kod analizi veya kaynak kod analizi, uygulamaya ait kaynak kodların bir kod analiz yazılımı vasıtasıyla incelendiği bir zafiyet analiz faaliyetidir. Yazılım yaşam döngüsünde, uygulamanın geliştirilme aşamasında yapılan bir faaliyettir ve kod yazılırken veya yazımın hemen sonrasında statik kod analizi araçlarının çalıştırılmasını ifade eder.

Kaynak kod analizinde, bozukluk analizi (taint analysis) ve veri akışı analizi (data flow analysis) gibi teknik kullanılır[66]. Veri akışı analizi, yazılım statik durumdayken yazılımdaki veriler hakkında çalışma zamanı bilgileri toplamak için

[65] Kaynak Kod Analizi, https.//www.prismacsi.com/kaynak-kod-analizi (11.03.2021)
[66] Static Code Analysis, https.//owasp.org/www-community/controls/Static_Code_Analysis (11.03.2021)

kullanılır[67]. Bozukluk analizi ise kullanıcı tarafından girilmesi beklenen verilerle ilgili değişkenlerin ortaya çıkarabileceği zafiyetleri tespit eder.

Kaynak kod analiz araçları, uygulamaların geliştirildiği Tümleşik Geliştirme Ortamı (IDE) tarafından sunulan biz hizmet olabilir, ilgili IDE'ye ektenti olarak eklenebilen bir yapıda olabilir veya IDE'nin dışında ayrı bir araç olarak çalışabilir. IDE ile birlikte çalışan araçlar daha kod yazım aşamasında yapılan bazı hataların veya ortaya çıkan zafiyetlerin tespitinde faydalı olmaktadır. Birden fazla araçla aynı kodun incelenmesi ise riski azaltmak açısından önem arz etmektedir.



Şekil 3.1: VS Üzerinde Çalışan CodeRush Kod Analiz Aracı [68]

[67] Wögerer, Wolfgang, *A Survey of Static Program Analysis Techniques*, Technische Universität Wien, 2005, s. 2-5

[68] Static Code Analysis, https.//docs.devexpress.com/CodeRushForRoslyn/115805/static-code-analysis (11.03.2021)

3.1.1 Statik Kod Analizinde Kullanılan Araçlar

Statik Kod Analizi araçları IDE dışında kullanılan türleri genellikle ücretli yazılımlar olup IDE içerisinde kullanılan kod analiz araçlarını tamamlayıcı niteliktedir. Ücretli araçlar haliyle daha profesyonel ve daha fazla çıktı üreten araçlardır.

Statik Kod Analizi yapan araçlar işlevlerine göre 3 tipte incelenmektedir[69].

- Statik Uygulama Güvenlik Test Araçları (SAST Tools),
- Uygulama Bileşenlerinin (Framework, Library) Güncelliğini Denetleyen Araçlar,
- Statik Kod Kalitesi Denetimi Araçları

Statik kod analizi için kullanılacak araçlar seçilirken dikkat edilmesi gereken bazı kriterler vardır. OWASP tarafından belirlenen bu kriterler şunlardır[70]:

- İhtiyaç: Öncelikle seçilecek kod analizi aracının uygulamalarda kullanılan programlama dillerini karşılayıp karşılamadığı kontrol edilmelidir.
- Tespit edebildiği zafiyetler: Bu konuda, araçların karşılaştırmalı testlerinin yapıldığı raporları incelemek faydalı olacaktır. Bu konuda da en iyi kaynaklardan birisi yine OWASP'tır.
- Etkinlik oranı: Yine aynı raporlara göre kod analizi araçlarının Yanlış Alarm (False-Positive) oranları karşılaştırılmalı ve buna göre bir seçim yapılmalıdır.
- Kod analizi aracı, gerekli durumlarda kaynak kod yerine Binary[71] kod üzerinde de çalışabilmelidir.
- İlgili kod analizi aracı, web uygulamasında kullanılan library, framework gibi bileşenleri desteklemelidir.

[69] Free for Open Source Application Security Tools, https.//owasp.org/www-community/Free_for_Open_Source_Application_Security_Tools (11.03.2021)
[70] Secure Code Analysis Tools, https.//owasp.org/www-community/Source_Code_Analysis_Tools (11.03.2021)
[71] Binary code, https.//www.britannica.com/technology/binary-code (11.03.2021)

- İlgili aracın, web uygulamasının geliştirildiği IDE yazılımına entegre çalışabilmesi gerekmektedir.
- İlgili aracın kolay kurulabilir ve kolay kullanılabilir olması özellikle takım çalışmasında faydalı olacaktır.
- İlgili araç istenilen durumda sürekli çalışabilmeli ve kaynak koda eklenen kodları anlık olarak denetleyebilmelidir.
- İlgili araç maliyet etkin bir çözüm sunmalıdır.

Java, C#, Python gibi çok kullanılan diller için geliştirilmiş Kod Analizi araçlarının en popüler olanlarını maddeler halinde şu şekilde sıralayabiliriz[72]:

- Raxis
- RIPS Technologies
- PVS-Studio
- Kiuwan
- reshift
- Embold
- SmartBear Collaborator
- CodeScene Behavioral Code Analysis
- Visual Expert
- Veracode
- Fortify Static Code Analyzer
- Parasoft
- Coverity
- CAST
- CodeSonar
- Understand
- Code Compare

[72] TOP 40 Static Code Analysis Tools (Best Source Code Analysis Tools), https.//www.softwaretestinghelp.com/tools/top-40-static-code-analysis-tools (11.03.2021)

3.1.2 Statik Kod Analizinin Artı ve Eksileri

Bir Araç (Tool) ile yapılan Statik Kod Analizlerinin en güçlü ve avantajlı yanlarını şu şekilde sıralayabiliriz[73]:

- Bu araçlar çok sayıda yazılım için kullanılabilir ve defalarca üst üste çalıştırılabilir.
- Arabellek taşmaları ve SQL Enjeksiyona sebep verecek zafiyetlere benzer durumları rahatlıkla tespit edebilirler.
- Çıktı olarak ürettikleri bilgiler geliştiriciler için faydalıdır. Kaynak koddaki bulgunun hangi dosya ve hangi satırda olduğunu net olarak göstermeleri bir avantajdır. Geliştirici buradaki açığı kapatıp tekrar test ederek sonucu hemen görebilir.

Statik kod analiz araçlarının olumsuz yanlarından en önemlisi ise bazı durumlarda hata bulunmayan kodları hatalı gibi gösterebilmesidir. Bu durum literatürde "false positive" olarak tanımlanmaktadır. Bu araçların kendisi de bir yazılım olduğundan bazı durumlarda gerçek zafiyet ile zafiyet olmayan durumları ayırt edememekte veya kritik önemde olmayan bir zafiyeti kritik önemde gösterebilmektedir. Bu sebeple kaynak kodların bir analiz ekibi tarafından statik kod analizinden farklı olarak, bir sonraki bölümde inceleyeceğimiz gibi kod gözden geçirme incelemesine tutulması önem taşımaktadır.

Statik kod analizi tek başına yapıldığında istenen sonucu verememekte, ilgili kaynak kodun geliştirici ekip dışında farklı bir uygulama güvenlik uzmanı ekip tarafından testlere tabi tutulması gerekmektedir.

[73] Source Code Analysis Tools, https.//owasp.org/www-community/Source_Code_Analysis_Tools (11.03.2021)

Statik Kod Analizi araçlarının genel olarak dezavantajları aşağıdaki maddelerde sıralanmıştır:

- Bu araçlar ile kaynak koddaki mantık hatalarının tespit edilmesi mümkün değildir.
- Kullanıcı giriş çıkış işlemleri, erişim yönetimi gibi bileşenlerdeki zafiyetlerin Statik Kod Analizi araçlarıyla tespit edilmesi çoğu durumda mümkün değildir.
- Yapılandırma ve Yönlendirme hatalarını ve zafiyetlerini bulmaları çoğu durumda mümkün değildir.
- Çok fazla false-positive üretmeleri bazı durumlarda binlerce hata bulunmuş gibi bir liste üretmekte, bu durum da listenin geliştirici tarafından okunmasını zorlaştırmaktadır.
- Tespit ettikleri hata ve zafiyetlerin kritiklik seviyesinde hata yapabilmektedirler.

Kod analizi çalışmaları içerisinde, statik kod analizinin yanında dinamik kod analizi adlı güvenlik analiz çalışması bulunmaktadır. Statik kod analizi kaynak koda erişilerek yapılan bir çalışma iken, dinamik kod analizi kaynak kodun derlenmesinin ardından yapılan ve çalışma zamanı (run-time) hatalarının tespiti için kullanılan bir yöntemdir. Dinamik kod analizi ile uygulamanın çalışır vaziyetteki davranışları gözlenmektedir[74]. Dinamik kod analizi çalışmaları, uygulamanın kodlanmasının tamamlanması sonrasındaki dönemde yapılan bir çalışma olduğundan tespit edilen hataların geliştirici tarafından giderilmesi zorlaşmaktadır[75]. Statik kod analizi çalışmalarını güçlendirmek için kaynak kodun derlenmesi ve dinamik kod analiz araçları ile de güvenlik analizine tâbi tutulması tüm zafiyetlerin tespitinde faydalı olabilmektedir.

[74] Akcura Korhan, Shalchian Reze, Patil Abhijit, Singh Rattandeep, Tanna Jay, Static Versus Dynamic Source Code Analysis, *Electronic Notes in Theoretical Computer Science*, 2005, 131(2):3-14 s. 3, DOI:10.1016/j.entcs.2005.01.018

[75] Dülger Berk, Tanlak Mehmet (December 2017), Kod Kalitesi Ve Kod Analizi Serisi #4, https://keytorc.com/blog/kod-kalitesi-ve-kod-analizi-serisi-4-dynamic-code-analysis (29.06.2021)

3.2 KOD GÖZDEN GEÇİRME

OWASP'ın yayınladığı Code Review Guide dokümanına göre kod gözden geçirme (Code Review) tekniği, zafiyet analizi yöntemleri içerisinde en etkili yöntemdir[76].

Kod gözden geçirme tekniğinde bir analiz ekibi tarafından ilgili uygulamanın kaynak kodları incelenir. Analiz ekibi kod yapısını ve iş akışını ne kadar iyi anlarsa faaliyetin etkinliği o kadar artacaktır. Bundan dolayı analiz ekininin, bazı aşamalarda yazılım geliştiren ekiple birlikte çalışması faydalı olmaktadır.

Analiz ekibi tarafından kod gözden geçirme faaliyetine başlamadan önce aşağıdaki soruların cevabının alınmış olması gerekmektedir:

- Uygulama hangi teknolojileri kullanıyor?
- Uygulama ne kadar kod satırı içeriyor?
- Uygulama ne kadar hassas ve gizli veri barındırıyor?
- Uygulama nerede çalışacak? Sadece kurum için bir uygulama mı yoksa internette mi hizmet verecek?
- Uygulamanın yönetici paneli kurum içinde mi yoksa internette mi hizmet verecek?

Kod gözden geçirme çalışmalarında uygulamanın barındırdığı teknoloji ve tekniklere göre nelerin analiz edilmesi ve doğrulanması gerektiğine dair bir liste çıkarılmalıdır. Bu liste genel olarak aşağıdaki bileşenleri içermektedir[77]:

- Yazılım geliştirme standartlarına bağlılık,
- Kullanıcı kimlik doğrulama işlemleri,

[76] OWASP Code Review Guide, https.//owasp.org/www-pdf-archive/OWASP_Code_Review_Guide_v2.pdf (11.03.2021)
[77] Kaynak Kod Analizi, https.//www.prismacsi.com/kaynak-kod-analizi (11.03.2021)

- Oturum (Session) yönetimi,
- Log yönetimi,
- Şifreler ve Hash işlemleri,
- Yetkilendirme işlemleri,
- Hata yakalama ve hata yönetimi,
- Dış kaynaklı yazılım bileşenlerinin güvenilirliği,
- Katmanlı mimari standartlarına uyum,
- Güvenlik yapılandırması,
- Yönlendirme (Routing) yapılandırması,
- Veri tabanı ilişki metotları

Kod gözden geçirme faaliyetlerinde keşif aşamasından sonra Tehdit Modelleme aşaması gelmektedir. Tehdit modellemede amaç güvenlik sorunlarının bileşen ve uygulama düzeyinde dikkate alınmasını sağlamaktır. Tehdit tanımları ve modelleme yöntemleri bir yazılımın nasıl bir saldırıya maruz kalabileceğine, saldırgan türlerine, saldırı eğilimlerine dair kapsamlı bir öngörü sağlar. Bu öngörülerle uygulamaların güvenlik tasarımları devamlı olarak güncellenir[78]. Tehdit Modelleme sayesinde uygulamaların içeriği risk ve zafiyetler derecelendirilir, ölçeklendirilir ve kategorize edilir.

Code Review işlemleri sırasında bir önceki bölümde bahsettiğimiz statik kod analizi araçları da önemli yer tutar. Bu araçlardan alınan çıktılar analiz ekibi için yol gösterici olmaktadır çünkü statik kod analizi araçları binlerce satır kodu çok kısa bir sürede taramakta ve analiz ekibine hem uygulamanın geneli hakkında bir fikir sunmakta hem de analiz ekibinin gözünden kaçan zafiyetleri görmeye yardımcı olmaktadır.

[78] Sarıman, Güncel ve Küçüksille, Ecir Uğur (2013), Güvenli Yazılım Geliştirme Süreci ve Statik Kod Analizi*, 6. Uluslararası Bilgi Güvenliği ve Kriptoloji Konferansı, s. 283.*

Statik kod analizi araçlarından gelen çıktıların, kod gözden geçirme çalışmasını yapan uzman ekip tarafından yorumlanması ise zorunludur çünkü statik analiz araçları birçok yanlış veya eksik çıktı üretebilmektedir.

3.2.1 Kod Gözden Geçirme Çalışmasında Gerekli Denetimler

İlgili uygulamanın kaynak koduna ulaşılarak yapılan, web uygulama güvenliği konusunda uzman bir ekip tarafından yürütülen ve içerik bakımından en geniş çalışma türü olan kod gözden geçirme çalışmalarının içermesi gereken denetimler aşağıdaki maddelerde açıklanmıştır.

3.2.1.1 Yazılım Geliştirme Standartlarına Bağlılık

Uygulama kaynak kodunun, Nesne Yönelimli Programlama, Katmanlı Mimari gibi genel kabul görmüş yazılım standartlarına uygun şekilde tasarlandığı teyit edilmelidir. Bu standartlar hem uygulamanın kalitesi açısından hem de yazılım yaşam döngüsü içerisinde gelecekte yapılacak kaynak kodu değişiklikleri açısından önem taşımaktadır. Katmanlı Mimari yapısı, veri tabanı ile ilgili işlemleri diğer katmanlardan ayırdığı için ileriki dönemde veri tabanı uygulamasında bir değişikliğe gidilirse (örneğin Microsoft SQL Server'dan PostgreSQL'e geçiş gibi) sadece veri tabanı işlemleriyle ilgili katmanda yapılacak bazı değişiklerle bu geçiş mümkün olacaktır. Aksi taktirde kaynak kodunun büyük bölümünün yeniden yazılması gerekir. Yazılım standartlarına uygunluk, web uygulama güvenliği ile ilgili standartlara uyum konusunda da önem taşımaktadır.

3.2.1.2 Kimlik Doğrulama İşlemleri

Kimlik doğrulama işlemlerinde, yetkilendirme gereken tüm sayfaların giriş-çıkış işlemleri gerektirdiğinden emin olunmalı, giriş-çıkış işlemleri POST metoduyla yapılmalıdır. Authorization işlemlerinde yetkisiz girilebilecek yerler istisna olarak tanımlanmalıdır. Aksi durumun istisna olarak tanımlanması hatalara müsait bir yaklaşımdır.[79]

3.2.1.3 Oturum (Session) Yönetimi

Session bilgilerinin oluşturulma yöntemleri, hangi aşamada oluşturulduğu, session bilgisinin nerede tutulduğu önem taşımaktadır[80]. Saldırganların kullanıcı oturumlarını çalabildikleri yöntemlerin uygulama kodunda bulunup bulunmadığın dikkat edilmelidir. Cookie yönetiminin güvenli olup olmadığı incelenmelidir. Session ID'sinin yapısı, uzunluğu, sona erme tarihi, cache işlemlerinin yapısı gibi etmenler kontrol edilmelidir. İstemciye, hareketsiz kalınan bir süre sonra otomatik çıkış yaptırılması önem taşımaktadır.

