T.C.
SÜLEYMAN DEMİREL ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YAPAY ZEKÂ VE DİNAMİK ANALİZ TABANLI WEB UYGULAMA ZAFİYET TARAYICISI

Mehmet Ali YALÇINKAYA

Danışman
Prof. Dr. Ecir Uğur KÜÇÜKSİLLE

DOKTORA TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
ISPARTA - 2020

TEZ ONAYI

Mehmet Ali YALÇINKAYA tarafından hazırlanan **"Yapay Zekâ ve Dinamik Analiz Tabanlı Web Uygulama Zafiyet Tarayıcısı"** adlı tez çalışması aşağıdaki jüri üyeleri önünde Süleyman Demirel Üniversitesi Fen Bilimleri Enstitüsü **Bilgisayar Mühendisliği Anabilim Dalı**'nda **DOKTORA TEZİ** olarak başarı ile savunulmuştur.

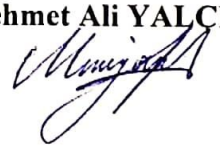
Danışman	**Prof. Dr. Ecir Uğur KÜÇÜKSİLLE** Süleyman Demirel Üniversitesi	.....
Jüri Üyesi	**Prof. Dr. Tuncay AYDOĞAN** Isparta Uygulamalı Bilimler Üniversitesi	.....
Jüri Üyesi	**Doç. Dr. Mustafa YAĞCI** Kırşehir Ahi Evran Üniversitesi	.....
Jüri Üyesi	**Dr. Öğr. Üyesi Tuna GÖKSU** Isparta Uygulamalı Bilimler Üniversitesi	.....
Jüri Üyesi	**Dr. Öğr. Üyesi Murat IŞIK** Kırşehir Ahi Evran Üniversitesi	.....

Enstitü Müdürü **Doç. Dr. Şule Sultan UĞUR** .....

TAAHHÜTNAME

Bu tezin akademik ve etik kurallara uygun olarak yazıldığını ve kullanılan tüm literatür bilgilerinin referans gösterilerek tezde yer aldığını beyan ederim.

Mehmet Ali YALÇINKAYA



İÇİNDEKİLER

ÖZET

Doktora Tezi

YAPAY ZEKA VE DİNAMİK ANALİZ TABANLI WEB UYGULAMA ZAFİYET TARAYICISI

Mehmet Ali YALÇINKAYA

Süleyman Demirel Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Prof. Dr. Ecir Uğur KÜÇÜKSİLLE

Günümüz bilişim dünyasında internet kullanımının gün geçtikçe yaygınlaşması, web uygulama kullanımının da aynı oranda artmasına neden olmuştur. Bu duruma paralel olarak insanlar alıveriş, yemek siparişi, vatandaşlık işlemleri, fatura ödeme, bankacılık gibi birçok işlemi web uygulamaları üzerinden gerçekleştirmektedirler. Web uygulama kullanımının bu kadar yaygınlaşması ve hassas veriler üzerinde çalışıyor olması, söz konusu uygulamaları siber saldırganların en önemli hedeflerinden biri haline getirmiştir. Web uygulamaları üzerinde her gün yeni bir zafiyet ortaya çıkmakta, söz konusu zafiyetlere yapılan saldırılar ciddi maddi ve manevi zararlara neden olmaktadır.

Web uygulamaları üzerinde yer alan zafiyetlerin saldırganlardan önce tespit edilerek kapatılması alınabilecek en önemli güvenlik önlemlerinin başında gelmektedir. Web uygulamaları üzerinde yer alan güvenlik zafiyetlerinin tespit edilmesinde otomatize web uygulama zafiyet tarayıcıları yaygın olarak kullanılmaktadır. Mevcut web uygulama zafiyet tarayıcıları, web uygulaması üzerinde tüm zafiyet testlerini herhangi bir sıralama ya da ayrım gözetmeden gerçekleştirmektedirler.

Bu çalışmada dinamik analiz ve yapay zekâ tabanlı, web uygulamalarını GET ve POST metodları üzerinden test edebilen, 21 farklı zafiyet türü için test sınıfına sahip bir web zafiyet tarayıcısı geliştirilmiştir. Geliştirilen zafiyet tarayıcısı, yine bu çalışma kapsamında oluşturulan, 262 farklı web uygulamasını bünyesinde barındıran bir web uygulama test laboratuvarı üzerinde test edilmiştir. Yapılan testler sonrasında elde edilen sonuçlardan bir veri seti oluşturulmuştur. Oluşturulan veri seti içerdiği öznitelikler bakımından literatürde tek olma özelliği taşımaktadır. Söz konusu veri seti kullanılarak önce web sayfası sınıflandırma işlemi gerçekleştirilmiştir. Gerçekleştirilen sınıflandırma işleminde Rastgele Orman algoritması kullanılmış, başarı oranı %96 olarak elde edilmiştir. Sayfa sınıflandırma işleminin ardından, veri seti kullanılarak zafiyetler arasında birliktelik analizi gerçekleştirilmiştir. Gerçekleştirilen analiz sonrasında hangi tür zafiyetlerin, hangi zafiyetler ile aynı yerlerde görülebileceği incelenmiştir. Oluşturulan sayfa sınıflandırma- birliktelik analizi tabanlı modelin, standart tarama modellerine göre %21 oranında süre kazancı sağladığı görülmüştür. Bu tez çalışması gerçekleştirilen birliktelik analizi açısından da, literatürde tek olma özelliği taşımaktadır. Gerçekleştirilen çalışmada ayrıca,

kapsam genişletme işleminde web sayfası gezilirken, kullanıcı giriş sayfalarının tespiti için sınıflandırma işlemi kullanılmıştır. Kullanıcı giriş sayfalarının otomatik olarak tespit edilerek giriş yapılması, mevcut kullanılmakta olan web afiyet tarayıcılarda olmayan bir özellik olup, bu alanda literatüre kazandırılmış yeni bir tekniktir.

Ulusal yazılım güvenliği alanında yapılan çalışmaların azlığı göz önüne alındığında, bu tez çalışması kapsamında geliştirilen yapay zekâ tabanlı web zafiyet tarayıcısının çalışma alanına ciddi katkıda bulunacağı düşünülmektedir.

Anahtar Kelimeler: Web Güvenliği, Dinamik Kod Analizi, Makine Öğrenmesi, Siber Güvenlik.

2020, 152 sayfa.

ABSTRACT

PhD. Thesis

ARTIFICIAL INTELLIGENCE AND DYNAMIC ANALYSIS BASED WEB APPLICATION VULNERABILITY SCANNER

Mehmet Ali YALÇINKAYA

Süleyman Demirel University
Graduate School of Natural and Applied Sciences
Department of Computer Engineering

Supervisor: Prof. Dr. Ecir Uğur KÜÇÜKSİLLE

In today's informatics world, the widespread using of internet has caused the use of web applications to increase at the same rate. Nowadays, people carry out many applications such as shopping, food ordering, citizenship transactions, bill payment and banking through web applications. The widespread using of web applications and the fact that they are working on sensitive data have made these applications one of the most important targets of cyber attackers. A new vulnerability is emerging every day on the web applications, attacks on these vulnerabilities cause serious material and moral damages.

One of the most important security measures that can be taken is the detection and closure of vulnerabilities on web applications before the attackers. Automated web vulnerability scanners are widely used to detect security vulnerabilities on web applications. Existing web vulnerability browsers perform all vulnerability testing on the web application without any sorting or distinction. In this study, a web vulnerability scanner has been developed based on dynamic analysis and artificial intelligence, which can test web applications through GET and POST methods and has test classes for 21 different vulnerability types. Developed vulnerability scanner was tested on a web application testing laboratory that also includes 300 web applications created within the scope of this study. A data set was created from the results obtained after the tests. The data set is unique in the literature in terms of its attributes. First, web page classification process was performed by using this data set. The success rate in the classification process was 96%. After the page classification process, relationship analysis between vulnerabilities was performed using the data set. After the transaction, which kind of vulnerability, which vulnerability can be seen in the same places was examined. This thesis is unique in the literature in terms of performed relationship analysis. In this study, while the web page was visited for the scope extension, classification process was used for the identification of the user login pages. Automatically detecting user login pages and logging in is a feature that is not available in the web browser currently in use and is a new technique that has been introduced to the literature in this field.

Considering the scarcity of studies in the field of national software security, it is thought that the artificial intelligence-based web vulnerability browser developed within the scope of this thesis will contribute to the study area.

Keywords: Web Application Security, Dynamic Code Analysis, Machine Learning, Cyber Security.

2020, 152 pages.

TEŞEKKÜR

Bu araştırma için beni yönlendiren, karşılaştığım zorlukları bilgi ve tecrübesi ile aşmamda yardımcı olan, değerli danışman hocam Prof. Dr. Ecir Uğur KÜÇÜKSİLLE' ye teşekkürlerimi sunarım.

Tezimin her aşamasında gösterdiği anlayış ve desteğinden dolayı değerli eşime, maddi ve manevi desteklerini hiçbir zaman esirgemeyen anneme ve babama sonsuz sevgi ve saygılarımı sunarım.