3.2.1.4 Log Yönetimi

Log Yönetiminin yapıldığı kodlar ve veri tabanı tabloları detaylıca incelenmeli ve her türlü CRUD işlemlerinin kaydının tutulduğundan emin olunmalıdır. Log kayıtlarında şifre bilgisi gibi hassas bilgilerin bulunmadığı teyit edilmelidir. Log kayıtları Web üzerinden erişilebilen yerlerde gösterilmemeli ve sınırlı erişime sahip olmalıdır. Düz metin MIME türü (HTML değil) ile yapılandırılmalıdır. Kayıtlarda;

[79] Tumin, Sharil ve Encheva, Sylvia (2012), A Closer Look at Authentication and Authorization Mechanismsfor Web-based Applications, *5th World Congress: Applied Computing Conference*, s. 100-101.
[80] Session Management in HTTP: How does it work?, https.//www.packetlabs.net/session-management (12.03.2021)

işlemlerin tarihi, ilgili kullanıcı bilgisi, ilgili IP adresi gibi bilgilerin eksiksiz olarak bulunduğundan emin olunmalıdır[81].

3.2.1.5 Şifreler ve Hash İşlemleri

Şifrelerin veri tabanında hash işleminden geçirilerek tutulduğundan emin olunmalıdır. Doğru ve güvenilir hash fonksiyonlarının kullanıldığı teyit edilmelidir. Şifrelerin yönetimi konusunda belirli standartlar bulunduğu, şifrelerde büyük harf, sayısal değer, özel işaret gibi bazı zorunlulukların bulunduğu ve kullanıcıların belirli aralıklarla şifrelerini değiştirmeye zorlandıkları teyit edilmelidir.

3.2.1.6 Yetkilendirme İşlemleri

Yetki sisteminin doğru çalıştığı, erişim kontrol kurallarının yeterli olduğu, tüm alanların ve tüm metotların yetkilendirme sistemi içinde bulunduğu ve gerektiği taktirde hiyerarşik bir yetkilendirme sistemin oluşturulduğu teyit edilmelidir. Yetki dışı erişim denemeleri mutlaka kayıt altına alınmalı ve incelenmelidir. Yüksek yetkiye sahip rollerin eklenmesi, silinmesi mutlaka birden fazla seviyede kontrolü içermelidir. Rol yönetiminde rol yetkileri dinamik olarak düzenlenebilmelidir[82].

3.2.1.7 Hata Yönetimi

Uygulamada meydan gelen hatalarda istemcinin karşısına çıkan hata sayfasının uygulamaya dair hassas yapılandırma ayarlarını ifşa etmediğinden emin olunmalıdır. Özel tasarlanmış (custom) hata sayfalarının bulunduğu teyit edilmelidir. Error yönetimiyle ilgili kod gözden geçirme çalışmalarında kontrol edilecek hususlar, kullanılan web sunucusuna göre farklılıklar gösterebilmektedir[83].

[81] Conklin Larry, Robinson Gary, *Code Review Guide*, OWASP, 2007, s.151.
[82] a.g.e., s.135.
[83] C. Larry, 2007, s. 157.

3.2.1.8 Dış kaynaklı Yazılım Bileşenlerinin Güvenilirliği

Uygulamalarda çoğu zaman kullanılan Library, Framework gibi bileşenlerin ve bunlarla ve beraber çalışan "package" bileşenlerinin resmi kaynaklar tarafından doğrulandığından ve güvenilir olduğundan emin olunmalıdır. Gereksiz bileşen kullanımları tespit edilmelidir.

3.2.1.9 Yapılandırma Ayarları

Uygulamanın güvenlik yapılandırmasının ve yönlendirme yapılandırmasının doğru şekilde yapılıp yapılmadığı test edilmelidir. Yapılandırma dosyalarının dışarıdan erişimine engel olunduğundan kesin olarak emin olunmalıdır. Kod gözden geçirme faaliyetlerinde yapılandırma ayarlarında kontrol edilecek hususlar programlama diline ve kullanılan bileşenlere (framework, library) göre farklılık gösterebilmektedir[84].

3.2.2 Kod Gözden Geçirme Çalışmalarının Artı ve Eksileri

Kod gözden geçirme çalışmaları genel olarak; uygulamanın daha kaliteli bir kaynak koduna kavuşturmakta, güvenlik açıklarını ortadan kaldırmakta, iki ekip arasında bilgi transferi sağlamaktadır.

Code Review olarak da anılan bu çalışmalarda dezavantaj olarak görülebilecek hususların başında ise, uygulamanın kritik kaynak kodlarının farklı bir ekip tarafından görülmesi sebebiyle ortaya çıkabilecek riskler gelmektedir. Dolayısıyla çalışılan ekibin güvenilirliği mutlaka sorgulanmalıdır.

[84] a.g.e, s. 84-86.

Bunun yanı sıra, Kod gözden geçirme çalışmaları kaynak kodu satır sayısına da bağlı olarak çok uzun bir zaman alabilmekte ve uygulamanın yayına girmesini geciktirebilmektedir. Uygulamalarında bakım aşamasında da bakımına bu çalışma dahil edildiğinde bakım süresi uzamaktadır ve maliyet artmaktadır.

Kod gözden geçirme çalışmaları birden fazla aşamadan oluşmalı, ilk aşamada tespit edilen hata ve açıkların giderilmesi için geliştirici ekibe bir süre verilmeli ve bu süre sonunda yeniden inceleme faaliyeti yapılarak eksiklerin giderilip giderilmediği kontrol edilmelidir. Gerekli durumları tekrar süre verip üçüncü kez kontrol yapılmalıdır[85].

3.3 SIZMA (PENETRASYON) TESTİ

Sızma veya penetrasyon testi, sistemleri ve uygulamaları daha güvenli hale getirmek için yapılan yasal bir saldırı simülasyonu işlemidir. Web uygulamaları söz konusu olduğunda sızma testleri de bahsettiğimiz diğer güvenlik analiz yöntemleri gibi önemli bir yer tutar. Uygulamanın kodları herhangi bir analiz ekibiyle paylaşılmadan, bir uzak bilgisayardan ilgili web uygulamasına saldırı simülasyonları yapılabilir.

Penetrasyon testi aşağıdaki isimlerle de anılır.[86]

- Pentest,
- PT,
- Hacking,
- Ethical Hacking (Etik Hacker),

[85] Code Review Best Practices, https.//medium.com/palantir/code-review-best-practices-19e02780015f (12.03.2021)
[86] Engebretson, Patrick, *The Basics of Hacking and Penetration Testing*, SYNGRESS, ABD, 2011, s. 1.

- White Hat Hacker (Beya Şapkalı Hacker)

Bilgi güvenliği ile ilgili güvenlik açığı tespit/analiz yöntemleri iki türe ayrılmıştır. Bunlar ofansif güvenlik (offensive security) ve defansif güvenlik (defensive security) olarak isimlendirilir. Daha önce bahsettiğimiz statik kod analizi ve kod gözden geçirme işlemleri defansif güvenliğin bir parçasıydı. Sızma testi ise ofansif güvenliği bir parçası olup diğer yöntemlerden ayrılır.

Sızma testi, web uygulama güvenliği barındıran bir müşterinin istediği üzerine ilgili uygulamaya mümkün olan her yöntem denenerek sızma gerçekleştirilmeye çalışılması işlemlerine verilen genel isimdir.

Sızma testlerinde amaç sadece uygulamadaki bir zafiyeti bulmak değildir. Aynı zamanda sızma testleri, bu zafiyetin gerçekten kötü amaçlı kullanıldığında sisteme bir zarar verilebilip verilemediğine dair bir bulgu verir. Yani sızma testleri web uygulamasına yapılacak gerçek bir saldırının simülasyonudur ve sızma testi işlemleri çeşitli senaryolar kullanılarak gerçekleştirilir.

Sızma Testi kavramını ve amaçlarını açıklamak için birkaç maddeden bahsetmek faydalı olacaktır:

- Sızma testinde önceden bilinmeyen güvenlik açıkları tespit edilir ve böyle uygulamanın daha güvenli hale gelmesi sağlanır, ayrıca geliştirici ekibin deneyimine de büyük katkı sunar.
- Sızma testleri tamamen yasal bir çalışmadır.
- Web uygulama işleticileri, özellikle büyük kuruluşlar düzenli olarak uygulamalarına sızma testi yapılması için hizmet satın alırlar.
- Sızma testinde test edilen sadece uygulama değil, onun bulunduğu ortam ve çevresel bileşenlerdir. Bu durum da çalışmanın sonucunda, uygulamanın iç

etmenlerden kaynaklanın zafiyetleri gibi dış etmenlerden kaynaklanan zafiyetleri hakkında da bilgi verir.

- Sızma testi uygulamanın zayıflıklarını tanımladığı gibi güçlü yanlarını da rapor olarak sunar.
- Sızma testi sonucu, ilgili web uygulamasını geliştiren ekip ve teknik personelin bilgi durumu ile önemli bilgiler verir.
- Web uygulamasını işleten kuruluşun gelecekte alacağı kararların daha sağlıklı olması için sızma testleri büyük önem taşımaktadır.

Çeşitli bilişim firmalarından alınabilecek sızma testi hizmetleri bazı bileşenlerden oluşur. Bu bileşenlerin web uygulama güvenliği ile ilgili kısımları uygulanırken de web uygulamasında bazı kriterler üzerinden sızma girişimleri yapılır. Bir kuruluşa ait web uygulamalarına sızma testi gerçekleştirilirken, firmanın internet üzerinden hizmet veren servisleri (Mail, DNS, Web, FTP gibi) kullanılarak web uygulamasına sızma işlemleri gerçekleştirilir[87].

3.3.1 Sızma Testi Türleri

Sızma testi, izlenilen senaryoya göre 3 farklı şekilde yapılır. Bunlar Black-Box Testing, Grey-Box Testing, White-Box Testing olarak isimlendirilir. Bu senaryolar test ekibinin test edilecek web uygulaması hakkında önceden sahip olduğu bilgiye göre adlandırılmıştır.

3.3.1.1 Siyah Kutu

Black Box Testing olarak bilinen test yöntemi, testi yapan kişinin uygulama hakkında ortalama bilgiye sahip bir saldırgan olduğu senaryodur. Bu senaryoda saldırgana sızma testi öncesinde web uygulaması ve sunucu hakkında herhangi bir

[87] Sızma Testleri (Penetrasyon Testleri), https.//www.bgasecurity.com/danismanlik-hizmetleri/penetrasyon-testi-sizma-testi (11.03.2021)

bilgi verilmez.[88] Saldırgan sistem hakkında hiçbir bilgisi olmadan saldırı yapan bir saldırganı simüle eder.

3.3.1.2 Beyaz Kutu

White Box Testing olarak adlandırılan sızma testi, saldırgana web uygulaması hakkında işine yarayacak tüm bilgilerin verildiği test yöntemidir. Bu test yönteminde saldırganın başarı şansı en yüksek seviyededir. Beyaz kutu testinde ilgili web uygulama hakkında tüm bilgilere sahip bir saldırganın uygulamayı hedef aldığı durumlarda ortaya çıkacak hasar simüle edilir.

3.3.1.3 Gri Kutu

Gray Box Testing olarak adlandırılan sızma testinde, saldırgana uygulama hakkında sınırlı bilgi verilir. Saldırganın sızma şansı siyah kutu testine göre daha fazladır. Gri kutu testi çok tercih edilen bir yöntemdir. Bunun birkaç sebebi vardır [89]:

- Hem siyah kutu testinin hem de beyaz kutu testinin faydalarını içerir.
- Uygulamayı geliştiren ekibin ve test ekibinin girdilerini birleştirir ve uygulamanın genel kalitesini iyileştirir.
- Uygulama geliştiricilerin güvenlik açıklarını kapatması için yeterli zamanı tanır.
- Test süresini siyah kutuya kıyasla kısaltır.
- Test işlemi, geliştirici gözlüğünden ziyade kullanıcı gözlüğüyle yapılır.

[88] What are Black Box, Grey Box, and White Box Penetration Testing, Erişim Tarihi: 11.03.2021, https.//resources.infosecinstitute.com/topic/what-are-black-box-grey-box-and-white-box-penetration-testing (11.03.2021)
[89] *What is Grey Box Testing? Techniques, Example,* https.//www.guru99.com/grey-box-testing.html (11.03.2021)

3.3.2 Sızma Testinin İçermesi Gereken Metotlar

Sızma testlerinde kullanılan metotlar ve uygulanan testler genellikle OWASP ve WASC tarafından belirlenen standartlara göre belirlenmektedir. Bu saldırı simülasyon metotlarını başlıklar halinde sıralayacağız[90].

3.3.2.1 Yapılandırma Güvenliği Testleri

Yapılandırma güvenliği testleri genel olarak taşıma katmanı güvenliği testlerinden oluşur. Bu testlerde uygulamanın yapılandırma ayarlarından kaynaklanın bir açıklık varsa bundan yararlanılarak uygulamaya zarar verme denemesi yapılır.

3.3.2.2 Kullanıcı Tanımlama ve Kimlik Doğrulama Testleri

Web uygulamasının kullanıcı yönetimiyle ilgili zaaflardan faydalanılarak yapılan sızma testidir.

Kimlik Doğrulama testler aşağıdaki yöntemlerden meydana gelir:

- Kullanıcı yönetim sisteminin genel olarak tanımlanması ve güvenlik/sızma analizi,
- Kullanıcı parolalarının kalitesi ve parola kırmaya yönelik Brute-Force saldırısına dayanıklılık testi,
- Hassas verilerin taşınma, saklanma ve dağıtılma güvenliği üzerinde yapılan testler,

[90] BEAM Teknoloji (2019), Güvenlik Analizi Raporu Örneği, *Web Güvenlik Analizi Sunumu*, BEAM Teknoloji, Ankara

3.3.2.3 Yetkilendirme Sistemi

Web uygulamasındaki yetkilendirme sisteminin zafiyetleri aranır ve bulunduğunda bu zafiyetler ile sızma gerçekleştirilir.

Yetkilendirme sistemi testleri aşağıdaki alt bileşenlerden oluşur:

- Yetki yönetimi bileşenin genel olarak tanımlanması ve güvenlik/sızma testi,
- Farklı hesaplarla yatay yetki yükseltme testleri,
- Dikey yetki yükseltme testleri,
- Doğrudan nesne erişimi testi,
- Parametre manipülasyonuyla yetki dışı işlem gerçekleştirme testi

3.3.2.4 Oturum Yönetimi Testleri

Web uygulamasının oturum (session) yönetim bileşenlerindeki zafiyetler aranır ve bir zafiyet tespiti sonrası sızma denemeleri yapılır.

Oturum Yönetimiyle ilgili testler aşağıdaki gibidir[91]:

- Bileşenin genel olarak tanımlanması ve güvenlik/sızma analizi,
- Oturum sabitlenmesi testi,
- Sunucu veya İstemci tarafından oturumun sonlandırıldığı durumda, oturum sonlandırma güvenliği testi,

[91] BEAM Teknoloji (2019), Güvenlik Analizi Raporu Örneği, *Web Güvenlik Analizi Sunumu*, BEAM Teknoloji, Ankara

3.3.2.5 Girdi Doğrulama ve Enjeksiyon Testleri

Web uygulamasındaki input alanları ve bu alanlardan dönen verilerin işlenmesi ile ilgili bileşenlerde bulunan güvenlik açıkları ile ilgili sızma denemelerini içeren penetrason testleridir.

İlgili test yöntemleri aşağıdaki gibidir:

- Girdi(input) doğrulama mekanizmasının tanımlanması ve geçilmezlik testi,
- SQL enjeksiyon testi,
- XSS, HTML enjeksiyon testleri,
- OS komut enjeksiyon testleri
- Dizin Gezinimi testi,
- Open Redirect (Zararlı siteye yönlendirme) testi,
- Dosya uzantı ve dosya boyut kontrolü testleri

3.3.2.6 Hata ve İstisna Yönetimi Testleri

Web uygulamasında bazen istemci kaynaklı ve bazen de uygulama hatalardan kaynaklı olarak hatalar meydana gelebilmektedir. Bunlarla baş edebilmek için web uygulamalarında genellikle bir hata yönetim sistemi vardır. Bu sistemin ne derece güvenli olduğunu test etmek için çeşitli penetrasyon yöntemleri kullanılmaktadır. Bu testlere genel olarak "Hata ve İstisnai durumlar sonucu bilgi sızıntı testi" denilmektedir.

3.3.2.7 İstemci Taraflı Testler

Web uygulamasına istemci tarafına dair zafiyetlerinin test edildiği penetrasyon işlemleridir. İlgili penetrasyon yöntemleri aşağıdaki gibidir:

- CSRF: Cross-site Request Forgery (Çarpaz sayfadan gelen sahte istek) testi
- Güvenlik başlıklarıyla ilgili testler
- Cache-control güvenlik bağlığı testi
- X-Xss-Protection güvenlik başlığı testi
- X-Frame-Option güvenlik başlığı testi
- HSTS güvenlik başlığı testi
- Cross-origin Resource Sharing (CORS) Kökler Arası Kaynak Paylaşımı testi,
- Yerel depolama güvenliği testleri,

3.3.2 Sızma Testinde Risk Derecelendirme

Web uygulamalarına yönelik sızma testlerine genellikle ortaya çıkan bulgular, önem ve önceliklerine göre sınıflara ayrılmaktadır. Bu derecelendirme sınıfları maddeler halinde sıralanacaktır[92]:

3.3.2.1 Acil Bulgu

Penetrasyon testi sonucu ortaya çıkan bulgulardan "acil" olarak isimlendirilenler uygulamanın ilgili bulgu üzerinden bir saldırıya maruz kalması durumunda tamamen ele geçirebileceğini, uygulamanın ve kullanıcı verilerinin çok ciddi bir güvenlik riski altında olduğunu gösterir. Acil olarak değerlendirilen bulgular uygulamanın internete açık bulunan kısmında ve tüm istemcilerin ulaşabileceği yerdedir.

Örneğin uzaktan dosya yükleme (file upload) bileşenin sahip bir web uygulamasında eğer dosya uzantıları filtrelenmeden kabul ediliyor ve web üzerinden çalıştırılabilir (.aspx, .jar gibi) dosyalar yüklenebiliyorsa ve bu internete açık şekilde tüm istemciler tarafından yapılabiliyorsa, bu durum Acilen çözülmesi gereken bir

[92] BEAM Teknoloji (2019), Güvenlik Analizi Raporu Örneği, *Web Güvenlik Analizi Sunumu*, BEAM Teknoloji, Ankara

güvenlik açığına işaret eder. Bunun yanı sıra SQL Enjeksiyon açıkları ve Stored XSS gibi açıklar da bu kapsamda değerlendirilir. [93]

3.3.2.2 Kritik Bulgu

Test sonucu ortaya çıkan bulgulardan "kritik" olarak isimlendirilenler uygulamanın kısmen ele geçirilebileceğini ve ciddi bir risk bulunduğunu gösterir. Genellikle acil seviyesindeki bulgularla aynı zafiyetler için kullanılmakla beraber, kritik seviyedeki bulgular bazı durumlarda acil bulgularla aynı bulgu olmakla birlikte, ilgili bulgu uygulamanın internete açık olmayan ve sadece yönetici girişi sonucu ulaşılabilecek bölümlerde bulunmasıyla acil bulgulardan ayrışır.

Örneğin acil bulgu örneğindeki gibi bir dosya yükleme bileşenine sahip web uygulamasına, uzaktan çalıştırılabilir dosya yüklenebiliyorsa, ancak bu yükleme işlemi ancak yönetici girişinden sonra gerçekleşebiliyorsa, bu durum kritik seviyede bir güvenlik açığına işaret eder. Bunun yanı sıra Reflected XSS gibi zafiyetler de bu gruba dahildir. [94]

3.3.2.3 Yüksek Riskli Bulgu

"Yüksek risk" olarak adlandırılan bulgular uygulama dosyalarının bir saldırı sonucu ifşa olabileceğini belirtir. Bu kategoride risk halen ciddi seviyededir. Örneğin bir dosya yöneticisi (file manager) aracı kullanan web uygulamalarında, dosya yöneticisine yetkisiz erişim mümkünse, bu durum yüksek riskli bulgu olarak değerlendirilebilir. Eğer yetkisiz istemci dosya yöneticisi aracılığıyla dosya ekleme-

[93] Pci Scanning Procedures - Pci Security Standarts Council, https://www.pcisecuritystandards.org (10.03.2021)
[94] Reporting (Penetration Test), http://www.pentest-standard.org/index.php/Reporting (12.03.2021)

silme gibi işlemleri de gerçekleştirebiliyorsa bu durumda güvenlik açığı acil veya kritik seviyesinde değerlendirilecektir.

3.3.2.4 Orta Riskli Bulgu

Uygulama dosyalarının saldırı sonucu ifşa olabileceğini gösteren bulgulardan bir kısmı "orta riskli" olarak değerlendirilmektedir. Örnek olarak güvelik yapılandırma ayarlarındaki zafiyetler veya dosya yöneticisine yetkisiz erişimlerin sadece kullanıcı girişinden sonra gerçekleştiği zafiyetler orta riskli bulgu kapsamında değerlendirilebilir.

3.3.2.5 Düşük Riskli Bulgu

Bir saldırı durumunda uygulamanın yapısı ve çalışma şekliyle ilgili bazı bilgilerin ele geçirilebileceği durum "düşük riskli" olarak adlandırılır. Örnek olarak şifrelerin hash işleminden geçirilmeden veri tabanında tutulması kaba kuvvet şifre kırma yöntemlerine karşı bir önlemin bulunmaması veya uygulamanın web tarayıcı üzerinde aktifleştirdiği bazı güvenlik başlıklarının eksikliği düşük riskli bulgu kapsamında değerlendirilir.

3.3.2.6 Bilgi

Bir güvenlik açığı tespit edildiğinde fakat bu açık bir risk oluşturmadığında bu bulgu, bir "bilgi" olarak isimlendirilip analiz raporunda bu şekilde ilgili kuruluşa veya uygulama yöneticisine sunulur. Bunun yanı sıra web uygulamasının çalışma mantığında görülen bazı yanlışlar bu kapsamda değerlendirilebilir.

3.3.3 Sızma Testinde Değerlendirme Raporu

Penetrasyon testi gerçekleştirildikten sonra sunulan rapordaki değerlendirme bölümlerden ile olarak genel değerlendirme bölümü yer alır ve bu bölümde uygulamanın güvenlik durumuyla ilgili genel durum ilgili kuruluşa sunulur. Bu değerlendirmede öncelikle web uygulamasının kullandığı teknolojiler ve teknolojilere ait ön tanımlı bilgiler verilir. Uygulamanın harici olarak kullandığı üçüncü parti araçların değerlendirmesi yapılır.

Daha sonra uygulamanın genel yapısı, kullanıcı yönetimi, yetkilendirme yönetimi gibi sistemlerin bulunup bulunmadığı değerlendirilir. Bunun yanında sızma testinde hangi teknolojilerin kullanıldığı ve ne türlü testlerin yürütüldüğü açıklanır.

Genel Değerlendirme bölümünün ardından çoğunlukla Genel Bulgular isimli bir bölüm yer almaktadır. Bu bölümde yapılan tüm penetrasyon test işlemlerine karşı web uygulamasının yüzde kaç başarı sağladığı gösterilir ve bu bilgi üzerinden genel bir değerlendirme yapılır. Zafiyetlerin risk dağılımı ile ilgili grafiklerle risk ne düzeyde olduğu anlatılır.

Bunun ardından tavsiye özeti bölümünde güvenlik açıklıklarından yüksek ve üstü risk içerenler anlatılıp bunlara karşı ne gibi önlemler alınabileceği özet halinde anlatılır.

Ardından gelen teknik bilgiler bölümünde yapılan tüm sızma testleri detaylı olarak sıralanır. Bu bölümde sızma testinde kullanılan araçlar, bir güvenlik açığının bulunduğu durumlarda bulgu detayları, güvenlik açığının risk seviyesi anlatılır ve detaylı analiz yapılır. Detay-Kanıt olarak ilgili güvenlik açığının ekran görüntüleri paylaşılır ve zafiyetin ne olduğu net olarak açıklanır.

Ardından her bulgu üzerine bir çözüm önerileri bölümü bulunur ve bu bölümde zafiyetin ortadan kaldırılması için web uygulamasında, yapılandırma dosyalarında veya kaynak kodlarında ne gibi değişiklikler yapılması gerektiği anlatılır. Bunun sonrasında daha detaylı çalışma yapılabilmesi için ilgili zafiyetle ilgili referans kaynaklar paylaşılır.

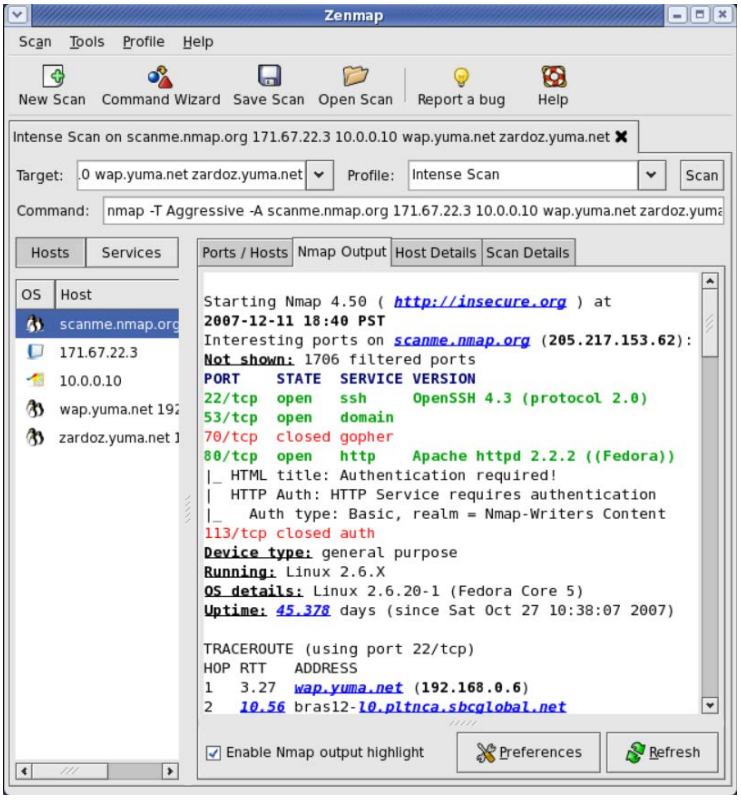
3.3.4 Sızma Testlerinde Kullanılan Araçlar

Penetrasyon testleri bu amaca yönelik olarak programlanmış çeşitle yazılımlar kullanılarak yapılmaktadır. Testlerde çeşitli fonksiyonları yerine getirmek için bu fonksiyonların her biri için geliştirilmiş çok sayıda araç vardır. Bunların içinden bazıları genel kabul görmüş ve öne çıkmış araçlardır[95]. Sızma testlerinde en sık kullanılan 5 araç[96] bu bölümde maddeler halinde işlenecektir.

3.3.4.1 Nmap

Temelde bir port tarama yazılımı olan Network Mapper (Nmap), bu konuda dünya çapında en yaygın kullanılan araçtır[97]. Nmap port tarama özelliğini dışında hedef sistemin veya uygulamanın çeşitli bilgilerini sızma testi gerçekleştiren ekibe sunmaktadır. Hem network, hem sistem, hem de web uygulamaları alanında kullanışlı bir tarama ve keşif yazılımıdır. Web uygulamalarla ilgili hemen her sızma testinde kullanılan bir araçtır. Genellikle Zenmap GUI (Grafiksel Kullanıcı Arayüzü) ile birlikte kullanılmaktadır.

[95] The Top 5 Pentesting Tools You Will Ever Need, https.//resources.infosecinstitute.com/topic/the-top-5-pentesting-tools-you-will-ever-need (12.03.2021)
[96] Guide to Web Application Penetration Testing, https.//relevant.software/blog/penetration-testing-for-web-applications (12.03.2021)
[97] Burlu, Kamil, Bilişimim Karanlık Yüzü, NİRVANA, Ankara 2015, s.43



Şekil 3.2: Zenmap GUI Ekran Görüntüsü [98]

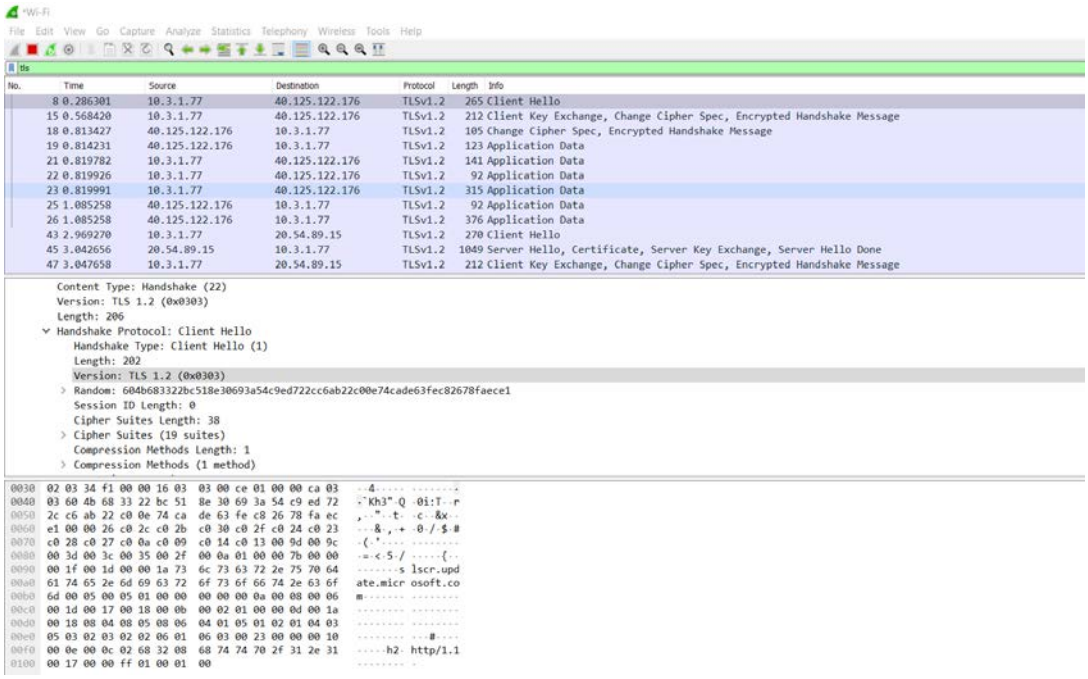
3.3.4.2 WireShark

WireShark ağ protokol analizi amacıyla geliştirilmiş bir programdır. Sızma testlerinde web uygulaması ile ilgili paketlerin güvenliğini test etmek için buna benzer programlara ihtiyaç duyulmaktadır. Bu tür araçların en popülerlerinden birisi olan WireShark, WinPcap isimli bir uygulamayla birlikte gelmektedir. WinPcap anlık olarak ilgili bilgisayarın Ethernet trafiğini yakalayan bir yazılımdır[99].

Web Uygulama Güvenliği'nde ve sızma testlerinde de bu tür yazılımlara sıkça ihtiyaç duyulmakta ve özellikle uygulama güvenliğinin ağ katmanını ilgilendiren

[98] Zenmap GUI, https.//nmap.org/zenmap (13.03.2021)
[99] Wireshark Nedir? Nasıl Kullanılır?, https.//www.karel.com.tr/bilgi/wireshark-nedir-nasil-kullanilir (01.03.2021)

bileşenleri için veri paketlerinin takibinde ve sızma testlerindeki veri takibi, paket incelemesi gibi işlemlerde bu uygulamalar kullanılmaktadır. HTTP protokolünü kullanan web uygulamalarına ait paketler WireShark yazılımı ile yakalanıp içindeki hassas bilgiler görüntülenebilmektedir.