Mehmet Ali YALÇINKAYA
ISPARTA, 2020

ŞEKİLLER DİZİNİ

Sayfa

ÇİZELGELER DİZİNİ

Sayfa

SİMGELER VE KISALTMALAR DİZİNİ

ASP	Active Server Pages
CRLF	Carriage Return Line Feed
GIS	Generalized Instance Set
HTML	Hyper Text Markup Language
IIS	Internet Information Service
IP	Internet Protocol
KNN	K Nearest Neighbour
PHP	Hypertext Preprocessor
SQL	Structured Query Language
URL	Uniform Resource Locator
WWW	World Wide Web

1. GİRİŞ

Günümüzde teknoloji alanında meydana gelen gelişmeler ile bilgisayar, tablet, akıllı telefon gibi araçların kullanımı oldukça yaygınlaşmıştır. Söz konusu cihazların yanında internet kullanımı, dolayısıyla da web uygulama sayısı ciddi boyutlara ulaşmıştır. Günümüzde kurum ve kuruluşlar, hizmet verdikleri kitlelere daha hızlı ulaşabilmek ve daha etkili hizmet sağlamak adına web uygulamalarını etkin bir şekilde kullanmaktadırlar. Web uygulamaları üzerinden kullanıcılar; alışveriş, araştırma, dosya transferi, sosyal medya ile haberleşme gibi birçok işlemi gerçekleştirilebilmektedir. Ayrıca ülkemizin de dâhil olduğu birçok ülkede vatandaşlar, çeşitli vatandaşlık işlemlerini, devletlerine ait web uygulamaları üzerinden gerçekleştirmektedir. Web uygulamaları kurumların, firmaların hatta devletlerin dış dünyaya açılan kapısı, diğer bir tabirle vitrini haline gelmiştir. Web uygulamalarının bu derece yaygın olarak kullanılması ve çok hassas veriler üzerinde işlem yapması, söz konusu uygulamaları siber saldırganların bir numaralı hedefi haline getirmiştir.

Günümüzde web uygulama geliştirme işlemi başlı başına bir sektör haline gelmiştir. Web uygulamalarında kullanıcı ile etkileşime girecek arayüzün ve sayfaların tasarlanması, kullanıcı işlemlerinin yorumlanarak sunucuya iletilmesi, sunucudan gelen yanıtların kullanıcıya aktarılması, web uygulamaları ile veri tabanı sunucuları arasında işlemlerin gerçekleştirilmesi gibi süreçler web uygulaması geliştirmenin en önemli bileşenleridir. Her bir süreç için farklı programlama dilleri ve frameworkler kullanılmaktadır. Belirtilen aşamaların herhangi birinde gereken dikkat ve önemin verilmemesi, girdi kontrollerinin yetersiz yapılması, filtreleme işlemlerinin unutulması gibi hatalar, siber saldırganların web uygulamalarına karşı takındığı saldırgan tutumla birleştiğinde ortaya çok kritik veri kayıpları, maddi ve manevi hasarlar çıkmaktadır.

Web uygulama zafiyetlerinin ve web uygulamalarına yönelik saldırıların daha iyi anlaşılabilmesi için öncelikle bilgi güvenliği ve güvenli yazılım geliştirme kavramlarına değinmek gerekmektedir. Bilgi güvenliği; bilgilerin elektronik ortamda saklanması ve taşınması sırasında, bilgi gizliliğinin ve bütünlüğünün bozulmaması için harcanan çabaların tümüne verilen isimdir. Tam kapsamlı bir bilgi güvenliğinin sağlanabilmesi için detaylı bir bilgi güvenliği politikası belirlenmeli ve eksiksiz yerine

getirilmelidir (Canberk ve Sağıroğlu, 2006). Bilgi güvenliği; ağ güvenliği, uç nokta güvenliği, veri güvenliği, uygulama güvenliği, kimlik ve erişim güvenliği, güvenlik yönetimi ve sanallaştırma olmak üzere sekiz alt kategoriye ayrılmıştır. Söz konusu alt kategoriler içerisinde yer alan uygulama güvenliğinde, uygulamaların geliştirilmesi ve kullanımı süreçlerinde karşılaşılabilecek zafiyetler incelenmektedir. Uygulama güvenliği kategorisi; web uygulama güvenlik duvarları, uygulama testi, güvenli geliştirme, tehdit modelleme, geliştirme süreci, test metodolojileri, web sızma testleri, web uygulama güvenlik duvarları, web uygulama değerlendirme ve yönetilir servisler kategorilerine ayrılmaktadır (Sarıman, 2015).

Bir web uygulamasının olabildiğince güvenli, saldırılara kapalı olarak geliştirilebilmesi ve kullanılabilmesi için, bir önceki paragrafta belirtilen kategorilerden güvenli geliştirme, geliştirme süreci, uygulama testi ve web sızma testleri büyük önem taşımaktadır. Web uygulamalarında tespit edilen zafiyetlerin en temel kaynağı, yazılım temelli hatalardır. Günümüzde web uygulama geliştiricilerin birçoğunun hız, bütçe, kullanıcı talepleri ve daha fazla fonksiyonellik gibi konulara yoğunlaşması nedeniyle güvenlik konusu unutulmakta ya da göz ardı edilmektedir. Uygulamaların geliştirilmesi esnasında yazılım geliştiricilerin güvenli yazılım geliştirme modellerinden birini benimseyerek uygulaması ve yazılım geliştirme süreci aşamalarının (gereksinim, tasarım, geliştirme, test, doğrulama) tümünü eksiksiz gerçekleştirmesi gerekmektedir. Bir güvenlik zafiyetinin, yazılım geliştirmenin erken bir sürecinde farkedilerek düzeltilmesi, daha ileri süreçlerde farkedilerek düzeltilmesine kıyasla çok daha az maliyetli olacaktır (Michael, 2005). Yazılımın geliştirilmesi esnasında farkedilmeyen zafiyetler daha sonra saldırganlar tarafından tespit edilirse ortaya ciddi derecede veri ve hizmet kayıpları çıkabilmektedir.

Çeşitli kuruluşlardan gelen güvenlik ihlali raporları, web uygulamalarının güvenliğinin sağlanmasının önemini vurgular niteliktedir. Verizon' un araştırma raporuna göre, 2019' daki güvenlik olaylarının %60' ı web uygulamalarına yönelik saldırılardan kaynaklanmaktadır (Verizon, 2019). ITRC (Identity Theft Resource Center) tarafından yayınlanan bir raporda ise, 2018 yılında web uygulamalarına yönelik saldırıların bir önceki yıla göre %23 arttığı ve çoğunun iş, askeri, bankacılık ve tıbbi amaçlı olduğu belirtilmektedir (ITRC, 2018). Bu denli yüksek oranda

gerçekleştirilen saldırıların nedeni, geliştirilen web uygulamalarında yer alan çeşitli güvenlik açıklarıdır.

Web uygulamalarının saldırganlar tarafından bu denli yoğun olarak hedef alınması, web uygulamaları üzerinden çeşitli hizmetler veren kurum ve kuruluşlar için ciddi sorun teşkil etmektedir. Kurumlar sahip oldukları web uygulamalarının saldırganlar tarafından istismar edilmesinin önüne geçebilmek amacıyla web uygulaması üzerindeki olası zafiyetleri saldırganlardan önce tespit ederek gerçek bir saldırı öncesinde ilgili zafiyetleri kapatmaya çalışmaktadırlar.

Bir web uygulaması üzerinde yer alan güvenlik açıklarını belirlemek için statik analiz ve dinamik analiz yöntemleri kullanılmaktadır. Statik analiz yönteminde, web uygulamasının kaynak kodu incelenmekte ve olası güvenlik açıklarının tespit edilebilmesi için tüm program dallanmaları denenmektedir. Fakat statik kod analizinin dezavantajı; sadece uygulamanın çalıştırılması sırasında karşılaşılan güvenlik açıklarını belirlemede yetersiz kalmasıdır. Dinamik analiz yönteminde ise, aktif olarak hizmet vermekte olan bir web uygulamasına yönelik zararlı girdiler gönderilerek uygulamanın yanıtı analiz edilmekte, elde edilen yanıtlara göre güvenlik açıkları tespit edilmektedir. Dinamik analiz yöntemi ile elde edilen sonuçlar, uygulamanın çalışma zamanı (runtime) davranışlarına dayanarak üretildiğinden, statik analize göre daha güvenilir sonuçlar vermektedir. Bunun yanında uygulamanın olası tüm alanları test edilemediğinden dolayı, statik analize göre sayısal olarak daha az sonuç ortaya çıkabilmektedir (Deepa ve Thilagam, 2016).

Bu tez çalışmasında web uygulamaları üzerinde yer alan zafiyetlerin tespiti için dinamik analiz yöntemi kullanılmıştır. Dinamik analiz ile web uygulamasında yer alan zafiyetleri tespit etmenin en etkili yolu; web uygulamasının bir güvenlik uzmanı tarafından sızma testine tabi tutulmasıdır. Ülkemizde, Bankacılık Düzenleme ve Denetleme Kurulu 2007 yılında, bilgi sistemlerine yönelik olarak belirli periyotlarla sızma testlerinin gerçekleştirilmesi zorunlu hale getirilmiştir. İlgili resmi gazete yayınında; "Kurum, Kuruluş ve Ortaklıkların bilgi sistemleri, bilgi güvenliği gereklerinin yerine getirilmesi hususunda herhangi bir görevi bulunmayan ve sızma testi konusunda ulusal veya uluslararası belgeye sahip gerçek veya tüzel kişiler

tarafından en az yılda bir kez sızma testine tabi tutulur" denilmektedir (Resmi Gazete, 2007).

Web uygulama güvenliği sağlamanın öneminin bu kadar artması, dinamik analiz yolu ile web uygulama sızma testlerini gerçekleştirecek sızma testi uzmanlarına olan ihtiyacı da beraberinde arttırmıştır. Sızma testi uzmanları kurumun sahip olduğu web sayfası üzerinden yola çıkarak ilk önce kuruma ait ulaşabildiği tüm alt sayfaları (subdomain ve alt URL adresleri) toplamaktadır. Güvenlik uzmanları, test edilecek web adreslerinin toplanmasından sonra, ilgili sayfalar üzerinde çeşitli saldırı teknikleri kullanarak, web uygulamasının ilgili zafiyetleri barındırıp barındırmadığını kontrol etmekte ve raporlamaktadır. Fakat büyük ölçekli bir kurumun web sayfasının birçok alt domain ve alt URL adreslerine sahip olduğu, bunun yanında her bir web sayfası içerisinde değişken miktarda kullanıcı girdi alanı olduğu düşünüldüğünde, test işleminin çok zaman alacağı, testin istenilen süre içerisinde tamamlanamayacağı aşikardır. Ayrıca her geçen gün yeni bir tür güvenlik açığının bulunması, test edilmesi gereken yeni bir zafiyet anlamına gelmektedir. Bu da aynı şekilde test sürecinin uzamasına neden olmaktadır. Bu gibi nedenlerden dolayı, dinamik analiz tabanlı otomatik web uygulama zafiyet tarayıcıları günümüzde büyük önem kazanmışlardır.

Web uygulama zafiyet tarayıcıları kendisine parametre olarak verilen URL adresi üzerinde, bünyesinde barındırdığı test metodlarını kullanarak güvenlik testleri gerçekleştirmektedir. Sızma testi uzmanlarının elle yapacağı zafiyet testlerini hızlı bir şekilde otomatik olarak gerçekleştiren web uygulama zafiyet tarayıcıları, test sonrasında elde ettiği sonuçları da raporlamaktadır. Öyle ki web sızma testinde ileri seviyede uzman olmayan kişiler dahi web sayfalarını söz konusu araçlar ile tarayabilmekte ve elde ettiği raporu yorumlayarak gerekli önlemleri alabilmektedir.

Web uygulama testlerinde aktif olarak kullanılan zafiyet tarayıcılar, tarama esnasında test sınıflarını herhangi bir mantık ya da modele oturtmadan, sırayla çalıştırmaktadır. Literatürde incelenen çalışmalarda ise tek ya da birkaç zafiyetin test edilmesini amaçlayan araçlar geliştirilmiştir. Bu çalışmada temel olarak yukarıda bahsedilen iki eksikliğin giderilmesi amaçlanmıştır. Gerçekleştirilen çalışmada, "Sansar" isminde dinamik analiz tabanlı bir web uygulama zafiyet tarayıcısı geliştirilmiştir. Sansar aracı, hem literatürde incelenen çalışmalardan daha fazla sayıda zafiyeti test edebilmekte,

hem de çalıştırılacak test sınıflarını, mevcut zafiyet tarayıcıların aksine, yapay zeka temelli bir algoritma doğrultusunda tanımlanmış kurallar ile seçmektedir.

Geliştirilen tarayıcı ilk olarak kendisine parametre olarak gönderilen URL adresinden yola çıkarak tüm web uygulamasını gezmekte (örümcekleme- indeksleme yapmakta), gezdiği adresleri kaydetmekte böylelikle de web uygulama sızma testinin kapsamını belirlemektedir. Geliştirilen web uygulama zafiyet tarayıcısı, mevcut zafiyet tarayıcıların birçoğundan daha kapsamlı örümcekleme işlemi yapmaktadır. Geliştirilen araç, sayfa sınıflandırma işlemi sonrası tespit ettiği kullanıcı giriş sayfası üzerinde, çalıştırma esnasında kendisine verilen geçerli kullanıcı adı ve parola bilgilerini kullanmakta, başarı ile login işlemini gerçekleştirmektedir. Bu sayede, admin panelleri gibi zafiyetlerin bulunması muhtemel en kritik alanlar başarılı bir şekilde test edilebilmektedir. Geliştirilen Sansar aracının mevcut birçok web zafiyet tarayıcısından üstün bir diğer noktası, web sayfaları üzerinde yer alan form ve girdi alanlarını varsayılan (default) değerler ile doldurarak sayfanın bir sonraki aşamaya geçmesini sağlamasıdır. Örneğin arama girdi alanı tespit edildiğinde, Sansar tarafından arama inputuna varsayılan bir değer girilerek gönderilecek, böylelikle de arama sonuçlarının listelendiği sayfaya ulaşılabilecektir. Geliştirilen Sansar aracı, güvenlik testlerini hem GET hem de POST metodu üzerinde gerçekleştirebilmektedir. Geliştirilen test aracı en bilinen açık kaynak kodlu web zafiyet tarayıcılarından Wapiti ve Wascan ile karşılaştırıldığında, test edebildiği zafiyet sayısı yönünden de üstünlük göstermektedir.

Geliştirilen "Sansar" aracının yeterliliğinin test edilmesi için hem local hem de uzak sunucu üzerine ASP ve PHP programlama dillerinde geliştirilmiş 262 farklı web uygulaması yüklenmiştir. Github, Gitlab, Sourceforge gibi repositoryler üzerinde geliştiricileri tarafından kullanıma açılmış alışveriş, haber, spor, okul otomasyonu, hastane otomasyonu, kütüphane otomasyonu gibi farklı türdeki web uygulamalarının kurulması ile, literatürde görülen en geniş ve en fazla çeşitliliğe sahip web uygulama sızma testi laboratuvarı kurulmuştur. İnternet üzerinde konumlandırılmış web uygulamalarının test edilebilmesi için yasal kanunlar çerçevesinde web sitesi sahiplerinden izin almak gerekmektedir. Ayrıca kurum ve kullanıcılar, web sitesine ya da veri tabanına bir zarar gelir düşüncesi ile web uygulamalarının test edilmesine sıcak bakmamaktadırlar. Kurulumu gerçekleştirilen web uygulama test laboratuvarı

sayesinde, kullanıcı izni almaya gerek kalmadan, web sitesine ya da veri tabanına zarar veririm korkusu hissetmeden web uygulama güvenlik testi gerçekleştirilebilmektedir.

Sansar aracı bünyesinde ayrıca zaafiyetin test edildiği web sayfası, form ve inputa ait çeşitli verilerin kaydedilmesine olanak sağlayan bir öznitelik çıkarma modülü bulunmaktadır. Söz konusu modül ile, bir önceki paragrafta bahsedilen laboratuvar üzerindeki web uygulamalarında tespit edilen güvenlik açıklarına ait çeşitli öznitelikler kaydedilmiş ve ortaya bir veri seti çıkartılmıştır. Bu tez çalışması kapsamında toplanan verilerden elde edilen veri seti, sahip olduğu öznitelikler, örnek sayısı, zafiyet çeşitliliği gibi konularda yine literatürde bir ilk olma özelliği taşımaktadır.

Bu çalışma kapsamında elde edilen veri seti kullanılarak ilk olarak elde edilen verilere göre sayfa sınıflandırma işlemi yapılmıştır. Gerçekleştirilen sınıflandırma işleminde örnekler, sayfa türlerine göre etiketlendirilmiştir. Etiketleme sonrası gerçekleştirilen sınıflandırma işleminde en yüksek doğruluk oranı Rastgele Orman algoritması ile %96 olarak elde edilmiştir. Sınıflandırma modelinin geliştirilmesinden sonra, tespit edilen zafiyetlerin sayfa türleri ile ilişkilendirilmesi aşamasına geçilmiştir. Bu aşama gerçekleştirilen tez çalışmasının bir diğer orijinal kısmına karşılık gelmektedir. Veri seti üzerinde Apriori algoritması kullanılarak elde edilen birliktelik kuralları, sayfa türleri ile zafiyet türleri arasında bir ilişki kurulmasına olanak sağlamıştır. Geliştirilen bu teknik sayesinde web uygulaması bir sayfayı ziyaret ettiğinde ilk olarak söz konusu sayfanın türü belirlenmekte, daha sonra ilgili sayfa üzerinde ‘öncelikle’ gerçekleştirilmesi gereken testler, birliktelik kurallarına göre gerçekleştirilmektedir.