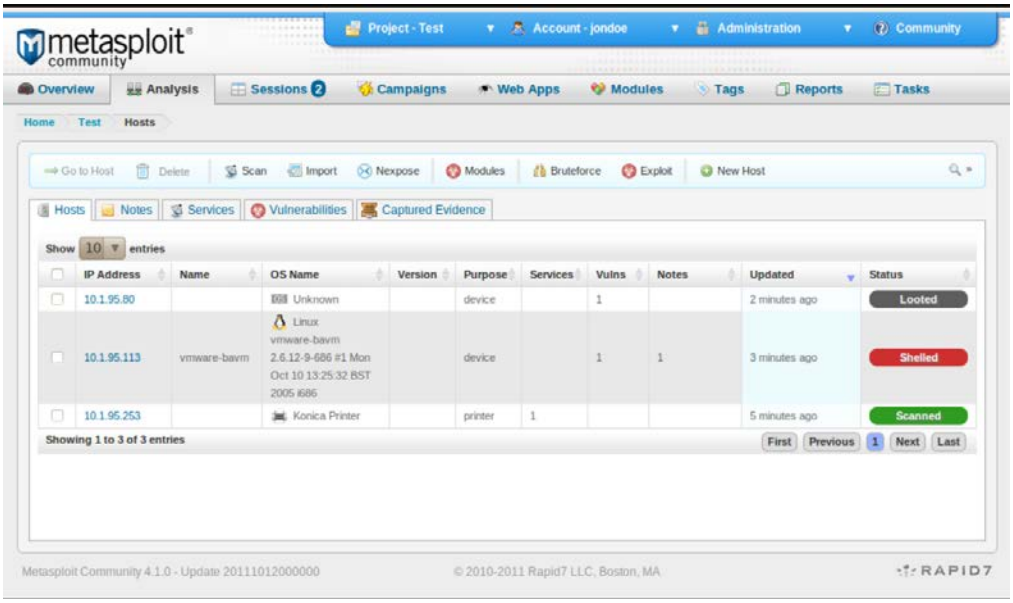


Şekil 3.3: WireShark TLS Trafiği İncelemesi

3.3.4.3 Metasploit

Metasploit tek başına bir ugulamadan ziyader bir Framework olarak görev yapmaktadır. Sızma testlerine duruma göre programlanabilme ve özgün görevleri yerine getirebilmektedir. Metasploit üzerinde hedef alınacak güvenlik açığı seçilir, ilgili zararlı içerik seçilir ve hazırlanacak bir plan üzerinden saldırı gerçekleştirilir. Metasploit içinde birçok hazır exploit (güvenlik açığı istismarı) aracıyla birlikte gelir. "*Güvenlik uzmanları ve saldırganlar tarafından exploit uygulama ve geliştirmede kullanılan bir altyapıdır*"[100].

[100] *B. Kamil*, 2015, s.432

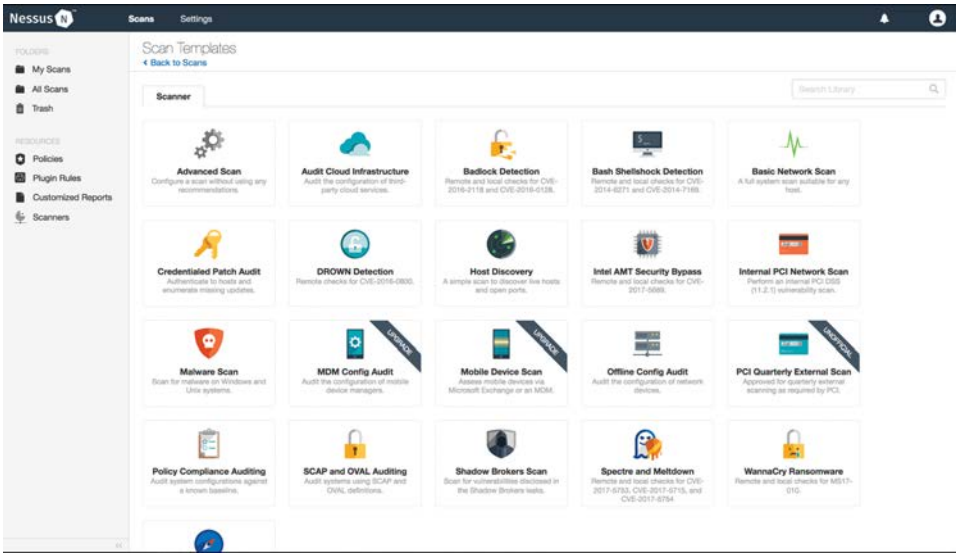


Şekil 3.4: Metasploit Kullanıcı Arayüzü [101]

3.3.4.4 Nessus

Bir zafiyet tarayıcısı olan Nessus, herhangi bir saldırıda bulunmak için değil sadece keşif yapmak için tasarlanmıştır. Penetrasyon test ekibine, web uygulamasının zafiyetlerini, yapılandırma sorunlarını bulmada ve uygulamadaki zararlı yazılımların tespitinde yardımcı olur.

[101] *Metasploit Project*, https.//www.wikiwand.com/en/Metasploit_Project (12.03.2021)

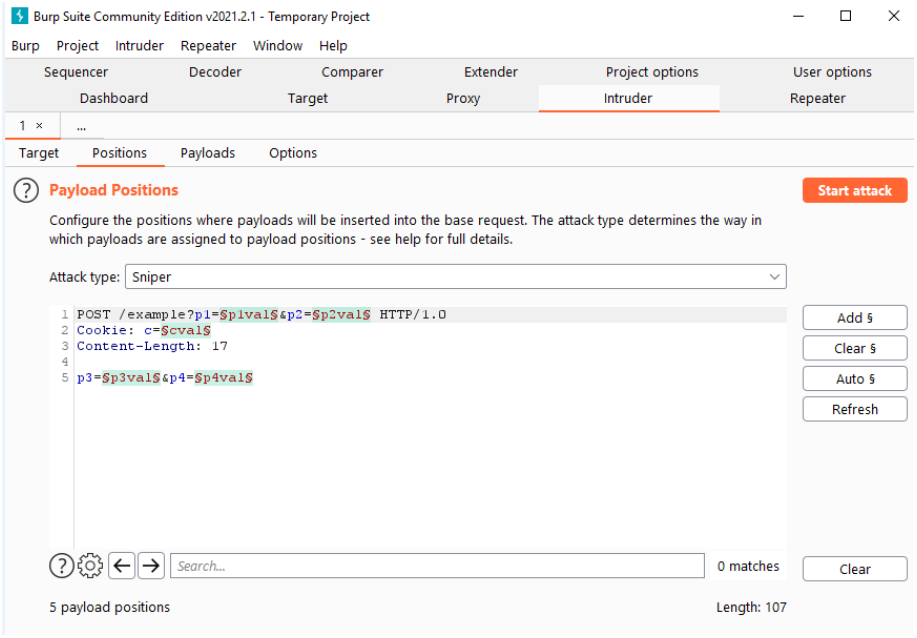


Şekil 3.5: Nessus Kullanıcı Arayüzü [102]

3.3.4.5 Burp Suite

Burp Suite en kapsamlı web uygulama güvenliği programlardan bir tanesidir ve *sızma testleri için kullanılan araçlardan oluşan bir set* olarak tanımlanmaktadır[103]. Sızma testi ekiplerinin işini oldukça kolaylaştıran önemli bir yazılımdır. Burp Suite ile herhangi bir çaba harcamadan yapılabilecek hazır testler olduğu gibi el ile daha detaylı sızma girişimleri yapmak da mümkündür.

[102] The Nessus Famly, https.//www.tenable.com/products/nessus (12.03.2021)
[103] Lozano, Shah Walikar, *Hands-On Application Penetration Testing with Burp Suite*, Packt, 2019, s. 6.



Şekil 3.6: Burp Suite Kullanıcı Arayüzü

Burp Suite yazılım ile bir web uygulamasının, web tarayıcısında aktifleştirdiği güvenlik başlıklarını detaylı şekilde incelenebilmekte, web uygulamasına upload edilen bir dosyanın içeriğini (content) farklı bir içerik olarak gösterip sızma girişiminde bulunulabilmekte ve buna benzer çok sayıda penetrasyon testi gerçekleştirilebilmektedir.

3.3.5 Sızma Testi Artı ve Eksileri

Sızma testleri web uygulamasının kaynak kodlarına erişilmeden, web uygulama güvenliği konusunda uzman bir ekibin ilgili uygulamaya çeşitli yöntemler ve araçlar kullanılarak sızma girişimlerinde bulunmasıdır.

Uygulamanın kaynak koduna erişilmemesi bir dezavantaj olup, kaynak kodunda var olan bazı güvenlik açıklarının sızma testi kapsamında test edilmediği durumda, ilgili açıkların gözden kaçmasına sebep olmaktadır. Dolayısıyla sızma testinin güvenilirlik açısından kod gözden geçirme faaliyetlerinin bir basamak altında

olduğu söylenebilir. Bununla birlikte sızma testleri maliyet ve süre açısından kod gözden geçirme çalışmalarına göre tasarruf sağlamaktadır.

Sadece standart testler yapan hazır araçlar yerine bir uzman ekip tarafından yürütülüyor olması sızma testlerinin güvenilirliğini artıran bir unsur olmaktadır. Hazır araçlar kendi veri tabanlarında bulunan ve önceden belirlenmiş hazır test yöntemleri üzerinden web uygulamada zafiyet ararken, sızma testini yapan uzman ekip uygulama mimarisinin kendine özgü yönlerini değerlendirerek duruma göre proaktif test yöntemleri geliştirebilmektedir. Bununla birlikte uzman ekibin yeterli beceriye sahip olmadığı durumlarda istenen sonucun elde edilememesi sonucu ortaya çıkabilmektedir.[104]

3.4 ZAFİYET TARAMASI

Tez çalışmasında işlenecek son web uygulama zafiyet tespit yöntemlerinden olan zafiyet taraması (vulnerability assesment), web uygulama güvenliği için geliştirilmiş hazır araçlarla yapılan bir sistematik zafiyet tespit yöntemidir. Zafiyet taramalarında, otomatik tarama yöntemleriyle web uygulamalarındaki ve uygulama kaynak kodundaki güvenlik açıkları tespit edilir.

Zafiyet tarama çalışmalarında; zafiyet tanımlama, zafiyet analizi, risk değerlendirmesi, iyileştirme bölümlerinden oluşur ve kullanılan hazır araçlarla, önceden bilinen saldırı yöntemleri kullanılarak web uygulamasına yönelik bir saldırı simülasyonu yapılır.[105]

[104] Pros and cons of penetration testing, https.//www.itgovernance.eu/blog/en/pros-and-cons-of-penetration-testing (15.03.2021)
[105] Vulnerability Assesment, https.//www.imperva.com/learn/application-security/vulnerability-assessment (13.03.2021)

3.4.1 Zafiyet Taramasında Kullanılan Araçlar

Kaynak koduna ulaşmadan yapılan bir web uygulama güvenliği zafiyet analiz yöntemi olan zafiyet taramalarında en sık kullanılan ve en yüksek başarı oranına sahip olan araçlardan, web uygulama güvenliğine odaklanmış olanlar aşağıda sıralanmıştır[106]. Bu araçlar dinamik uygulama güvenlik test araçları (DAST Tools) olarak da bilinmektedir.

- Nikto2
- Netsparker
- OpenVAS
- W3AF
- Arachi
- Acunetix
- OpenSCAP
- GoLismero
- Retina CS Community
- Nexpose

3.4.2 Zafiyet Taraması Artı ve Eksileri

Zafiyet taramalarında kullanılan hazır araçlarla güvenlik testinin yapmak, literatürde var olan, daha önceden keşfedilmiş, üzerinde çalışmalar yapılmış web uygulama güvenlik açıklarının tespitinde etkili bir yöntemdir. Bununla birlikte sıfırıncı gün saldırıları (Zero Day Attack) olarak tanımlanan ve daha önceden literatürden örneği bulunmayan yeni tür saldırıları tanımlamakta ve tespit etmekte yetersiz kalacaktır. İşte bu noktada zafiyet taraması, sızma testinden ayrılmaktadır.

[106] 17 Best Vulnerability Assessment Scanning Tool, https.//phoenixnap.com/blog/vulnerability-assessment-scanning-tools (13.03.2021)

Sızma testleri bu konuda uzmanlaşmış kişi ve firmalarca yapılırken, zafiyet taraması web uygulamasını yöneten kişi veya kurum tarafından da ilgili zafiyet tarama aracı satın alınarak yapılabilmektedir. Dolayısıyla zafiyet taramasında maliyetin daha düşük olması ve daha düşük eforlar yapılması beklenebilir. Ancak zafiyet taraması sonrası ortaya çıkan bulgular ve sonuç raporunu yorumlama ve tespit edilen güvenlik açıklarını giderme noktasında uzman bir firmadan danışmanlık alınması da sıkça tercih edilen bir yöntemdir.

Web uygulamaları içerisinde, uzman bir firma tarafından geliştirilip birden fazla kuruma satılan paket programlar için genellikle zafiyet taraması yapılması yeterli olurken, kurum personeli tarafından geliştirilen veya bir firma tarafından ilgili kuruma özgü olarak geliştirilen web uygulamaları için sızma testinin tercih edilmesi daha doğru olacaktır. [107]

Zafiyet taramasında kullanılan araçların avantajları ve dezavantajları birlikte değerlendirildiğinde, aynı statik kod analizi araçlarında olduğu gibi kurum içinde bulunan personeller tarafından, alınacak eğitimlerin ardından kullanılabileceği ve kullanılması gerektiği ancak hazır araçları kullanma yönteminin tek başına yeterli yazılımlar olmadığı değerlendirilmektedir.

[107] *Sızma Testi mi? Zafiyet Taraması mı?* https.//medium.com/@hemreyilmaz/s%C4%B1zma-testi-mi-zafiyet-taramas%C4%B1-m%C4%B1-8d4b01ed9b25 (13.03.2021)

DÖRDÜNCÜ BÖLÜM

WEB UYGULAMA GÜVENLİĞİ İLE İLGİLİ MEVZUAT

İnternetin, ortaya çıkışından itibaren dünyada meydana getirdiği değişimin en büyük bileşenlerinden birisini web uygulamaları oluşturmaktadır. İlk ortaya çıktıkları dönemden günümüze kadar hızlı bir gelişim gösteren web uygulamaları üzerinde her gün milyarlarca dolarlık işlem yapılmakta, aynı zamanda bu uygulamalar milyarlarca insanın kişisel verilerini işlemektedir.

İnsanlığa sundukları imkanların yanında web uygulamalarının aynı zamanda illegal yöntemlerle çeşitli çıkarlar elde edilmesi noktasında da eşsiz fırsatlar sunduğu açıktır. Bu durum; internet, siber güvenlik, bilgi güvenliği, web uygulama güvenliği, kişisel verilerin güvenliği gibi konularda çeşitli hukuki hükümlerin hazırlanmasını zorunlu kılmıştır.

Günümüzde web uygulamaları üzerinden işlenecek suçlarla ilgili cezai yaptırımlar düzenlenmiştir. Suçu işleyen saldırganlarla ilgili yaptırımların yanında, ilgili web uygulamasının sorumlusunun da, gerekli önlemlerin alınmadığının ortaya çıktığı durumlarda cezai yaptırımla karşılaşmaları söz konusudur. Bu durum, web uygulama güvenliğini sağlamayı hukuki bir zorunluluk haline getirmektedir.

Tez çalışmasının bu bölümünde web uygulama güvenliğini ilgilendiren hukuki düzenlemeler alt başlıklar halinde incelenecektir. Verilen bilgilerin analizi tartışma ve yorum bölümünde yapılacaktır.

4.1 SİBER OLAYLARA MÜDAHALE EKİPLERİNİN KURULUŞ, GÖREV VE ÇALIŞMALARINA DAİR USUL VE ESASLAR HAKKINDA TEBLİĞ

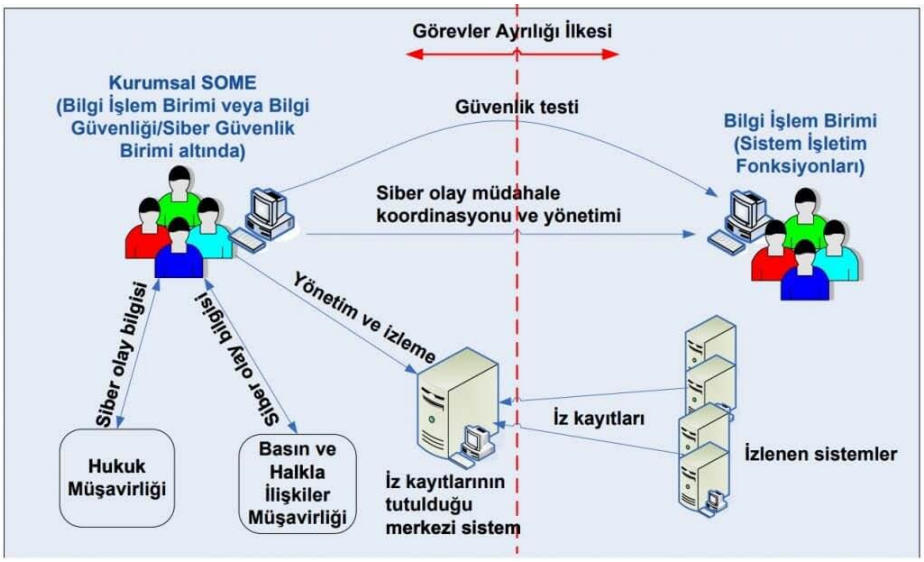
11.11.2013 tarihinde Resmî Gazete'de yayımlanan *Siber Olaylara Müdahale Ekiplerinin Kuruluş, Görev Ve Çalışmalarına Dair Usul Ve Esaslar Hakkında Tebliğ*'in amaç ve kapsamı "*Siber Olaylara Müdahale Ekiplerinin kuruluş, görev ve çalışmalarına ilişkin usul ve esaslarını belirleyerek, hizmetlerin etkin ve verimli bir şekilde yürütülmesini sağlamak"* olarak ifade edilmiştir.

Söz konusu tebliğin 3.Maddesinde tanımlar düzenlenmiştir ve *"Bilişim sistemleri: Bilgi ve iletişim teknolojileri vasıtasıyla sağlanan her türlü hizmet, işlem, veri ve bunların sunumunda yer alan sistemleri ifade eder"* bendinden, web uygulamalarının *"Bilişim sistemleri"* tanımına dahil olduğu anlaşılmaktadır.

4.Madde'de kamu kurumlarında Kurumsal Siber Olaylara Müdahale Ekibi (SOME) kuruluşu düzenlenmiştir. Buna göre Kurumsal SOME'ler Bakanlıkların bünyesinde kurulur. Bunun dışında kalan kamu kurumlarının da kendi SOME'lerini kurabileceği belirtilmiştir.

5.Madde'de Kurumsal SOME'lerin görev ve sorumlulukları belirtilmiştir. Söz konusu maddenin 1.fıkrasında *"Kurumsal SOME'ler kurumlarına doğrudan ya da dolaylı olarak yapılan veya yapılması muhtemel siber saldırılara karşı gerekli önlemleri alma veya aldırma, bu tür olaylara karşı müdahale edebilecek mekanizmayı ve olay kayıt sistemlerini kurma veya kurdurma ve kurumlarının bilgi güvenliğini sağlamaya yönelik çalışmaları yapmak veya yaptırmakla yükümlüdürler"* ve 2.fıkrasında *"Kurumsal SOME'ler, siber olayların önlenmesi veya zararlarının azaltılmasına yönelik olarak, kurumlarının bilişim sistemlerinin kurulması, işletilmesi veya geliştirilmesi ile ilgili çalışmalarda teknik ve idari tedbirler konusunda öneri sunarlar."* denilmiştir. Bu da tez çalışmasının önceki bölümlerinde bahsedilen

yöntemler kullanılarak web uygulama güvenliğini sağlamanın kamu kurumları için hukuki önemini göstermektedir.



Şekil 4.1: Kurumsal SOME'lerin Görevlerini Gösteren Şema [108]

İlgili tebliğin 6. Maddesi'nde Sektörel SOME'lerin kuruluşu düzenlenmiştir. Buna göre Sektörel SOME'ler düzenleyici ve denetleyici kurumların bünyesinde kurulur ve 1. Fıkra'da belirtildiği üzere "*kendi sektörlerinde faaliyet gösteren kurum, kuruluş ve işletmeleri"* kapsar. Sektörel SOME'ler arasındaki eşgüdüm ise BTK tarafından yürütülür.

7.Madde Sektörel SOME'lerin görev ve sorumlulukları ile ilgilidir. 1.Fıkra'daki *"Sektörel SOME'ler, siber olayların önlenmesi veya zararlarının azaltılmasına yönelik faaliyetlerini USOM'la koordineli şekilde yürütürler."* ifadesi web uygulamalara yönelik saldırı henüz yaşanmadan, uygulamadaki zafiyetlerin giderilmesinin önemini göstermektedir.

[108] SOME (Siber Olaylara Müdahale Ekibi) Nedir, https.//www.bgasecurity.com/2018/11/some-siber-olaylara-mudahale-ekibi-nedir (13.03.2021)

5.Madde'nin 6.Fıkrası'nda yer alan *"Kurumsal SOME'ler kurumlarına yapılan siber olayları raporlar ve gecikmeksizin USOM ve birlikte çalıştığı sektörel SOME'ye bildirirler."* ve 7.Madde'nin 2.Fıkrası'nda yer alan *"Sektörel SOME'ler birlikte çalıştıkları SOME'lerde yaşanan siber olayları gecikmeksizin USOM'a bildirirler."* ifadeleri web uygulamalarında loglamanın önemini ve aynı zamanda siber saldırı sonucu uygulamanın tamamen ele geçirilmesi ihtimaline karşı, uygulama içerisinde veri tabanındaki log kayıtları tablosuyla ilgili bir silme veya düzenleme imkanının bulunmaması gerektiğini göstermektedir.

4.2 KİŞİSEL VERİLERİN KORUNMASI KANUNU

6698 Kanun Numarası ile 07.04.2016 tarihinde Resmî Gazete'de yayımlanan Kişisel Verilerin Korunması Kanunu'nun amacı 1.Madde'de *"kişisel verilerin işlenmesinde başta özel hayatın gizliliği olmak üzere kişilerin temel hak ve özgürlüklerini korumak ve kişisel verileri işleyen gerçek ve tüzel kişilerin yükümlülükleri ile uyacakları usul ve esasları düzenlemektir."* şeklinde ifade edilmiştir.

İlgili Kanun'un 3.Maddesi'nde kişisel veri kavramı *"Kimliği belirli veya belirlenebilir gerçek kişiye ilişkin her türlü bilgi"* olarak, Kişisel Verilerin İşlenmesi kavramı *"Kişisel verilerin tamamen veya kısmen otomatik olan ya da herhangi bir veri kayıt sisteminin parçası olmak kaydıyla otomatik olmayan yollarla elde edilmesi, kaydedilmesi, depolanması, muhafaza edilmesi, değiştirilmesi, yeniden düzenlenmesi, açıklanması, aktarılması, devralınması, elde edilebilir hâle getirilmesi, sınıflandırılması ya da kullanılmasının engellenmesi gibi veriler üzerinde gerçekleştirilen her türlü işlem" olarak,* veri sorumlusu kavramı *"Kişisel verilerin işleme amaçlarını ve vasıtalarını belirleyen, veri kayıt sisteminin kurulmasından ve yönetilmesinden sorumlu olan gerçek veya tüzel kişi"* olarak, Veri İşleyen kavramı ise *"Veri sorumlusunun verdiği yetkiye dayanarak onun adına kişisel verileri işleyen gerçek veya tüzel kişi"* olarak tanımlanmıştır.

İlgili kanunun 12. Maddesi'nde "Veri Güvenliğine İlişkin" yükümlülükler düzenlenmiş, (b) bendinde Veri sorumlusunun yükümlülüklerinden birisi *"Kişisel verilere hukuka aykırı olarak erişilmesini önlemek"* olarak ifade edilmiştir. Bu ifade siber saldırı sonucunda web uygulamasındaki bir kişisel veriye erişilmesi durumunda veri sorumlusunun hukuken sorumlu olduğunu göstermektedir.

İlgili kanunun "kabahatler" ile ilgili 18. Maddesi'nin (b) bendinde ise *"12 nci maddesinde öngörülen veri güvenliğine ilişkin yükümlülükleri yerine getirmeyenler hakkında 15.000 Türk lirasından 1.000.000 Türk lirasına kadar para cezası uygulanır."* ifadesi bulunmaktadır. Bu ifade de siber saldırı sonucunda web uygulamasındaki bir kişisel veriye erişilmesi durumunda veri sorumlusunun karşılaşabileceği cezai yaptırımı gösterir.

BULGULAR

Tez çalışmasının ikinci bölümünde web uygulamalarındaki güvenlik açığı türleri incelemiş ve ardından ilgili güvenlik açıklarına yönelik literatürde önerilen yöntemlerin etkinliği, tez çalışmasına özgü olarak yazılan web tabanlı uygulamalar üzerinden test edilerek veri toplanmıştır.

Test edilen zafiyetler, ilgili zafiyetin CWE (Common Weakness Enumeration) kodu, zafiyetin literatürde genel kabul görmüş ismi, zafiyetle ilgili test edilen önleme yönteminin literatürdeki ismi ve ilgili zafiyet üzerinde yapılan testin sonucu olarak önleme yönteminin zafiyeti ortadan kaldırmadaki etkinliği aşağıdaki tabloda belirtilmiştir.

Tablo 1: Zafiyet Önlem Etkinliği Bulguları

Kodu	**Zafiyet Adı**	**Önlem Adı**	**Etkinliği**
CWE-89	SQL Enjeksiyonu	EF Kullanımı	Başarılı
CWE-311	Hassas Bilgi Kriptolama Eksikliği	TLS Kriptolama	Başarılı
CWE-209	Hassas Bilgi İçeren Hata Mesajı Gösterimi	Özel Hata Sayfası Oluşturma	Başarılı
CWE-79	Siteler Arası Betik Çalıştırma	Input Doğrulaması	Başarılı
CWE-434	Tehlikeli Türde Dosya Yüklenmesi ve Çalıştırılması	Dosya Uzantısı ve Content Kontrolü	Başarılı
CWE-352	Siteler Arası İstek Sahteciliği	Token Validasyonu	Başarılı

Zafiyetlerin önlenmesinde, her zafiyet için OWASP'ın önerdiği yöntemlerden birisi denenmiş ve yazılan web uygulamalar üzerinden etkinliği ölçülmüştür. 6 adet zafiyet üzerinde yapılan testlerin tamamı başarılı sonuçlanmıştır ve önerilen önleme yöntemi etkin bulunmuştur. Bu yöntemlerin etkinliğine bulgular, web uygulama

güvenliğinde ve zafiyetlerle mücadele literatürde önerilen ve denenmiş yöntemleri kullanmanın önemini göstermektedir. Tablo 1'de özetlenen bu çalışmanın detaylı yorumlanması tartışma ve yorum bölümünde yapılmıştır.

Tez çalışmasının üçüncü bölümünde web uygulamalardaki güvenlik açıklarının tespiti için yapılan zafiyet tespit çalışmaları incelenmiş, bu çalışmaların içerikleri, yapılma yöntemleri ve etkinlikleri hakkında literatür taraması yapılmıştır.

Zafiyet tespit çalışmalarının özellikleri, içerikleri, yöntemleri ve sahip oldukları etkinliğe dair bulgular aşağıdaki tabloda belirtilmiştir.

Tablo 2: Zafiyet Tespit Çalışmaları Karşılaştırması

	Statik Kod Analizi	***Kod Gözden Geçirme***	***Sızma Testi***	***Zafiyet Taraması***
Uzman Ekip Gerekliliği	Hayır	Evet	Evet	Hayır
Kaynak Koduna Ulaşım	Evet	Evet	Hayır	Hayır
Hazır Yazılım Araçları Kullanımı	Evet	İsteğe Bağlı	Evet	Evet
Çalışmanın Sonuçlanma Süresi	Kısa	Uzun	Orta	Kısa
Maliyet	Düşük	Yüksek	Orta	Düşük
Öz Kaynaklarla Yapılabilme	Evet	Hayır	Hayır	Evet
Yanlış Alarm Bulguları	Yüksek	Düşük	Düşük	Yüksek
Zafiyet Tespit Etkinliği	Orta	Çok Yüksek	Yüksek	Orta
Zafiyetin Çözümüyle İlgili Öneri Sunma	Yüksek	Çok Yüksek	Orta	Orta

Zafiyet tespit çalışması olarak 4 tür çalışma incelenmiş ve çeşitli karşılaştırma yöntemleriyle kıyaslanarak yukarıdaki veriler elde edilmiştir. Bu çalışma 4 adet zafiyet analiz çalışmasının kıyaslanmasının yapılmasına olanak sağlamış ve üzerinde çalışmanın yapılacağı web uygulamasının türüne ve niteliğine göre hangi çalışmanın daha uygun olacağını değerlendirme fırsatı sunmuştur. Tablo 2'de özetlenen bu çalışmanın detaylı yorumlanması tartışma ve yorum bölümünde yapılmıştır.

TARTIŞMA VE YORUM

Web uygulamaları işleten kişi ve kuruluşlar için web uygulama güvenliği her zaman gündemde olan bir problemdir ve uygulamanın yazılım yaşam döngüsü içerisinde güvenliği sağlamak için sürekli bir çaba gerekmektedir. Tez çalışmasının giriş bölümünde bahsedilen CDNetworks raporunda; 2020 yılında web uygulama saldırılarında, 2019 yılına göre %800 gibi astronomik bir artışın yaşandığı görülmektedir. Bu durum da web uygulama güvenliğinde tehlikenin artış eğiliminde olduğunu göstermektedir.

Web uygulama teorisinde, uygulamanın kalite bileşeni içerisinde güvenliğin önemli bir yer tutması ve web uygulama güvenliğinin göz ardı edildiği veya önlemlerin yetersiz kaldığı durumlarda yaşanacak bir saldırının hangi hasarlara yol açacağı, tez çalışmasının birinci kısmında anlatılmıştır. Başarılı olan web saldırıları, uygulamalarda; veri kaybı, veri hırsızlığı, veri enjeksiyonu ve hizmet reddi olarak tanımlanan hasarlardan biri veya birkaçına sebep olacaktır. Dolayısıyla web uygulama güvenliğinin sağlanması için gerekli çalışmaları yapmak, özellikle uygulamalarında kişisel verileri barındıran kamu kurumları için bir zorunluluktur.

Tez çalışmasının ikinci bölümünde web uygulamalara yönelik; *OWASP, WASC, CWE* gibi web uygulama güvenliği üzerine çalışan kuruluşların listelediği en yaygın ve en tehlikeli 13 saldırı yöntemi araştırılmıştır. Ardından bu saldırı yöntemlerine yönelik, literatürde yer alan ve yine adı geçen kuruluşların önerdiği çözüm önerilerinin etkinliği incelenmiştir.

Bu inceleme için tez çalışmasına özgü web uygulamaları yazılarak, uygulamada ilgili güvenlik açığının bulunduğu durumda uygulamaya sızma gerçekleştirilmiş ve gerçek bir uygulamaya benzeri saldırı durumunda ortaya çıkacak

hasar simüle edilmiştir. Ardından yine tez çalışması için yazılan web uygulamalarında ilgili güvenlik açığı literatürden önerilen yöntemle kapatılmış ve önerilen yöntemin etkinliği incelenmiştir.

Bu aşamada ortaya çıkan sonuç; literatürde önerilen yöntemlerin güvenlik açıklarını kapatmada etkili olduğu ve yazılım geliştiricilerin, web uygulamalarının kaynak kodlarını yazmaları esnasında güvenlik açıklarıyla mücadelede literatürde tavsiye edilen yöntemleri kullanmaları gerektiğidir. Bunun yanında yazılımın yaşam döngüsü içerisinde sonradan tespit edilen güvenlik açıklarının da, yine literatürde önerilen yöntemlerle kapatılmasına dikkat edilmelidir.