Bu çalışma kapsamında oluşturulan sayfa sınıflandırma modelinin, geliştirilen araç içerisinde kullanıldığı bir diğer kısım, crawler (örümcek) modülüdür. Günümüzde birçok web uygulama zafiyet tarayıcısı kendisine verilen URL adresinden yola çıkarak web sitesinin haritasını çıkarmaktadır. Fakat kullanıcı girişinin gerektiği durumlarda, başarılı bir giriş işlemi için, çok uzun ve karmaşık ayarlamalar ve parametreler gerekmektedir. Bu durum çoğu zaman, kullanıcı giriş ekranı arkasındaki sayfaların ziyaret edilmesine ve teste tabi tutulmasına engel olmaktadır. Geliştirilen araç, sayfa sınıflandırma işlemi sonrası tespit ettiği kullanıcı giriş sayfası üzerinde, çalıştırma esnasında kendisine verilen kullanıcı adı ve parola bilgilerini kullanarak, başarılı bir

login (giriş) işlemi gerçekleştirmektedir. Bu sayede, admin panelleri gibi zafiyetlerin bulunması muhtemel en kritik alanlar başarılı bir şekilde test edilebilmektedir. Kullanıcı giriş sayfasının sınıflandırma işlemi ile otomatik olarak tespit edilmesi ve giriş yapılması, mevcut kullanılmakta olan ticari ve açık kaynak kodlu zafiyet tarayıcılarda görülmeyen bir işlevdir. Bu özellik gerçekleştirilen çalışmanın literatüre katkılarından biridir.

Geliştirilen Sansar web zafiyet tarayıcısından elde edilen sonuçlar, geliştirilen aracın farklı türde web zafiyetlerini başarı ile tespit ettiği göstermektedir. Geliştirilen test aracının hem sayfa indeksleme, hem de dinamik analiz tabanlı zafiyet testlerinde yapay zekâ tekniklerini kullanması, literatürde bu alandaki boşluğun giderilmesine katkı sağlayacaktır. Ayrıca bu araç ile ulusal bir web zafiyet tarayıcısı eksikliği giderilmeye çalışılmış, bu durum çalışmanın ulusal alanda önemli bir hale gelmesini sağlamıştır.

2. KAYNAK ÖZETLERİ

Bu tez çalışması kapsamında gerçekleştirilen literatür taramasında; statik analiz, dinamik analiz ve sayfa sınıflandırma başlıkları altındaki çeşitli yayınlar incelenmiştir. Literatürde yer alan çalışmalar incelendiğinde web uygulamaları için geliştirilmiş hem statik hem de dinamik analiz metodunu kullanan çeşitli çalışmaların ve tarayıcıların olduğu görülmektedir. Ayrıca sayfa sınıflandırma, form sınıflandırma gibi alanlarda yapılmış çeşitli çalışmalar da bulunmaktadır. Fakat literatürdeki çalışmalar içerisinde web uygulama güvenlik testlerinde, aynı anda hem dinamik analiz, hem sayfa sınıflandırma ve hem de birliktelik analizi tekniklerinin beraber kullanıldığı bir çalışmaya Aralık 2019 itibari ile rastlanmamıştır.

Golub ve Ardö (2005) tarafından yapılan çalışmanın amacı, farklı web sayfası öğelerine (metadata, title, headings ve main text) atanan önem göstergelerinin, otomatik sınıflandırmayı nasıl etkilediğini belirlemektir. Kullanılan veri toplama işlemi, mühendislik alanı ile ilgili 1000 web sayfası üzerinde gerçekleştirilmiştir. Önemli belirteçler, toplam ve kısmi doğruluk, anlamsal mesafe ve çoklu regresyon gibi birkaç farklı metot kullanılarak türetilmiştir. Çalışmada, en iyi sonuçları alabilmek için, tüm unsurların sınıflandırma sürecine dahil edilmesi gerektiği gösterilmiştir.

Huang vd. (2005) çalışmalarında, web uygulamalarında SQLI ve XSS güvenlik açıklarını tespit etmek için WAVES adında bir framework geliştirmişlerdir. Aynı ekip tarafından geliştirilen WebSSARI statik analiz metodu ile çalışırken (Huang vd., 2004), WAVES dinamik analiz metodu ile çalışmaktadır. WAVES istismar edilebilir enjeksiyon noktalarını tespit etmek için uygulamayı taramakta ve tespit ettiği olası istismar edilebilir alanlara saldırı vektörlerini enjekte etmektedir.

Tsipenyuk vd. (2005) gerçekleştirmiş oldukları çalışmalarında, web uygulamalarının güvenliğini tehlikeye atan güvenlik açıklarını 8 kategoriye ayırmıştır. Gerçekleştirilen çalışmada söz konusu kategorilerden yedisi uygulama geliştiriciden kaynaklı kodlama kusurlarına, 1 tanesi ise yanlış yapılandırma ve çevresel nedenlere dayandırılmıştır.

Pietraszek ve Berghe (2005) gerçekleştirdikleri çalışmalarında; enjeksiyon saldırılarının, uygulama seviyesinde güvenlik için büyük bir tehdit oluşturduğunu

ifade etmişlerdir. En yaygın emjeksiyon saldırıları; SQL enjeksiyonu, çapraz betik çalıştırma ve shell enjeksiyon saldırılarıdır. Enjeksiyon saldırılarına karşı savunma için mevcut yöntemler, uygulama geliştiricilerin yetenekleri ile doğru orantılıdır ve bu nedenle de hataya açıktır. Bu çalışmada, enjeksiyon saldırılarını önlemeye yarayan bir metot olan CSSE (Context-Sensitive String Evaluation) tanıtılmıştır. CSSE, bu tür saldırıların neden başarılı olabileceğini, yani kullanıcı tarafından girilen değerin saldırı vektörü olmasının ana nedenini ele alarak çalışmaktadır. CSSE ne uygulama geliştirici etkileşimi ne de uygulama kaynak kodu modifikasyonu gerektirmemektedir. Yalnızca altta yatan platformda değişiklik yapılması gerektiğinden, küçük uygulama geliştiricilerden, büyük çapta uygulama geliştiren ekiplere kadar, enjeksiyon saldırılarına karşı önlem alma yükünü etkili bir şekilde azaltmaktadır. Uygulanan metodun, çoğu enjeksiyon saldırı tipine karşı etkili olup, şu ana kadar önerilen diğer çözümlerden daha az hata eğilimli olduğu gösterilmiştir.

Valenur vd. (2005) gerçekleştirmiş oldukları çalışmalarında, uygulamaya özgü kodun, güvenlik kurallarından habersiz programcılar tarafından sıkı zaman kısıtlamaları altında geliştirildiğine ve bu nedenle birçok zafiyetin oluştuğuna değinmişlerdir. Gerçekleştirilen çalışmada, web tabanlı uygulamalardaki güvenlik açıklarını kullanarak bir veri tabanını tehlikeye atan saldırıların tespiti için bir yaklaşım sunulmaktadır. Sunulan yaklaşım, veri tabanına normal erişim profillerini karakterize etmek için çoklu modeller kullanmaktadır. Önerilen sistem bir saldırıyla ilişkili olabilecek anormal sorguları tanımlayabilmektedir. Önerilen anormallik tespit sisteminin geliştirilmesi için nesne yönelimli bir framework kullanarak geliştirilmiştir. Geliştirilen saldırı tespit sistemi, bilinmeyen saldırıları sınırlı sayıda yanlış pozitif ile tespit edebilmektedir.

Kals vd. (2006) çalışmalarında, Secubat isminde SQL enjeksiyonu ve siteler arası betik çalıştırma zafiyetini test eden, dinamik analiz tabanlı bir web zafiyet tarayıcısı geliştirmişlerdir. Söz konusu çalışmada, ilk olarak ilgili zafiyetlerin yapısı ve istismar edilmesi için gerekli adımlar anlatılmıştır. Daha sonra, zafiyet test işleminin başarılı olup olmadığını anlamak için anahtar kelime ve hata mesajlarına dayalı bir analiz modülü geliştirilmiştir.

Kosuga vd. (2007) gerçekleştirmiş oldukları çalışmalarında, web uygulaması aracılığı ile veri tabanına gönderilen SQL sorgularına müdahale edebilmek amacıyla Sania adında bir araç geliştirmişlerdir. Geliştirilen araç SQL enjeksiyonu saldırılarını tespit etmek için, normal sorgularda ve saldırı esnasındaki sorgularda oluşturulan ayrıştırma (parse) ağaçlarını kullanmaktadır. Gerçekleştirilen çalışmada, ilk olarak SQL sorgularında görünen giriş parametreleri belirlenmekte, daha sonra da olası güvenlik açıklarının belirlenebilmesi için parametreler saldırı satırları ile değiştirilmektedir.

Ru ve Horowitz (2007) çalışmalarında, çoğu e-ticaret web sitesinin, kullanıcı kimlik doğrulaması, yeni kullanıcı kaydı, bülten aboneliği ile ürün ve hizmetlerin aranması için HTML formlarını kullandığına değinmişlerdir. Çalışmada amaç, arama sonuçlarının doğruluğunu iyileştirmek için, Yahoo Shopping and Google's Froogle gibi arama motorları için önemli olan HTML formlarını otomatik sınıflandırmaktır. Söz konusu çalışmada; HTML formlarını özelliklerine göre sınıflandırılması için bir teknik önerilmiştir. HTML formlarının sınıflandırılması için yapay bir sinir ağı algoritması kullanılmıştır. Araştırmacılar, sınıflandırıcılarını, bir e-ticaret veri seti, bir de rastgele elde edilmiş veri seti üzerinde test etmiş ve sırasıyla, %94.7 ve %93.9 oranında başarı sağlamışlardır. Deneysel sonuçlar, sınıflandırıcının etkili, verimli ve umut verici bir metot olduğunu ortaya koymuştur.

Thomas ve Williams (2007) çalışmalarında, algılanan anormalliklerin, saldırılar gerçekleşmeden hemen önce düzeltilmesini amaçlamaktadırlar. Gerçekleştirilen çalışmada, zafiyete neden olan SQL ifadelerini otomatik olarak güvenli SQL ifadeleri ile değiştirmektedirler. Böylelikle tamamlanan bir web uygulaması, olası SQL saldırılarına karşı korunmaktadır. Bu işlemlerin yanında önerilen model, Java uygulamalarındaki güvenlik açıklarını tanımlamakta ve düzeltmektedir.

Wang vd. (2008) çalışmalarında, büyük veri setlerini eğitmek için web sayfalarını sınıflandırmada kullanılacak yeni bir algoritma, CUCS (UC ve SVM birleşimi) sunulmuştur. CUCS, SVM (Support Vector Machine) ve UC'nin (Unsupervised Clustering) avantajlarını birleştirerek yüksek hassasiyet ve hızda çalışma sağlamaktadır. Eğitim aşamasında, CUCS, UC aracılığıyla pozitif örnek merkezleri ve negatif olanları içeren kümeleme merkezlerini alır. Daha sonra CUCS, SVM tarafından sınıflandırıcı üretmek için eğitim seti hazırlar. Sınıflandırma aşamasında,

negatif merkezlerin yanı sıra pozitif merkezlere de web sayfasından minimum uzaklık hesaplanmıştır. İki mesafe arasındaki uzaklık yeterince büyükse, web sayfası UC tarafından sınıflandırılmaktadır. Aksi takdirde, web sayfası, kesilmiş SVM tarafından sınıflandırılmaktadır. Deneylerden elde edilen sonuçlara göre, CUCS, UC'den çok yüksek ve SVM'den biraz daha yüksek olan bir tahmin oranı göstermektedir. Tüketilen zamana göre ise, CUCS, UC'den daha fazla ve SVM'den çok daha az maliyetlidir.

Chen ve Wu (2010) çalışmalarında, web uygulama zafiyetlerini farketme ve önlemenin kolay olmasına rağmen, web uygulama geliştiricilerin farkındalık seviyesinin düşük olması nedeniyle söz konusu zafiyetler ile çok sık karşılaşıldığına değinilmiştir. Gerçekleştirilen çalışmada, web uygulamaları üzerinde SQL enjeksiyonu ve XSS zafiyetlerini tespit etmek için bir zafiyet tarayıcısı geliştirilmiştir. Geliştirilen zafiyet tarayıcısı, bünyesinde söz konusu zafiyetleri barındıran 7 farklı web uygulaması üzerinde test edilmiştir.

Galan vd. (2010) çalışmalarında, XSS zafiyetinin web uygulama güvenliğini en çok tehdit eden zafiyetlerden biri olduğuna değinmişlerdir. Çalışmada; XSS zafiyetinin tespitinde web uygulama zafiyet tarayıcıların kullanımının yaygın olduğu, fakat söz konusu tarayıcıların en fazla bir ya da iki türde XSS tespit edebildiği belirtilmiştir. Gerçekleştirilen çalışmada, web uygulamaları üzerinde XSS zafiyet testi yapan bir zafiyet tarayıcısı geliştirilmiştir. Geliştirilen aracın zafiyet tespitindeki yeterliliği, farklı senaryolar altında değerlendirilmiş ve XSS zafiyet tespitinde başarılı olduğu vurgulanmıştır.

Artunes ve Vieira (2010) çalışmalarında, web uygulama zafiyet tarayıcıların, web servislerini taramada yetersiz kaldıklarına değinmişlerdir. Çalışmada, söz konusu araçların yüksek false-positive orana sahip olduğu belirtilmiştir. İlgili çalışmada, web servisleri üzerinde SQL enjeksiyonu zafiyetini tespit eden bir zafiyet tarayıcısı geliştirilmiştir. Gerçekleştirilen zafiyet tarayıcısının, web servisleri üzerinde yer alan SQL enjeksiyonu zafiyetlerini tespit etmede oldukça başarılı olduğu belirtilmiştir.

Halfand vd. (2009) çalışmalarında, web uygulamaları üzernde yer alan zafiyetlerin tespiti için iki aşamalı bir model geliştirmiştir. Geliştirilen modelde; sayfa üzerindeki potansiyel saldırı noktalarını tespit etmek için statik analiz yöntemi kullanılmaktadır.

Tespit edilen potansiyel saldırı noktalarına testlerin gerçekleştirilmesi ve yanıtların yorumlanması için ise dinamik analiz yöntemi kullanılmaktadır.

Bau vd. (2010) çalışmalarında; siyah kutu tabanlı web uygulama zafiyet tarayıcıları yapısal, işleyiş ve istatistiksel olarak karşılaştırmıştır. Gerçekleştirilen çalışmada, 8 farklı web uygulama zafiyet tarayıcısı, hedef zafiyetleri tespit etmedeki başarı oranları yönünden karşılaştırılmıştır. Gerçekleştirilen test işlemleri sonrasında, karşılaştırma işleminde kullanılan araçların özellikle kalıcı XSS ce SQL enjeksiyonu zafiyetini tespit etmede yetersiz kaldığı görülmüştür.

Ali vd. (2011) çalışmasında, web uygulamalarına yönelik olarak gerçekleştirilen saldırıların sebep olduğu zararın büyüklüğüne değinilmiştir. Söz konusu çalışmada, web uygulamaları üzerinde yer alan SQL enjeksiyonu zafiyetlerinin tespit edilmesi için MySQLlInjector adında bir zafiyet tarayıcısı geliştirilmiştir.

Awad (2012) çalışmasında, web sayfalarının sınıflandırılması probleminde davranış karşılaştırması yapmak için SVM, K-Nearest Neighbour (KNN) ve Generalized Instance Set (GIS) gibi makine öğrenme algoritmaları kullanılmıştır. Ancak, farklı sayıdaki en yakın komşuya sahip neredeyse tüm GIS' ler, KNN'den daha yüksek bir depolama gereksinimine ve çevrimiçi test hesaplama maliyetine sahiptir. Bu durum, depolama gereksinimini ve GIS'in liste test maliyetini azaltmak için gelecekte bazı çalışmaların yapılması gerektiğini göstermektedir.

Katerina ve Goce (2012) gerçekleştirmiş oldukları çalışmalarında, web sistemlerine yönelik zafiyetlerin ve saldırıların sayısının giderek arttığını ve internet üzerinde hâkim olma eğilimi gösterdiğini belirtmişlerdir. Ayrıca, popülerlik ve kullanıcıların içerik oluşturma yeteneğinden dolayı, Web 2.0 uygulamalarının saldırganlar açısından cazip hedefler haline geldiğini belirtmişlerdir. Gerçekleştirilen çalışmada üç farklı makine öğrenme yöntemi kullanılarak kötü amaçlı web etkinliklerinin çok sınıflı sınıflandırılması işlemi gerçekleştirilmiştir. Gerçekleştirilen analiz işlemi web 2.0 uygulamalarını içeren üç katmanlı bir Web sisteminden oluşan yüksek etkileşimli bir honeypot ile dokuz aylık süre içerisinde toplanan verilere dayanmaktadır. Gerçekleştirilen çalışmada elde edilen bulgular, test edilen öğrenme yöntemlerinin

birden fazla savunma sınıfı ve saldırı sınıfı arasında verimli bir şekilde ayrım yapmak için kullanılabileceğini göstermiştir.

Kiezun vd. (2012) gerçekleştirmiş oldukları çalışmalarında, web uygulamaları üzerinde yer alan SQL enjeksiyonu ve XSS zafiyetlerini ortaya çıkaran girdiler oluşturmak için otomatik bir teknik sunmaktadırlar. Önerilen teknik; örnek girdileri üretmekte, zayıflıkları sembolik olarak izlemekte (veritabanına erişimler dahil) ve somut istismar üretmek için girdileri değiştirmektedir. Önerilen teknik gerçek saldırı vektörleri oluşturmakta, uygulama kodunu değiştirmeden dinamik olarak zafiyetleri test etmektedir.

Klassen (2012) gerçekleştirdikleri çalışmalarında, web sayfaları üzerinde sıklıkla karşılaşılan arama formları üzerinde sınıflandırma işlemi gerçekleştirmişlerdir. Söz konusu çalışmada, sınıflandırma işlemi için çeşitli HTML tagları içerisinden text verileri çekilmiştir. Sınıflandırmaya sokulan formlar, arama formu ya da değil olarak 2 farklı sınıfa ayrılmıştır. Gerçekleştirilen çalışmada çok katmanlı yapay sinir ağları sınıflandırıcısı kullanılmış ve %91 başarı oranı elde edilmiştir.