Tez çalışmasında yapılan deney ve çalışmalar ışığında, web uygulamalarında bulunması gereken temel güvenlik prensipleri maddeler hâlinde aşağıda belirtilecektir:

- Web uygulamasının TLS/SSL sertifikası ile HTTPS protokolü üzerinden yayınlanması ve buna ek olarak istemciden gelen HTTP isteklerinin de HTTPS'e yönlendirilmesi gerekmektedir.
- Web uygulamasının girdi (input) içeren sayfalarından dönen verilerin bir doğrulama (validasyon) metodu tarafından incelenerek işleme alınması, böylecede Siteler Arası İstek Sahteciliği ve Siteler Arası Betik Çalıştırma gibi zafiyetlere ve saldırılar karşı önlem alınması gerekmektedir.
- Yapılandırma ayarlarına dikkat edilmelidir. Hata yönetimi uygulamanın hassas bilgilerini hata mesajlarında sızdırmayacak şekilde tasarlanmalıdır.
- Kullanıcı girişlerinde belirli bir sayıda yanlış deneme yapıldıktan sonra captcha adı verilen bileşen işleme konmalıdır ve ilgili captcha bileşeninin de nitelikli olmasına dikkat edilmelidir.
- Kullanıcı parolaları güvenilir bir hash algoritmasından geçirilerek kriptolanmalı ve bu şekilde saklanmalıdır. Parolalara belirli bir karmaşıklık düzeyi standartı getirilmelidir.
- Uygulamada istemci tarafından dosya yükleme (upload) işlemi yapılıyorsa, dosyanın uzantısı ve içeriği bir doğrulama işleminden geçirilmelidir.

- Kaynak kodunda kullanılan harici bileşenlerin güvenilir kaynaklar tarafından yayınlandığından ve bir zafiyet içermediğinden emin olunmalıdır.

Bu maddelerin haricinde, farklı web uygulamaları kendi yapıların göre farklı güvenlik açıklarına sahip olabilmektedir. İlgili uygulamanın tüm bileşenleri incelenmeli ve ilgili bileşene özgü zafiyetler hususunda farklı güvenlik prensipleri belirlenmelidir.

Tez çalışmasının dördüncü bölümünde işlendiği üzere, mevzuattan kaynaklanan yükümlülükler sebebiyle, web uygulama güvenliğinde riskin en yüksek seviyede olduğu kişisel verileri işleyen web uygulamalarına sahip kamu kurumlarında yazılım geliştirme üzerine çalışan personeller veya dış kaynaklardan yapılan yazılım geliştirme hizmet alımlarını teknik açıdan denetleyen personeller, web uygulamalarında bulunan ve literatürde kritik seviyede olduğu değerlendirilen güvenlik açıkları ve yine ilgili açıklarla ilgili literatürde önerilen çözüm yöntemleri ile alakalı gerekli eğitimleri almalıdır. Kamu kurumları web uygulama güvenliğine gereken önemi vermeli ve personelin ilgili eğitimleri alması için gereken kaynağı ayırmalıdır.

Teknolojinin ve internetin gelişim süreci içerisinde web uygulamalarının da geliştiği ve değiştiği göz önüne alınarak, ilgili literatürün gelişimi takip edilmeli, yeni ortaya çıkacak web uygulama güvenlik zafiyeti türleri ve önleme faaliyetleri için önerilen yeni yöntemler hakkında yazılım geliştirme ile ilgili teknik personelin bilgi birikimi sürekli güncel tutulmalıdır. Bu konuda alınacak eğitimlerin yanında, web uygulama güvenliği ile ilgili yeniliklerin, yukarıda adı geçen kuruluşlar gibi güvenilir İnternet kaynaklarından ve güncel yazılı kaynaklardan takibi yapılmalıdır.

Tez çalışmasının üçüncü bölümünde, web uygulamalarındaki güvenlik açıklarını tespit etmek için kullanılan dört adet birbirinden farklı tespit hizmeti incelenmiştir. Bunlardan *statik kod analizi* ve *kod gözden geçirme* çalışmaları

uygulamanın kaynak koduna erişilerek yapılan zafiyet analiz çalışmalarıdır. *sızma testi* ve *zafiyet taraması* ise kaynak kodlarına erişmeden yapılan zafiyet analiz çalışmalarıdır.

Kaynak koduna erişilerek yapılan iki çalışma yöntemi arasındaki fark; *statik kod analizinin* bu işi için geliştirilmiş yazılımlar kullanılarak otomatize şekilde yapılması, *kod gözden geçirme* çalışmasının ise bir web uygulama güvenliği uzmanı ekip tarafından kodların bizzat incelenerek yapılmasıdır.

Bu aşamada ortaya çıkan sonuçlardan ilki; *statik kod analizi* yönteminde kullanılan yazılımların çok hızlı bir tarama faaliyeti yaptığı ancak uygulamadaki güvenlik açıklarının tamamını fark etmesinin mümkün olmadığıdır. Bununla birlikte *statik kod analizi* yazılımları çok sayıda yanlış alarm (false positive) üretmekte ve güvenlik taramasının sonuç raporunda bunları bulgu olarak sunmaktadır. Bu sebeple kaynak kod yazılma aşaması bitmiş uygulamaların, *Statik Kod Analizi* araçlarıyla taranması sonrası ortaya çıkan çok sayıda, kritik öneme sahip olmayan bulgunun düzeltilmesi mümkün olmamaktadır. Bu durum da web uygulamalarındaki güvenlik açıklarının tespiti için *statik kod analizi* yönteminin tek başına yeterli olmadığını göstermektedir.

Ortaya çıkan ikinci sonuç; *statik kod analizi* araçlarının, uygulamaların kaynak kodlarının ilk kez yazıldığı aşamada, hem güvenlik zafiyetlerinin tespiti ve önlenmesinde, hem de ortaya çıkardığı çok sayıda bulgunun, güvenlik zafiyeti içermese bile en ideal kod yazım tekniklerini önermesinden dolayı, kaynak kodu kalitesini artırmak için faydalı olduğudur. Buna karşın, kurumların bir dış kaynaktan *statik kod analizi* hizmet alımı yapması bir önceki sonuç maddesinde bahsedilen sebeplerden dolayı verimliliği ve etkinliği düşük bir yöntemdir.

Kaynak koduna erişilerek yapılan çalışmalardan ikincisi olan *kod gözden geçirme* çalışmaları ise, bir uzman ekip tarafından kodun incelenmesi ile yapılan bir çalışma olduğundan dolayı etkinliği yüksek bir web uygulama güvenlik Analizi yöntemidir. Bu çalışmada uzman ekip kodların tamamını veya güvenlik açığı içerme potansiyeli taşıyan kod bloklarının tümünü incelediğinden dolayı, kaynak kodunda yer alan herhangi bir güvenlik açığının "insan hataları" hariç gözden kaçma ihtimali yoktur. Uzman ekipte aynı kod satırlarını birden fazla uzmanın incelemesi durumunda insan hatası ihtimali de en aza inecektir.

Kod gözden geçirme çalışmaları sırasında uzman ekip tarafından *statik kod analizi* yazılımları da çalıştırılıp, incelenecek kaynak kodunun yapısı ve kaynak kodundan kullanılan teknolojiler hakkında fikir edinilebilmektedir. Bunun dışında *statik kod analizi* sonucu ortaya çıkan bulgular, güvenlik açıklarının varlığına dair bilgiler verebildiği için *kod gözden geçirme* ekibinin güvenlik zafiyetlerini daha hızlı bulmasına yardımcı olmaktadır.

Bu aşamada ortaya çıkan sonuçlarda ilki; dış kaynaktan hizmet alımı şeklinde yapılacak çalışmalar söz konusu olduğunda *kod gözden geçirme* çalışmalarının etkinlik ve verim açısından *statik kod analizi* çalışmalarından daha önde olduğudur.

Ortaya çıkan ikinci sonuç ise; *kod gözden geçirme* çalışması yapılırken *statik kod analizi* çalışmasına ait unsurların da çalışmaya dahil edilmesinin güvenlik açığı tarama faaliyetinin kalitesini artıracağıdır. Bu konuda bir hizmet alımının teknik şartnamesi oluşturulurken işin adının *kod gözden geçirme* faaliyeti olarak konulması ve hizmeti veren firmanın çalışmada *statik kod analizi* yazılımlarını da bulunduracağının belirtilmesi en ideal yöntem olacaktır. Bu aşamada, çalışmada kullanılacak *statik kod analizi* yazılımının hangisi veya hangileri olabileceği, kurum personeli tarafından yapılacak bir araştırmayla belirlenmeli ve şartnamenin ilgili maddesine seçilebilecek yazılımların isim listesi eklenmelidir. Burada dikkat edilmesi gereken husus, kullanılacak *statik kod analizi* yazılımının veya yazılımlarının, kurum

tarafından güvenlik testine girmesi karar verilen web uygulamalarının programlama dilleri ve kullandıkları teknolojik bileşenleri desteklemesidir.

Kod gözden geçirme çalışması ile ilgili ortaya çıkan üçüncü sonuç, bu çalışmalar sırasında uzman ekibin, analiz edilecek kaynak kodunun geliştiricisi olan ekiple birlikte çalışma gerekliliğidir. Harcanan zaman ve emek bakımından güvenlik analiz yöntemleri içerinde halihazırda en kapsamlısı olan *kod gözden geçirme* çalışması, uzman ekip ile kaynak kodunu geliştiren ekip arasında bir iletişim olmadığı durumda daha uzun zaman ve daha fazla emek gerektirecektir. Yazılım geliştirici ekip, ilgili uygulamanın mimarisini ve kaynak kodların yapısını güvenlik analizi yapan ekibe anlatırsa, iki ekibin birlikte çalışmasıyla *kod gözden geçirme* faaliyeti daha verimli ve etkin şekilde gerçekleşecek, zaman ve maliyetten tasarruf sağlanabilecektir. Güvenlik analizinden ortaya çıkan bulguların iki ekip tarafından birlikte değerlendirilmesi, hangi bulgunun gerçekten bir zafiyeti işaret ettiğini, hangi bulgunun ise bir yanlış alarm niteliğinde olduğunu ortaya çıkarmayı kolaylaştıracaktır.

Kaynak koduna erişilerek yapılan çalışmalarla ilgili ortaya çıkan bir diğer sonuç ise, kritik öneme sahip web uygulamalarının kaynak kodlarının kurum dışı bir kaynak tarafından incelenmesinin çeşitli riskler doğuracağıdır. Bir web uygulamasının kaynak kodunda gizlilik derecesi yüksek bilgiler bulunuyorsa, bu uygulamanın güvenlik analizi için kaynak koduna erişilerek yapılan çalışmalarda ilgili uzman ekibe kaynak kodlar teslim edilirken veya kaynak kodu üzerinde yazılım geliştirme ekibi ve güvenlik uzmanı ekip ortak bir çalışma yürütüyorken, yüksek gizlilik derecesine sahip ilgili kod blokları kaynak koddan çıkartılmalıdır. Eğer bu mümkün değilse, ilgili uygulamaların güvenlik analizleri için kaynak koduna erişilerek yapılan çalışmalar tercih edilmemelidir.

Kaynak koduna erişilmeden yapılan güvenlik analiz çalışmalarından *zafiyet tarama* çalışmaları, *statik kod analizi* çalışmasının mantığına benzer şekilde bu amaç için tasarlanmış yazılımlarla yapılmaktadır. *Zafiyet tarama* çalışması, hızlı sonuç

veren maliyet düşük bir yöntem olmakla birlikte *statik kod analizinde* olduğu gibi çok sayıda "yanlış alarm" niteliğinde bulguların ortaya çıkmasına sebep olmaktadır ve ilgili web uygulamasındaki tüm zafiyetleri tespit etme ihtimali düşük olan bir yöntemdir. *Zafiyet tarama* çalışmasını *statik kod analizi* çalışmasından ayıran özellik; kaynak koduna erişim yapılmadan, kullanılan yazılımın ağ üzerinden ilgili web uygulamasına otomatize saldırı girişimlerinde bulunmasıdır. Bunun yanında, iki yöntemde birbirinden farklı yazılımlar kullanılmaktadır.

Tez çalışmasının ilgili maddesinde bahsedildiği gibi *zafiyet taraması* çalışmaları hazır bir yazılımdaki standart yöntemler üzerinden tarama yaptığı için özellikle *sıfırıncı gün saldırıları (zero day attack)* olarak tanımlanan ve daha önceden literatürde örneği bulunmayan saldırıları tanımlamakta ve tespit etmekte yetersiz kalmaktadırlar.

Kaynak koduna erişilmeden yapılan güvenlik analiz çalışmalarından bir diğeri ise *sızma testi* adı verilen çalışmadır. *Sızma testi*, web uygulama güvenliği konusunda uzman bir ekip tarafından, web uygulamalarına gerçekleştirilen bir dizi saldırı simülasyonudur. Bu bakımdan *kod gözden geçirme* çalışmalarına benzemekle birlikte, analiz ekibinin kaynak koduna ulaşımı olmadığı için *kod gözden geçirme* çalışmasından ayrışmaktadır. *Sızma testi* çalışmalarında analiz ekibi tarafından gerçekleştirilen testlerde bazı hazır yazılımlar kullanılmaktadır ve bu yazılımlar hem testlerin süresini kısaltmakta hem de daha az eforla çalışmanın yapılmasını sağlamaktadır. Bir analiz ekibi tarafından yapılması sebebiyle *sızma testi* çalışmaları, *zafiyet taraması* çalışmalarına kıyasla daha verimli olmakta ve daha sağlıklı sonuçlar elde edilmektedir.

Bu aşamada ortaya çıkan sonuçlardan ilki; kaynak koduna erişilmeden yapılan güvenlik testlerinden *zafiyet taraması* testinin, özellikle kurum içi personel tarafından geliştirilmiş veya kurum dışı bir kaynak tarafından kuruma özgü olarak geliştirilmiş web uygulamalarının zafiyet analizinde yeterli bir yöntem olmadığıdır. Bu tür

uygulamalar ilk kez ilgili kurum tarafından kullanılacak olduğundan, kapsamlı *kod gözden geçirme* faaliyetleri veya *sızma testi* faaliyetlerinden geçirilmesi daha doğru olacaktır.

Kuruma özel geliştirilen uygulamaların aksine, birden fazla kurumun kullandığı ve paket program olarak geliştirilmiş web uygulamalarının (Örneğin TÜRKSAT'ın geliştirdiği EBYS web uygulamasının) güvenliğinden ise temelde yazılımı geliştiren firma sorumlu olmaktadır. Bu tür uygulamaların kaynak kodları da genellikle kullanıcı kurumlarla paylaşılmadığı için, kaynak koda erişilerek yapılan güvenlik testleri zaten kullanıcı kurum için söz konusu olmamaktadır.

Bu durumda ortaya çıkan sonuçlardan ikincisi; güvenliğinden ilgili firmanın sorumlu olduğu bu tür uygulamalar için yüksek maliyetli *sızma testi* hizmet alımları yerine, daha düşük maliyetli ve daha hızlı gerçekleştirilebilen *zafiyet taraması* hizmet alımlarının yapılması yeterli olduğudur. Bununla birlikte *zafiyet taraması* çalışmaları çeşitli yazılımlar aracılığıyla otomatize şekilde gerçekleştirildiğinden, gerekli eğitimlerin alınması durumunda kurum içi personel tarafından da yapılabilecektir. Ancak tez çalışmasının *zafiyet taraması*'yla ilgili maddesinde de bahsedildiği gibi; *zafiyet taraması* çalışmaları sonrası ortaya çıkan bulguların ve sonuç raporunun değerlendirilmesinde kurum içi personelin eksik kalması durumunda, web uygulama güvenliği konusunda uzman olan bir firmadan danışmanlık hizmeti alınması doğru olacaktır.

Kaynak koduna erişilerek yapılan güvenlik analizi çalışmaları arasında *kod gözden geçirme* çalışmasının, kaynak koduna erişmeden yapılan güvenlik analiz çalışmaları arasında da *sızma testi* çalışmasının daha verimli ve etkili yöntemler olduğu değerlendirilmiştir.

Kod gözden geçirme ve *sızma testi* birbiriyle karşılaştırıldığında oraya çıkan sonuç ise şöyle olmaktadır: *Kod gözden geçirme* çalışması daha çok zaman ve efor gerektirdiğinden *sızma testine* göre daha maliyetli bir hizmettir. Bununla birlikte kod gözden geçirme çalışmasında, web uygulamasındaki güvenlik zafiyetlerinin eksiksiz bir şekilde tespit edilmesi daha yüksek bir olasılığa sahiptir. *Sızma testi* faaliyeti sonucunda ilgili uygulama ekranlarından veya metotlarından hangisinde güvenlik zafiyeti bulunduğu tespit edilirken; *Kod gözden geçirme* faaliyeti sonucunda tespit edilen güvenlik zafiyetinin, kaynak kodunun hangi kısmından kaynaklandığı ve kodda ne gibi değişiklikler yapılması gerektiğine dair çözüm önerileri de ilgili firma tarafından hazırlanan analiz raporunda belirtilir. Bu bağlamda *kod gözden geçirme* faaliyetinin, web uygulamasında bulunan zafiyetlerin kapatılması için yazılım geliştirici ekibin yapması gereken çalışmaları kolaylaştırmasını sağlar. Yine bu durumun bir sonucu olarak *kod gözden geçirme* çalışması, yazılım geliştirme ekibinin bilgi birikiminin artmasına büyük katkı sağlamaktadır.