Krutil vd. (2012) çalışmalarında, internetin büyük miktarda bilgi içeren bir kütüphane olduğunu ve içeriğinin web sayfası sınıflamasına göre kategorize edilmesine ihtiyaç duyulduğunu belirtmişlerdir. Web sayfası içeriğinin sınıflandırılmasının, web arama kalitesini ve doğruluğunu artırabileceğini belirtmişlerdir. Ne yazık ki, web sayfalarının yüksek boyutsal veri kümesi olması sınıflandırma sürecini zorlaştırmaktadır. Web sayfası sınıflandırması için otomatik bir yöntem kullanılması, tüm süreci basitleştirebilmekte ve arama motorunun daha ilgili sonuçlar vermesine yardımcı olabilmektedir. Günümüzde, web üzerindeki bilgiler genellikle düzenli ve standart bir formda yapılandırılmamakta ve biçimlendirilmektedir. Bing, Google, Yahoo! ve Yandex içerik arama motorları tarafından, web semantiğini desteklemek ve arama sonuçlarını geliştirmek için Schema.org şemaları koleksiyonu oluşturulmuştur. Bu çalışma, web içeriğinin geniş bir koleksiyonunu sınıflandırmak için kaynak kod yapısının kullanımını incelemiştir. Web sayfalarını sınıflandırmak ve onları açık bir şekilde kategorize etmek için Schema.org şemaları koleksiyonunun kullanımına odaklanılmıştır.

Lee vd. (2012) gerçekleştirmiş oldukları çalışmalarında, SQL enjeksiyonu saldırılarını tespit etmek için statik ve dinamik yaklaşımları birleştirmişlerdir. Gerçekleştirilen çalışmada ilk olarak kaynak kod analiz edilmekte, kullanılan SQL sorgularının yapısı çıkarılmakta ve sorgularda bulunan niteliklerin değerleri kaldırıldıktan sonra depolanmaktadır. Bu işlemden sonra saldırılar, SQL sorgularının söz dizimsel yapılarının, önceden belirlenmiş bir söz dizimiyle karşılaştırılmasından sonra çalışma zamanı sırasında saptanmaktadır. Bu yaklaşımın avantajı, algoritmaların saldırıları sabit zamanda algılama kabiliyetine sahip olmasıdır.

Natarajan ve Subramani (2012) çalışmalarında, Veritabanı odaklı web uygulamalarının güvenliği ve gizliliğinin, saldırılara karşı korunmasının zorluğundan bahsetmişlerdir. Gerçekleştirilen çalışmada, SQL enjeksiyon saldırılarını (SQLIA) tespit etmek ve önlemek için SQL güvenli bir algoritma tasarlanmıştır. Java dilinde yazılan algoritma, çalışma zamanına (runtime) entegre edilmiştir. Söz konusu çalışmada, geliştirilen aracın algoritması ve kodun mantığı sunulmuştur. Söz konusu algoritmanın, farklı türde saldırılara karşı gösterdiği başarı incelenmiştir.

Scholte vd. (2012) çalışmalarında, SQLI ve XSS güvenlik açıklarını tespit edebilmek amacıyla, makine öğrenmesi ve statik analizi birleştiren IPAAS adında bir yazılım geliştirmişlerdir. Gerçekleştirilen çalışmada veri türüne ilişkin kısıtlamalar ve input parametrelerinin değeri, kaynak koddan ve HTTP isteklerinden çıkartılmaktadır. Her iki kaynaktan elde edilen veriler, saldırıları önlemek için çalışma zamanı (runtime) boyunca yürütülebilen, girdi üzerindeki doğrulama politikalarının çıkarılması amacıyla kullanılmaktadır.

Sing ve Roy (2012) çalışmalarında, web uygulamaları üzerinde yer alan SQL enjeksiyonu zafiyetlerini tespit edebilmek için ağ tabanlı bir zafiyet tarayıcısı geliştirmişlerdir. Gerçekleştirilen çalışmada geliştirilen zafiyet tarayıcısı, aynı hedef uygulamalar üzerinde 4 farklı web uygulama zafiyet tarayıcısı ile karşılaştırılmıştır. Gerçekleştirilen karşılaştırma işleminde, geliştirilen aracın, diğer araçlara göre daha az false- positive sonuç ürettiği görülmüştür.

Djuric (2013) çalışmasında, kullanıcı girdilerinin doğrulanmamasından ortaya çıkan zafiyetlere değinmiş, söz konusu zafiyetlerde SQL enjeksiyonu zafiyetinin önemini

vurgulamıştır. Söz konusu çalışmada SQL enjeksiyonu zafiyeti için test işlemi gerçekleştiren bir web uygulama zafiyet tarayıcısı geliştirilmiştir. Geliştirilen araç, test edilen sayfalarda SQL enjeksiyonu varlığını hata mesajlarından değil, HTML sayfa benzerlik oranına göre tespit etmektedir.

Mirdula ve Manivannan (2013) çalışmalarında, web uygulamalarında güvenlik kavramının günümüzde çok önemli bir konu olduğunu belirtmişlerdir. Web uygulamalrının iletişim amaçlı kullanımının, web teknolojisini çevrede önemli bir hale getirdiğini söylemişlerdir. Web uygulamasının ve güvenliğinin önemi gün geçtikçe arttığını, ancak geleneksel ağların web uygulaması için güvenlik sağlayamadığını belirtmişlerdir. Bu çalışmada, web uygulamalarında meydana gelen çevrimiçi zararlı saldırılar ve bu tür saldırıların önlenmesi için sağlanması gereken çözümler tartışılmıştır. Uygulamalı olarak güvenlik açıklarının test edilmesi için SQL enjeksiyonu zafiyeti hedef alınmıştır. Ayrıca çeşitli metodolojiler de tartışılmıştır.

Shar ve Tan (2013) çalışmalrında, girdi kalıpları üzerinde makine öğrenmesi teknikleri kullanarak web uygulamalarında SQLI ve XSS güvenlik açıklarını öngörmek için PHPMinerl adında bir prototip geliştirmişlerdir. Gerçekleştirilen çalışmada SQLI güvenlik açığının tespit edilebilmesi için, kaynak kod analiz edilmekte yani statik analiz işlemi gerçekleştirilmektedir.

Fonseca vd. (2014) tarafından gerçekleştirilen bir çalışmada ise, SQLI ve XSS tabanlı saldırılardan korunmak için, hem C#, Java gibi güçlü diller ile hem de PHP gibi zayıf diller ile uygulama geliştirmede, kaçınılması gereken kodlama hataları vurgulanmaktadır.

Qianqian ve Xiangjun (2014) çalışmalarında, mevcut olarak kullanılmakta olan açık kaynak kodlu web uygulama zafiyet tarayıcıları, test edebildiği zafiyet sayısı ve çalışma türü bakımından karşılaştırmışlardır. Gerçekleştirilen çalışmada daha sonra, komut ekranı üzerinden çalışan açık kaynak kodlu w3af web uygulama zafiyet tarayıcısına bir arayüz geliştirilmiş ve servis olarak sunulmuştur.

Deri vd. (2015) çalışmalarında, web sınıflandırmasının, kullanıcıların geçerli güvenlik ilkesi tarafından izin verilmeyen seçili web sitelerine erişmelerini engellemek, web

aramasını iyileştirmek ve içeriksel reklamcılığı uygulamak için kullanıldığından bahsetmişlerdir. Piyasada birçok ticari web sınıflandırma servisi ve halka açık birkaç web dizini servisinin olduğunu belirtmişlerdir. Çalışmada çoğunlukla İngilizce dilindeki web sitelerine odaklanıldığından dolayı, elde edilen sınıflandırma güvenilirliği ve kapsamı, diğer diller için uygun sonuçlar üretilememesine neden olmaktadır. Bu çalışma, Top Level Domain (TLD)'ler için web tabanlı bir sınıflandırma aracının tasarımını ve uygulanmasını kapsamaktadır. Her domain ve bünyesinde yer alan web sitesi analiz edilerek ve içeriğine göre kategoriler halinde sınıflandırılmaktadır. Araç, sonuçları bu makalede sunulan tüm kayıtlı it uzantılı internet alan adlarının sınıflandırılması işleminde kullanılmış ve başarısı doğrulanmıştır.

Lee vd. (2015) çalışmalarında, www üzerindeki bilgi miktarındaki inanılmaz artış nedeniyle, hızlı bir şekilde yararlı bilgileri almak için verimli bir web sayfası sınıflandırmasına ciddi bir şekilde ihtiyaç duyulduğunu belirtmişlerdir. Bu çalışmada, eğitim veri setindeki her özellik için en iyi ağırlıkları öğrenmek ve test veri setindeki yeni web sayfalarını sınıflandırmada en iyi ağırlıkları almak için yeni bir Simplified Swarm Optimization (SSO) algoritması önerilmiştir. Ayrıca, parametre ayarları SSO'nun güncelleme mekanizmasında önemli bir rol oynadığından dolayı, parametre ayarlarını belirlemek için Taguchi yöntemi kullanılmıştır. Algoritmanın etkinliğini göstermek için, performansı dört veri setine göre iyi bilinen Genetic Algorithm (GA), Bayesian Classifier ve K-Nearest Neighbour (KNN) sınıflandırıcıları ile karşılaştırılmıştır. Deneysel sonuçlar, SSO'nun diğer üç yaklaşımdan daha iyi performans verdiğini göstermiştir.

Makino ve Klyuev (20015), çalışmalarında; web uygulamalarının yaygınlaşması ile, web zafiyet tarayıcıların önemimin arttığına değinmişlerdir. Gerçekleştirilen çalışmada 2 farklı zafiyet tarayıcısı çeşitli kriterlere göre karşılaştırılmıştır. Söz konusu araçlar kullanılarak, zafiyet içeren web uygulamaları üzerinde test işlemleri gerçekleştirilmiş, elde edilen sonuçlar yorumlanmıştır.

Deepa ve Thilagam (2016) gerçekleştirmiş oldukları çalışmalarında, son 10 yılda SQL enjeksiyonu ve XSS saldırıları hakkında gerçekleştirilmiş çalışmaları incelemişlerdir. Gerçekleştirilen çalışmada SQL enjeksiyonu konulu 17, XSS konulu 34 ve mantıksal

kusurları işleyen 34 olmak üzere toplamda 86 makale incelenmiştir. İncelenen makaleler, saldırılara karşı önerilen savunma mekanizmalarının kullanıldığı yazılım geliştirime yaşam döngüsü aşamalarına göre kategorize edilmiştir.

Jimenez (2016) çalışmasında, web uygulamalarında sızma testleri tekniğine odaklanmış, farklı aşamaları tartışmıştır. Web uygulama güvenliği, sızma testi kavramı, sızma testi türlerine değinmiştir. Ayrıca sızma testlerinde izlenecek metodolojileri incelemiş ve detaylandırmıştır. Çalışmada, sızma testlerinde kullanılabilecek otomatik zafiyet tarayıcılar da işlevleri ile birlikte anlatılmıştır.

Woogue vd. (2017) çalışmalarında, internetin binlerce kaynak içeren güçlü bir araç olduğundan bahsetmişlerdir. Web içeriğini daha iyi düzenlemek için bu kaynakları belirli kategorilere göre kategorize etmeye ihtiyaç duyulduğunu belirtmişlerdir. Bu araştırma, önceden tanımlanmış eğitim kategorilerini sınıflandıracak bir yapı oluşturmayı amaçlamaktadır. Bu çalışmada, web sayfalarını eğitim alanlarına göre kategorize edebilecek yedi farklı algoritma üzerinde çalışılmıştır. Web kategorizasyonu ile ilgili hali hazırda birçok çalışma zaten yapılmıştır ancak genel bir kategori setine dayanmaktadır. Bu araştırma öncelikle, eğitim alanlarıyla yakından ilgili olan önceden tanımlanmış bir dizi kategoriye odaklanmıştır. Makine öğrenmesinin kullanılmasıyla, sınıflandırıcı bir web sayfasının neyle ilgili olduğunu analiz edebilecek ve kategorisini belirleyebilecektir. Çalışma ayrıca kullanılan farklı sınıflandırıcıları da karşılaştırmaktadır. Sonuç olarak, sistem belirli bir eğitim alanına bir web sayfası atayabilecek ve öğrenciler tarafından sıkça talep edilen web sayfalarının kategorilerini belirlemek için okullar tarafından kullanılabilecektir. Ayrıca Linear SVM kullanılarak, farklı kategoriler için bir sözlük oluşturma işlemi de gerçekleştirilmiştir. Her kategori için en iyi kelimeler, oluşturulan bu sözlüğü kullanarak belirlenmiştir.

3. MATERYAL VE YÖNTEM

Bu bölümde, gerçekleştirilen tez çalışmasının temelini oluşturan konular ve kullanılan yöntemler hakkında bilgi verilmektedir. Günümüz bilişim dünyasında gerçekleştirilen siber saldırıların büyük çoğunluğunun web uygulamalarını hedef alması, bu tez çalışmasını web uygulama güvenliği alanına yönlendirmiştir. Materyal ve yöntem bölümünün birinci kısmında sızma testleri, ikinci kısmında web güvenliği ve web uygulama zafiyetleri, üçüncü kısmında rastgele orman algoritması ve dördüncü kısımda Apriori algoritması konuları hakkında bilgi verilmiştir.

3.1. Sızma Testleri

Sızma testleri; bilgi güvenliği uzmanları tarafından gerçekleştirilen, siber saldırganların bilişim sistemlerine verebileceği olası zararları önceden tespit etmek, raporlamak ve gerekli savunma önlemleri almak amacı ile gerçekleştirilen saldırı denemelerinin tamamıdır (Burlu, 2010).

Bilgi güvenliğini sağlamada sızma testlerinin rolü, kurumların bilgi sistemlerini daha güvenli seviyeye taşımalarına yardımcı olmaktır. Sızma testlerinde amaç; test edilen sistemler üzerindeki tüm güvenlik zafiyetlerini tespit ederek, olası gerçek bir saldırı öncesinde aksiyon almak ve söz konusu zafiyetleri gidermektir (Yalçınkaya, 2015). Diğer bir ifade ile sızma testleri; kurumsal bilgi güvenliğini ihlal etmek için gerçekleştirilen, yetkilendirilmiş saldırılar bütünüdür (Mohammad ve Pourdavar, 2010).

Siber saldırganlar tarafından bilişim sistemlerine yönelik olarak her geçen gün yeni bir saldırı türü ya da güvenlik açığı tespit edilmektedir. Bu da bilişim sistemlerine yönelik risklerin sürekli bir değişim içerisinde olduğunu göstermektedir. Değişen risklere karşılık olarak sızma testleri de belirli periyotlarla ve farklı senaryolarla tekrarlanmalıdır.

Sızma testleri; sızma testi uzmanı, beyaz şapkalı hacker gibi birçok farklı ünvana sahip uzmanlar tarafından gerçekleştirilmektedir. Güvenlik uzmanları, gerçek bir siber saldırı esnasında saldırganlar tarafından kullanılan yöntem ve tekniklerin aynısını test

esnasında kullanmaktadır. Böylelikle gerçek bir siber saldırının tam kapsamlı tatbikatı yapılmakta, elde edilen test sonuçlarının güvenilirliği artmaktadır. Güvenlik uzmanlarının gerçekleştirdiği sızma testleri, teste tabi tutulan kurumun isteği ve bilgisi doğrultusunda gerçekleştirilmektedir.

Web uygulamaları bir kurumun dünyaya açılan kapısı, tanıtım yüzü olduğu için siber saldırganlar tarafından hedef alınacaklar listesinin ilk sırasında yer almaktadır. Bu nedenle kuruma yönelik olarak gerçekleştirilecek sızma testleri kapsamında test edilmesi gereken bileşenlerin başında web uygulamaları gelmektedir. Web uygulamaları üzerinde belirli periyotlarla gerçekleştirilecek sızma testleri, kurumsal web güvenliğini sağlamada kilit rolü oynamaktadır.

Sızma testleri; kapsamı ve uygulama biçimleri yönünden üçe ayrılmaktadır. Bunlar; Siyah kutu sızma testleri, beyaz kutu sızma testleri ve gri kutu sızma testleridir (Jimenez, 2016).

3.1.1 Siyah Kutu Sızma Testleri

Siyah kutu sızma testlerinde, güvenlik uzmanı test edeceği sistemler ile ilgili hiçbir bilgiye sahip değildir. Güvenlik uzmanı ilk olarak hedef alacağı sistemler hakkında bilgi toplamalıdır. Siyah kutu sızma testinde güvenlik uzmanları, tıpkı bir saldırgan gibi düşünmeli, saldırganın takip edeceği adımları uygulayarak sistemleri hedef almalıdır. Bu nedenle siyah kutu sızma testi diğer sızma testi türlerine göre daha fazla yaratıcılığa ihtiyaç duymaktadır (Şahinaslan, 2012).

Web uygulamalarına yönelik siyah kutu sızma testlerinde, test uzmanına sadece bir URL adresi verilmekte, söz konusu adresten yola çıkarak test işlemlerini gerçekleştirmesi istenmektedir. Söz konusu sızma testi türünde saldırgan, URL adresi üzerinden ya da web uygulaması üzerinde gördüğü girdi alanlarını kullanarak çeşitli saldırı denemeleri yapmakta, gerçekleştirdiği işlemlere karşılık web uygulamasının verdiği yanıtı yorumlamaktadır.

3.1.2. Beyaz Kutu Sızma Testleri

Beyaz kutu sızma testlerinde, test uzmanına teste tabi tutacağı sistemin kendisi ve arka planda hizmet veren tüm teknolojiler hakkında detaylı bilgi verilmektedir. Bu test türünde sızma testi uzmanına test edilecek kuruma ait IP adresleri, aktif sunucular, aktif servisler ve kullanıcı bilgileri gibi birçok veri sağlanmaktadır. Beyaz kutu sızma testlerinin avantajı; güvenlik uzmanlarının söz konusu bilgileri elde etmek için zaman kaybetmemesi, testlerden daha hızlı ve daha güvenilir sonuçlar elde edilebilmesidir (Yalçınkaya, 2015).

Web uygulamalarına yönelik olarak gerçekleştirilen beyaz kutu sızma testlerinde test uzmanına web uygulamasının kaynak kodu da sağlanmaktadır. Böylelikle test uzmanı kaynak kod içerisinde daha önceden tespit edilememiş mantıksal hataları tespit edebilmekte, fazla satırları ayıklayarak ölü kod parçalarını uygulamadan çıkarmaktadır. Bu işlem sayesinde web uygulamaları daha optimize ve güvenli çalışmaktadırlar.