Web uygulamalarıyla ilgili tüm zafiyet analiz yöntemleri göz önüne alındığında, Radyo ve Televizyon Üst Kurulu ve diğer kamu kurumları için zafiyet tespit ve analizinde aşağıdaki yol haritasının izlenmesinin doğru olacağı değerlendirilmektedir:

- Kurum içinde geliştirilen web uygulamalar için yazılım geliştiricilerin kaynak kodu yazdıkları aşamalarda *statik kod analizi* araçlarını kullanmaları önem taşımaktadır.
- *Kod gözden geçirme* hizmetinin en yüksek maliyete sahip olduğu varsayılarak, bir web uygulamasının yazılım yaşam döngüsü içerisinde, özellikle kaynak kodun yazılma aşamasının bittiği ve uygulamanın web ortamında canlı yayına alınma aşamasına geldiği dönemde, ilgili uygulamanın bir kez *kod gözden geçirme* çalışmasıyla denetlenmesi ve ortaya çıkan bulgulara göre kaynak kodda gerekli değişikliklerinin yapılması gerekmektedir.
- Saldırı yöntemlerinin de teknolojik ilerlemeye paralel olarak geliştiği göz önünde bulundurularak, belirli zaman aralıklarıyla ilgili web uygulama üzerinde *sızma testi* çalışmalarının gerçekleştirilmesi gerekmektedir.

- Eğer uygulama kaynak kodunda bir değişikliğe gidildiyse, özellikle son yapılan *kod gözden geçirme* esnasında kaynak kodda benzeri bulunmayan yeni metotlar kaynak koda eklendiyse, eklenen kod satırlarının yeniden *kod gözden geçirme* çalışmasına tabi tutulması gerekmektedir.

Web uygulaması geliştiricilerinin web uygulama güvenliği alanında bilgi birikimlerinin artırılması için aşağıdaki hususlara önem gösterilmesinin doğru olacağı değerlendirilmektedir:

- *OWASP, WASC, CWE* gibi web uygulama güvenliği üzerine çalışan kuruluşların çalışmalarının incelenmesi, hem web uygulama güvenlik açığı türleri hem de bunlardan korunmak için kaynak kodunda ne gibi önlemlerin alınması gerektiğinin irdelenmesi gerekmektedir.
- Web uygulama güvenliğiyle ilgili eğitim faaliyetlerinde teorik bilginin yanında pratik uygulamalara ağırlık verilmesi, zafiyet türlerinin kaynak kodunda olduğu durumda yaşanacak sızmanın ve alınan önlemlerin çalışma mekanizmasının daha iyi anlaşılması için faydalı olacaktır.
- Uygulamaların kaynak kodları yazılırken *statik kod analizi* araçlarının kullanılması hem ortaya çıkacak güvenlik açıklarını anında görme ve düzeltmeyi şansını artıracak, hem de kod kalitesini geliştirecek, hem de uygulama geliştiricinin yazılım bilgi birikimine katkı yapacaktır.

Web uygulama güvenliği disiplinindeki standart kurallarına ek olarak, tez çalışmasının dördüncü bölümünde işlendiği üzere *Siber Olaylara Müdahale Ekiplerinin Kuruluş, Görev Ve Çalışmalarına Dair Usul Ve Esaslar Hakkında Tebliğ* ve *Kişisel Verilerin Korunması Kanunu* göz önüne alınarak kişisel verileri işleyen web uygulamalarını yöneten Radyo ve Televizyon Üst Kurulunun ve diğer kamu kurumlarının, standardın üzerinde öneme haiz uygulamalar hakkında web uygulama güvenliği için alması gereken tedbirlerin ne olması gerektiğinde dair değerlendirme aşağıdaki gibidir:

- Kritik öneme sahip web uygulamaları, üzerinde çalıştıkları sunuculardaki tek web uygulama olmalıdır. Aynı sunucuda çalışan diğer bir web uygulamasına yönelik olası saldırının sunucu üzerinden ilgili uygulamayı da etkileme ihtimali düşünülmeli ve kritik uygulamaların sunucusuna başka web uygulama eklenmemelidir.
- Bir siber olay yaşanması durumunda SOME'lerin yaşanan olayı detaylarıyla USOM'a bildirme yükümlülükleri olduğu için loglama işlemi büyük önem arz etmektedir. Log kayıtları sayesinde yaşanan saldırının detayları hakkında bilgi sahibi edinmek mümkün olmaktadır. Bu noktadan hareketle, web uygulamalarının saldırgan tarafından ele geçirilme ihtimali düşünülerek, uygulamaların yönetici panelinden log kayıtları üzerinde silme ve düzenleme gibi işlemlere izin verilmemelidir.
- Kritik web uygulamalarında, özellikle kişisel verileri işleyen uygulamalarda, başarılı bir saldırı sonucu yaşanacak olası veri hırsızlığının boyutunu olabildiğinde küçültmek için yetkilendirme sistemi daha hassas tasarlanmalı ve hiyerarşik bir yetkilendirme sistemi oluşturulmalıdır. Her yetki rolü, ihtiyaç duyduğu kadar kişisel veriyi işleyebilmelidir.
- Kritik web uygulamaları zorunlu olmadıkça İnternet ağında hizmet vermemeli, kurum içi Intranet gibi ağlarda hizmet vermelidir. Uygulamanın kullanıcı ve yönetici olmak üzere iki ara yüzü varsa ve kullanıcı ara yüzünün İnternet ağında hizmet vermesi zorunluysa, bu durumda yönetici ara yüzü zorunlu olmadıkça İnternet ağında hizmet vermemelidir.
- Kritik uygulamalar üzerinde daha fazla zafiyet tespit çalışmaları yapılmalıdır.

SONUÇ

Bu tez çalışmasında, ilk etapta web uygulama teorisi, web uygulama güvenliğinin teoride hangi konumda bulunduğu ve güvenliğin hangi bileşenlerden oluştuğu, en sık karşılaşılan ve en ciddi tehlikeyi arz eden güvenlik açıklarının hangileri olduğu, literatüre göre bu güvenlik açıklarına karşı alınması gereken tedbirlerin neler olduğu incelenmiştir ve bu tedbirlerin etkinliği, tez çalışmasına özgü yazılan web uygulamalar üzerinden deney yöntemiyle test edilmiştir.

Güvenlik açıklarıyla ilgili alınması gereken önlemlere dair deneyin sonucu bulgular bölümünde ilk tabloda gösterilmiş, tartışma ve yorum bölümünde de sonuçlar yorumlanmıştır. Buna göre, literatürde önerilen önlem türleri etkilidir ve güvenlik açıklarının kapatılmasını sağlamaktadır. Web uygulaması geliştiren kişi ve kuruluşların, güvenlik açıklarını mutlaka literatürde önerilen yöntemlerle önlemesi gerektiği değerlendirilmektedir. Bu konuda OWASP gibi kuruluşların çalışmaları web uygulama geliştiricileri için önemli bir yol gösterici işlevi görmektedir.

Bu kapsamda, web uygulamaların mutlaka TLS/SSL sertifikası ile HTTPS protokolü üzerinden yayınlanması, uygulamaların loglama modülünün kapsamlı şekilde tasarlanması, kullanıcı girişleri için kullanılan parolaların hash algoritmaları ile kriptolanarak saklanması, parolaların belirli bir karmaşıklık düzeyinde belirlenmeye zorlanması, kullanıcı giriş ekranlarında Captcha adı verilen, “Ben robot değilim” şeklinde de bilenen doğrulama bileşenlerinin kullanılması, uygulama kodlarında kullanılan tüm dış bileşenlerin güvenli bir kaynak tarafından yayınlandığına dikkat edilmesi, uygulamada girdi (input) alanlarının bulunduğu sayfalarda Siteler Arası İstek Sahteciliği ve Siteler Arası Betik Çalıştırma zafiyetlerine karşı gerekli validasyon metotlarının oluşturularak önlemler alınması hususları tüm web uygulamalarda dikkat edilmesi gereken standart hususlardır. Bunun yanında, her

web uygulamasının kendine özgü modülleri olabilmektedir ve bu modüllerin yapısına göre söz konusu olabilecek güvenlik açığı türlerinin de o uygulamaya özgü olarak analiz edilmesi ve önlenmesi gerekmektedir.

İkinci olarak, web uygulama güvenliğinin sağlanması ve güvenlik açıklarının tespiti için hangi zafiyet analiz çalışmalarının var olduğu hususu literatür taraması ile incelenmiş ve genel kabul görmüş dört adet zafiyet analiz çalışmasının içeriklerinden detaylı olarak bahsedilmiştir.

Bulgular bölümünün ikinci tablosunda ilgili zafiyet analiz tespit çalışmalarının karşılaştırılması yapılmış, tartışma ve yorum bölümünde ise detaylı olarak ilgili çalışmaların artı ve eksileri yorumlanmıştır. Buna göre, en kapsamlı çalışma *kod gözdem geçirme* çalışması olmaktadır. Bu çalışma web uygulama güvenliği konusunda uzman bir ekibin bizzat uygulama kaynak kodunu okuyarak yaptığı bir çalışma olup, uygulamadaki tüm güvenlik açıklarının tespiti hususunda en yüksek ve en doğru zafiyet bulgusu oranına sahip çalışma olduğu değerlendirilmektedir. Yine bir uzman ekip tarafından gerçekleştirilen *Sızma Testi* çalışmasının da etkinliği yüksek bir çalışma olarak değerlendirilmektedir. *Kod gözden geçirmeye* göre maliyeti daha düşük bir çalışma olan *Sızma Testi* çalışması uygulamanın kaynak koduna erişilmeden yapıldığı için tüm güvenlik açıklarının tespiti hususunda *kod gözden geçirmenin* gerisinde kalmaktadır.

Hazır araçlar kullanılarak yapılan *statik kod analizi* ve *zafiyet taraması* çalışmalarının, web uygulamanın geliştirilmesi aşamasında geliştirici ekip için faydalı olduğu, ancak uygulama geliştirildikten sonra sadece bu araçlar kullanılarak yapılan zafiyet analiz çalışmalarının çok fazla yanlış alarm (false positive) şeklinde ifade edilen zafiyet bulgusunu analiz sonucunda gösterdiği anlaşılmış olup, bu zafiyet tespit çalışmalarının etkin ve verimli çalışmalar olmadığı değerlendirilmektedir. Bununla birlikte, zafiyet tespitinden kullanılan hazır araçların, bir ekip tarafından

gerçekleştirilen zafiyet tespit çalışmalarında bir yardımcı bileşen vazifesi görmesi ve analiz ekibinin işini kolaylaştırmasının mümkün olduğu değerlendirilmektedir.

Tez çalışmasında üçüncü olarak, web uygulama güvenliğini ilgilendiren mevzuat incelenmiştir. Yöntem olarak, web uygulamasını geliştiren ve sonrasında web uygulamasının işletiminden sorumlu olan kişileri hukuki olarak sorumlu tutan mevzuat hükümleri üzerinde durulmuştur. Bu kapsamda, ilk olarak *Siber Olaylara Müdahale Ekiplerinin Kuruluş, Görev Ve Çalışmalarına Dair Usul Ve Esaslar Hakkında Tebliğ* ve ikinci olarak *Kişisel Verilerin Korunması Kanunu* üzerinde çalışma yapılarak hangi maddelerin web uygulama güvenliğini ilgilendirdiği analiz edilmiştir.

Mevzuattaki ilgili maddeler ışığında, web uygulama güvenliğinin bilgi güvenliği için en önemli konulardan birisi olduğu, web uygulamalarının birçoğunun internete açık olması ve herkes tarafından her an erişilebilir olması sebebiyle güvenliğe azami önemin verilmesi gerekmektedir. Web uygulamasına yönelik olası saldırıların yaşanması durumunda kurumlar arası irtibat ve SOME ekipleri önemli unsurlar olmaktadır. Yaşanan saldırı hakkında gerekli bilgilerin elde edilebilmesi için web uygulamasındaki log kayıt modülünün de gerekli tüm bilgileri web uygulama sorumlularına verebilecek kapsamda geliştirilmesi gerektiği değerlendirilmektedir.

Radyo ve Televizyon Üst Kurulu Uzmanlık Tezi olarak hazırlanan bu çalışmada, web uygulama güvenliğinin sağlanması için literatürdeki öneriler ve kurallar taranmış, ilgili öneriler pratik olarak test edilmiştir. Uygulamanın genel güvenlik durumunu ölçmek için yapılan analiz çalışmaları karşılaştırılmış ve hangisinin tercih edileceği hususunda bir karşılaştırma tablosu ortaya çıkarılmıştır. Mevzuatın web uygulama güvenliğini sağlama hususunda kurumlara ne gibi sorumluluk yüklediği incelenmiştir.

Bu tez çalışması, bilgi güvenliğinin sağlanması ve kurumsal itibarın korunması için web uygulama güvenliğinin önemini ortaya koymuştur. Özellikle kişisel verileri işleyen web uygulamaları yüksek risk arz etmekte ve kamu kurumları için web uygulamalarının güvenliğini sağlamak kritik önem taşımaktadır.

KAYNAKÇA

17 Best Vulnerability Assessment Scanning Tool, https.//phoenixnap.com/blog/vulnerability-assessment-scanning-tools (13.03.2021)

2017 Cost of Data Breach Study, https.//www.ibm.com/downloads/cas/ZYKLN2E3 (10.03.2021)

A10:2017-Insufficient Logging & Monitoring, https.//owasp.org/www-project-top-ten/2017/A10_2017-Insufficient_Logging%2526Monitoring.html (10.03.2021)

About the OWASP Foundation, https.//owasp.org/about (29.01.2021)

Akcura Korhan, Shalchian Reze, Patil Abhijit, Singh Rattandeep, Tanna Jay, Static Versus Dynamic Source Code Analysis, *Electronic Notes in Theoretical Computer Science*, 2005, 131(2):3-14 s. 3, DOI:10.1016/j.entcs.2005.01.018

Altınkaynak, Mustafa, *Uygulamalı Siber Güvenlik ve Hacking*, Abaküs Yayınları, İstanbul, 2017

BEAM Teknoloji (2019), Güvenlik Analizi Raporu Örneği, *Web Güvenlik Analizi Sunumu*, BEAM Teknoloji, Ankara

Binary code, https.//www.britannica.com/technology/binary-code (11.03.2021)

Brute Force Attack, https.//www.imperva.com/learn/application-security/brute-force-attack (12.03.2021)

Burlu, Kamil, Bilişimim Karanlık Yüzü, NİRVANA, Ankara, 2015

CAPTCHA Nedir, https.//support.google.com/a/answer/1217728?hl=tr (12.03.2021)

CDNetworks Inc., State of the Web Security for H1 2020, https.//www.cdnetworks.com/download-assets/?CDNetworks-State-Of-web-Security-H1-2020 (29.01.2021)

Code Review Best Practices, https.//medium.com/palantir/code-review-best-practices-19e02780015f (12.03.2021)

Conklin Larry, Robinson Gary, *Code Review Guide*, OWASP, 2007

Cox, Vicki ve Reding, Eisner Elizabeth ve Wermers, Lynn, *HTML Illustrated Introductory,* Cengage Learning, 2006

Cross Site Request Forgery (CSRF), https.//owasp.org/www-community/attacks/csrf (13.03.2021)

Çontar, Furkan, Ağ ve Yazılım Güvenliği, KODLAB Yayınları, İstanbul 2017

DDoS saldırısı nedir, www.kaspersky.com.tr/resource-center/threats/ddos-attacks (23.02.2021)

Debarros, Anthony, *Practical SQL,* William Pollock, 2018

Deshpande Varun M, Nair Mydhili K., ShahDhrumil (2017), Major Web Application Threats for Data Privacy & SecurityDetection, Analysis and Mitigation Strategies, *International Journal of Scientific Research in Science and Technology, Sayı: 3, No:7, s. 186*

Doaa Nabil, Abeer Mosad, Hesham A. Hefny, *Web-Based Applications quality factors. A survey and a proposed conceptual model, Egyptian Informatics Journal*, Volume 12, Issue 3, 2011, s. 211-217

Dülger Berk, Tanlak Mehmet (December 2017), Kod Kalitesi Ve Kod Analizi Serisi #4, https://keytorc.com/blog/kod-kalitesi-ve-kod-analizi-serisi-4-dynamic-code-analysis (29.06.2021)

Engebretson, Patrick, *The Basics of Hacking and Penetration Testing*, SYNGRESS, ABD, 2011

F. Wibowo, H. H. Nuha and S. Wibowo, "Network Security Analysis Using HTTPS with SSL On General Election Quick Count Website," 2020 IEEE International Conference on Communication, Networks and Satellite (Comnetsat), Batam, Indonesia, 2020, s. 204-207, doi: 10.1109/Comnetsat50391.2020.9328940.

Financial Application Vulnerabilities, https.//www.ptsecurity.com/ww-en/analytics/financial-application-vulnerabilities (29.01.2021)

Free for Open Source Application Security Tools, https.//owasp.org/www-community/Free_for_Open_Source_Application_Security_Tools (11.03.2021)

Gehani, N., & Annamalai, M., The Database Book : Principles & Practice Using the Oracle Database: Vol. First edition. Silicon Press. 2011

Global Digital Population as of January 2021, https.//www.statista.com/statistics/617136/digital-population-worldwide (28.01.2021)

Gray, Neil, *Web Server Programming*, Wiley, 2003

Grossman, Jeremiah, *XSS Attacks Cross Site Scripting Exploits and Defence*, SYNGRESS, 2007

Guide to Web Application Penetration Testing, https.//relevant.software/blog/penetration-testing-for-web-applications (12.03.2021)

Hoffman, Andrew, *Web Application Security Exploitation and Countermeasures for Modern Web Applications*, O'REILLY, ABD, 2020, s. 217

HtmlHelper.Raw Method, https.//docs.microsoft.com/en-us/dotnet/api/system.web.mvc.htmlhelper.raw (05.03.2021)

ICloud Leaks of Celebrity Photos, https.//en.wikipedia.org/wiki/ICloud_leaks_of_celebrity_photos (29.01.2021)

IIS Modules Overview, https.//docs.microsoft.com/en-us/iis/get-started/introduction-to-iis/iis-modules-overview (23.02.2021)

Insecure Deserialization, https.//portswigger.net/web-security/deserialization (09.03.2021)

J. Grossman. CSRF, The Sleeping Giant, 2006

Jianjun Huang, Bin Liang, Jiagui Zhong, Qianqian Wang and Jingjing Cai, "Vulnerabilities static detection for Web applications with false positive suppression," *2010 IEEE International Conference on Information Theory and Information Security*, Beijing, China, 2010

Jussila, Jusha, *Http Cookıe Weaknesses, Attack Methods And Defense Mechanısms. A Systematıc Lıterature Revıew*, Unıversıty Of Jyväskylä

Kaynak Kod Analizi, https.//www.prismacsi.com/kaynak-kod-analizi (11.03.2021)

Lozano, Shah Walikar, *Hands-On Application Penetration Testing with Burp Suite*, Packt, 2019

McKinnon, Linda, *XML in 60 Minutes a Day*, WILEY, 2003

Metasploit Project, https.//www.wikiwand.com/en/Metasploit_Project (12.03.2021)

Most Popular Databases in 2020 and New Trends, https.//www.eversql.com/most-popular-databases-in-2020 (24.02.2021)

Online Brute Force Attacks, https.//godlikesecurity.com/index.php/tag/brute-force (11.03.2021)

Oppliger R. SSL and TLS : Theory and Practice. Vol Second edition. Artech House; 2016, s.21

OWASP Code Review Guide, https.//owasp.org/www-pdf-archive/OWASP_Code_Review_Guide_v2.pdf (11.03.2021)

OWASP Top Ten, https://owasp.org/www-project-top-ten (25.02.2021)

Pci Scanning Procedures - Pci Security Standarts Council, https://www.pcisecuritystandards.org (10.03.2021)

Popular Tools for Brute-force Attacks, https.//resources.infosecinstitute.com/topic/popular-tools-for-brute-force-attacks (12.03.2021)

Pros and cons of penetration testing, https.//www.itgovernance.eu/blog/en/pros-and-cons-of-penetration-testing (15.03.2021)

Reporting (Penetration Test), http://www.pentest-standard.org/index.php/Reporting (12.03.2021)

Robbins, Jennifer Niederst, *Learning Web Design*, O'REILLY, 2012

Russia accused of unleashing cyberwar to disable Estonia (2007), https.//www.theguardian.com/world/2007/may/17/topstories3.russia (29.01.2021)

Sarıman, Güncel ve Küçüksille, Ecir Uğur (2013), Güvenli Yazılım Geliştirme Süreci ve Statik Kod Analizi*, 6. Uluslararası Bilgi Güvenliği ve Kriptoloji Konferansı, s. 283.*

Secure Code Analysis Tools, https.//owasp.org/www-community/Source_Code_Analysis_Tools (11.03.2021)

Security Considerations (Entity Framework), https.//docs.microsoft.com/en-us/previous-versions/dotnet/netframework-4.0/cc716760(v=vs.100) (02.03.2021)

Sensitive Data Exposure, https.//www.contrastsecurity.com/knowledge-hub/glossary/sensitive-data-exposure (03.03.2021)

Session Management in HTTP: How does it work?, https.//www.packetlabs.net/session-management (12.03.2021)

SHA 256 Encrypt & Decrypt, https.//md5decrypt.net/en/Sha256 (02.03.2021)

Sharma Rupal, Sheth Ravi, "Discover Broken Authentication and Session Management Vulnerabilities in ASP.NET Web Application", *Oriental Journal Of Computer Scıence & Technology* Sayı: 10/2, 2017, s. 361-362

Sızma Testi mi? Zafiyet Taraması mı? https.//medium.com/@hemreyilmaz/s%C4%B1zma-testi-mi-zafiyet-taramas%C4%B1-m%C4%B1-8d4b01ed9b25 (13.03.2021)

Sızma Testleri (Penetrasyon Testleri), https.//www.bgasecurity.com/danismanlik-hizmetleri/penetrasyon-testi-sizma-testi (11.03.2021)

SOME (Siber Olaylara Müdahale Ekibi) Nedir, https.//www.bgasecurity.com/2018/11/some-siber-olaylara-mudahale-ekibi-nedir (13.03.2021)

Source Code Analysis Tools, https.//owasp.org/www-community/Source_Code_Analysis_Tools (11.03.2021)

Spett, Kevin, *SQL Injection: Are Your Web Applications Vulnerable*, SPI DYNAMICS, 2005

SQL injection has surfaced as the no. 1 attack in 2015, https.//www.helpnetsecurity.com/2015/12/11/sql-injection-has-surfaced-as-the-no-1-attack-in-2015 (25.02.2021)

Static Code Analysis, https.//docs.devexpress.com/CodeRushForRoslyn/115805/static-code-analysis (11.03.2021)

Static Code Analysis, https.//owasp.org/www-community/controls/Static_Code_Analysis (11.03.2021)

The 15 biggest data breaches of the 21st century (2021), https.//www.csoonline.com/article/2130877/the-biggest-data-breaches-of-the-21st-century.html (23.01.2021)

The Nessus Famly, https.//www.tenable.com/products/nessus (12.03.2021)

The Top 5 Pentesting Tools You Will Ever Need, https.//resources.infosecinstitute.com/topic/the-top-5-pentesting-tools-you-will-ever-need (12.03.2021)

Thomas, Stephan. A. HTTP Essentials: Protocols For Secure, Scaleable, Web Sites, Wiley, 2001

TOP 40 Static Code Analysis Tools (Best Source Code Analysis Tools), https.//www.softwaretestinghelp.com/tools/top-40-static-code-analysis-tools (11.03.2021)

Top Ten Framework for Web Applications, https.//www.geeksforgeeks.org/top-10-frameworks-for-web-applications (23.02.2021)

Tumin, Sharil ve Encheva, Sylvia (2012), A Closer Look at Authentication and Authorization Mechanismsfor Web-based Applications, *5th World Congress: Applied Computing Conference*, s. 100-101.

Unrestricted File Upload, https.//owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload (12.03.2021)

Usage statistics of Default protocol https for websites, Erişim Tarihi: 23.02.2021. https.//w3techs.com/technologies/details/ce-httpsdefault

Usage Statistics of HTML5 for Websites, https.//w3techs.com/technologies/details/ml-html5 (23.02.2021)

V. Dehalwar, A. Kalam, M. L. Kolhe and A. Zayegh, "Review of web-based information security threats in smart grid," *2017 7th International Conference on Power Systems (ICPS)*, Pune, India, 2017, pp. 849-853, doi: 10.1109/ICPES.2017.8387407. s. 852

ValidateInputAttribute Class, https.//docs.microsoft.com/en-us/dotnet/api/system.web.mvc.validateinputattribute (05.03.2021)

Vulnerability Assesment, https.//www.imperva.com/learn/application-security/vulnerability-assessment (13.03.2021)

Web Uygulama Güvenliğinde False Positive Sorunu, https.//www.netsparker.com.tr/blog/web-guvenligi/web-uygulama-guvenliginde-false-positive-sorunu (25.02.2021)

What are Black Box, Grey Box, and White Box Penetration Testing, Erişim Tarihi: 11.03.2021,

What are CRUD Operations. How CRUD Operations Work, Examples, Tutorials & More, https.//stackify.com/what-are-crud-operations/ (02.03.2021)

What Is Web Applıcatıon Securıty And Why Is It Important?, https.//blog.eccouncil.org/what-is-web-application-security-and-why-is-it-important (28.01.2021)

What is an SSL, https.//www.znetlive.com/blog/what-is-an-ssl-certificate-and-why-is-it-a-must-have-for-every-website (13.03.2021)

What is Grey Box Testing? Techniques, Example, https.//www.guru99.com/grey-box-testing.html (11.03.2021)

What is Javascript, https.//developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript (02.03.2021)

What is SSL, TLS and HTTPS, https.//www.websecurity.digicert.com/security-topics/what-is-ssl-tls-https (23.02.2021)

What is Wireshark? What this essential troubleshooting tool does and how to use it, https.//www.csoonline.com/article/3305805/what-is-wireshark-what-this-essential-troubleshooting-tool-does-and-how-to-use-it.html (05.03.2021)

William Zeller ve Edward W. Felten, Cross-Site Request Forgeries. Exploitation and Prevention,, *International Journal of Computer Applications*, 106(11):20-25 2014, s. 20

Wireshark Nedir? Nasıl Kullanılır?, https.//www.karel.com.tr/bilgi/wireshark-nedir-nasil-kullanilir (01.03.2021)

Wögerer, Wolfgang, *A Survey of Static Program Analysis Techniques*, Technische Universität Wien, 2005

XML External Entities(XXE) Processing, https.//owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing (09.03.2021)

XML Overview, https.//www.tutorialspoint.com/xml/xml_overview.htm (03.03.2021)

XXE Bug Patched in Facebook Careers Third-Party Service, https.//threatpost.com/xxe-bug-patched-in-facebook-careers-third-party-service/110151 (08.03.2021)

Z. Liu, J. Pen, "The Analysis of HTTPS Privacy Protection Based on Malicious Code Injection," *2020 IEEE 20th International Conference on Communication Technology (ICCT)*, Nanning, China, 2020, s. 1075-1082, doi: 10.1109/ICCT50939.2020.9295688.

Zenmap GUI, https.//nmap.org/zenmap (13.03.2021)

ÖZET

İnternetin ortaya çıkardığı en önemli yeniliklerden olan web tabanlı yazılım uygulamaları; iletişim, haber alma, finans, alışveriş gibi ihtiyaçları karşılama noktasında insan hayatında önemli bir yer tutar hâle gelmiştir. Buna paralel olarak uygulamaların maruz kaldığı saldırıların artması ve saldırı türlerinin çeşitlenmesi sebebiyle web uygulama güvenliği bu tezin konusu olarak seçilmiştir.

Tez çalışmasının birinci bölümünde web uygulaması kavramı açıklanmış, ilişkili kavramların ne anlama geldiği, tanımlar ve çeşitli görseller üzerinden anlatılmıştır. İkinci bölümde, web uygulamalarının maruz kaldığı güvenlik açıklarının türleri, web güvenliği konusunda uzmanlaşan bazı kuruluşun bu konuda yaptığı çalışmalardan da faydalanarak incelenmiştir. Tez çalışmasına özgü bazı web uygulamaları yazılmış ve literatürde ilgili güvenlik açıklarına karşı önerilen çözüm önerileri, yazılan uygulamalar üzerinden incelenerek veri toplanmıştır. Üçüncü bölümde, web uygulamalarında bulunan güvenlik açıklarını tespitte kullanılan zafiyet analiz yöntemleri incelenmiş, farklı yöntemlerin avantajları ve dezavantajlarından bahsedilmiştir. Dördüncü bölümde, mevzuatta web uygulama güvenliğini ilgilendiren kısımlar incelenmiş, web uygulama yöneticilerinin ne gibi hukuki sorumluluklar taşıdığı irdelenmiştir. Analiz ve bulgular bölümünde, tez çalışmasında toplanan veriler açıklanmış, tartışma ve yorum kısmında ise web uygulama güvenliğinin sağlanması yolunda değerlendirmeler yapılmış, Radyo ve Televizyon Üst Kurulu için bir yol haritası ortaya konulmuştur. Sonuç bölümünde tez çalışmasının ortaya çıkardığı netice değerlendirilmiştir.

Anahtar Kelimeler: Web Uygulama Güvenliği, Siber Güvenlik, Bilgi Güvenliği, Yazılım Güvenliği

Arşiv Kayıt Bilgileri

Tezin Adı:	Web Tabanlı Uygulamalarda Güvenlik Açıkları ve Önlemler
Tezin Yazarı:	Necmettin ÜNAL
Tezin Danışmanı:	Cem ULUOĞLAKCI
Tezin Konumu:	Uzmanlık Tezi
Tezin Tarihi:	Nisan / 2021
Tezin Alanı:	Web Uygulama Güvenliği
Tezin Yeri:	RTÜK / ANKARA

ABSTRACT

Web based software applications are among the most important innovations that emerged with the Internet. They have become an important aspect of the human life in meeting needs such as communication, news, finance and shopping. At the same time, the attacks to which the applications are exposed have increased and the attack types have diversified so that web application security has been chosen as the subject of the thesis.

In the first chapter of the thesis, the meaning of the web application has been explained and the meaning of the related concepts has been explained over the definitions and various visuals. In the second chapter, the types of vulnerabilities that web applications are exposed to are examined by making use of the studies of some organizations specializing in web security. Some web applications specific to the thesis were coded and the data was collected by examining the proposed solutions to the related security vulnerabilities in the literature through those applications. In the third chapter, vulnerability analysis methods used to detect vulnerabilities in web applications have been examined. The advantages and disadvantages of different methods are mentioned. In the fourth chapter, parts of the legislation concerning web application security and the legal responsibilities of web application administrators has been examined. In the analysis and findings section, the data collected in the thesis are presented. In the Discussion and Comment chapter, suggestions have been made to ensure web application security and a road map for Radio and Television Supreme Council has been put forward. In the conclusion part, the result of the thesis has been evaluated.

Key Words: Web Application Security, Cyber Security, Information Security, Secure Software

ÖZGEÇMİŞ

Necmettin Ünal 1990 yılında Ankara'da doğdu. Sırasıyla Gazi İlköğretim Okulu, Gülen Muharrem Pakoğlu İlköğretim Okulu ve Ankara Atatürk Lisesinde okuduktan sonra 2016 yılında Çankaya Üniversitesi Elektronik ve Haberleşme Mühendisliği bölümünden mezun olup lisans derecesi aldı. 2018 yılından itibaren Radyo ve Televizyon Üst Kurulunda görev yapmaktadır. Evli ve bir çocuk babasıdır.