3.1.3. Gri Kutu Sızma Testleri

Gri kutu sızma testleri, siyah kutu ve beyaz kutu sızma test tekniklerinin birlikte kullanılmasıdır. Gri kutu sızma testlerinde; teste tabi tutulan sistemler siyah kutu sızma test teknikleri ile test edilmektedir, fakat test uzmanı, tıpkı beyaz kutu sızma testlerinde olduğu gibi sistemler ve uygulamalar hakkında çeşitli bilgiler ile desteklenmektedir (Xiong ve Peyton, 2010). Bu test türü; kurumun ağ altyapısı bilgilerini içeren belgeleri elde etmiş bir saldırganın dış ağdan gerçekleştirdiği saldırılar olarak tanımlanabilmektedir (Jimenez, 2016).

Web uygulama güvenliği açısından bakıldığında gri kutu sızma testlerinde, test durumları web uygulamasının iç yapısı baz alınarak belirlendikten sonra siyah kutu teknikleri ile gerçekleştirilmektedir. Bu sayede web uygulaması kullanım anında test edilmesinin yanı sıra içyapısı da sınanmış olmaktadır (Sarıman, 2015).

Sızma testlerinde ayrıca "kapsam belirleme, kullanılacak metodolojiyi belirleme ve takip edilmesi gereken aşamalar" konularının da üzerinde durulması gerekmektedir.

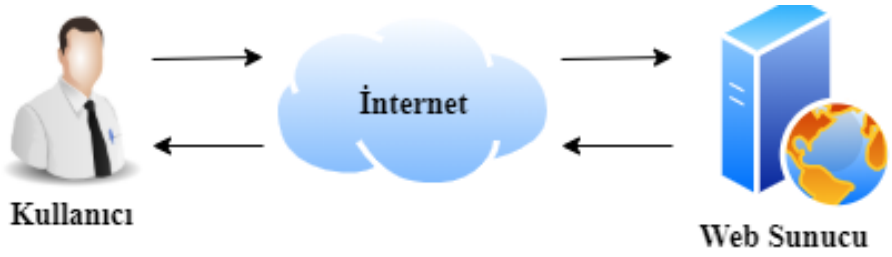
Fakat sözü geçen konular daha çok ağ tabanlı sızma testi kapsamına girdiğinden dolayı bu tez çalışmasında ayrı başlıklar olarak verilmemişlerdir. Söz konusu her bir konuya, web uygulama sızma testleri başlığı altında gerektiği durumlarda değinilmiştir.

3.2. Web Uygulama Güvenliği

90' lı yılların başından beri eposta işlemleri, bankacılık işlemleri, alışveriş işlemleri gibi kullanıcı odaklı web uygulamaları saldırganların hedefi olmuştur (Andreu, 2006). Fakat günümüzde internet kullanımının oldukça yaygınlaşması, neredeyse günlük işlerin tamamının web uygulamaları üzerinden gerçekleştirilebilmesi, söz konusu uygulamaları insan hayatının en merkezi bileşenlerinden biri haline getirmiştir. Ayrıca kurum ve kuruluşlar için web uygulamaları dünyaya açılan bir kapı, bir nevi vitrin haline gelmiştir. Web uygulamalarının hem kurumsal hem de bireysel olarak kullanımının bu denli artması, web güvenliği kavramını günümüzde en önemli konulardan biri haline getirmiştir.

Web uygulamalarının türünde ve sayısındaki artış her geçen gün yeni türde web zafiyetlerinin ortaya çıkmasına neden olmaktadır. Artan zafiyet türleri, web uygulama sızma testlerine olan ihtiyacı da beraberinde arttırmıştır. Web uygulama zafiyetlerine daha iyi anlaşılabilmesi için web uygulamalarının çalışma prensiplerine değinmek gerekmektedir.

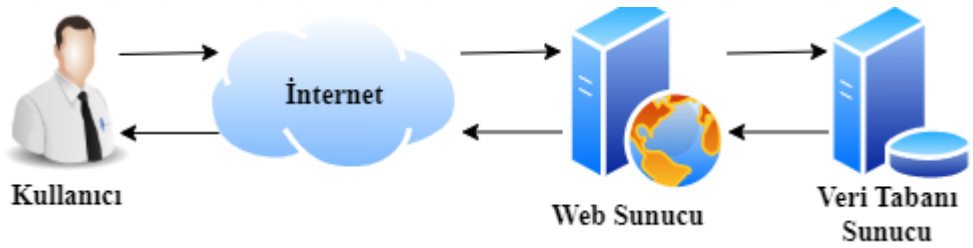
Günümüzde web uygulamaları, statik veya dinamik yapıda çalışan HTML içeriklerinden meydana gelmektedir. Statik yapıda çalışan web uygulamaları; kullanıcıdan aldığı isteklere göre ilgili web sayfalarını göstermektedirler (Vural, 2007). Şekil 3.1' de Statik bir web uygulamasının çalışma prensibi gösterilmektedir.



Şekil 3.1. Statik bir web uygulamasının çalışma prensibi

Statik tabanlı web uygulamalarında, kullanıcıdan alınan girdilere göre veri tabanında herhangi bir sorgu çalıştırılmamaktadır. Söz konusu web uygulamaları genellikle tanıtım, bilgi verme amaçlı, içeriği kullanıcı tercihlerine göre değişmeyen web uygulamalarıdır.

Daha önce bahsedildiği gibi günlük rutin birçok işlemin web uygulamaları üzerinden yapılabilmesi statik web uygulamalarının yerini dinamik web uygulamalarına bırakmasına neden olmuştur. Dinamik tabanlı web uygulamaları; içeriği ve davranışı kullanıcı istekleri doğrultusunda değişen web uygulamalarıdır. Dinamik web uygulamalarında kullanıcılar, Google Chrome, Mozilla Firefox, Safari, Microsoft Edge gibi internet tarayıcılar üzerinden isteklerini web sunucusuna iletmektedirler. Kullanıcıdan gelen talepler web sunucusu tarafından sorgu haline getirilip, veri tabanı sunucusuna gönderilmektedir. Veri tabanı sunucusu kendisine gönderilen sorguyu çalıştırdıktan sonra oluşan yanıtı web sunucusuna geri göndermektedir. Web sunucusu da veri tabanı sunucusundan kendisine gönderilen sorgu yanıtına göre işlem yapmaktadır. Şekil 3.2' de dinamik bir web uygulamasının çalışma prensibi gösterilmektedir.



Şekil 3.2. Dinamik bir web uygulamasının çalışma prensibi

Dinamik web uygulamalarının kullanıcı odaklı olarak esnek çalışması, birçok güvenlik zafiyetini de beraberinde getirmektedir.

Günümüzde web uygulama güvenliğini sağlama amacı ile birçok çalışma gerçekleştirilmektedir. Bunlardan en önemlisi 2001 yılında Mark Curphey tarafından başlatılan OWASP projesidir. OWASP; Open Web Application Security Project kelimelerinin baş harflerinden oluşan bir kısaltmadır. Türkçe' ye "açık kaynak kodlu web uygulama güvenliği projesi" olarak çevrilmektedir. OWASP kapsamında; web uygulama güvenliği alanında standartların belirlenmesi, web güvenlik araçlarının

geliştirilmesi ve yaygınlaştırılması gibi çalışmalar gerçekleştirilmektedir. OWASP tarafından 2004 yılında tam güvenli bir web uygulaması geliştirmek için dikkat edilmesi gereken en kritik 10 web güvenlik açığını yayınlamıştır. Söz konusu tarihten itibaren her yıl bu güvenlik açığı listesi güncellenerek, kullanıcıların tam güvenli web uygulaması geliştirmesi amaçlanmaktadır (Sedek vd., 2009). Bu bölümde web uygulamaları üzerinde yer alan en tehlikeli zafiyetler anlatılmaktadır.

Web uygulamalarındaki güvenlik açıkları; yetkilendirme, şifreleme, bilgi sızıntısı ve girdi doğrulama olmak üzere dört kategori altında toplanmaktadır (Sarıman,2015).

Yetkilendirme açıkları; bir web uygulamasında çeşitli işlevleri gerçekleştirmeden önce ön koşul olarak belirlenmiş kullanıcı yetkilendirme yöntemlerinin istismar edildiği zafiyetlerdir. Yetkilendirme atlatma açıkları sayesinde saldırganlar, erişim yetkisine sahip olmadığı web sayfalarında, bilgi sızdırma, değiştirme, kopyalama ya da silme gibi işlemler gerçekleştirebilmektedir. Günümüzde kullanıcı giriş sayfalarında captcha kullanılması, iki aşamalı kimlik doğrulama kullanılması gibi ek önlemler alınması ve internet kullanıcılarının güçlü parola oluşturma konusunda artık daha bilinçli olması sayesinde, söz konusu açıkların neden olduğu zarar ciddi oranda azalmıştır.

Şifreleme açıkları; web uygulamaları üzerinde işlem gören önemli ve hassas verilerin taşınmasında herhangi bir şifreleme işlemi kullanılmamasından kaynaklanmaktadır. Şifreleme açıklarının neden olduğu saldırılar, iki nokta arasındaki veri iletişimini şifreli kanal üzerinden yaparak güvenli bir iletim olanağı sağlayan SSL sertifikasının kullanılması ile azalmıştır.

Bilgi sızıntısı; saldırganların web uygulamasının kaynak kodlarında yer alan notlar, çeşitli durumlarda oluşan hata mesajları gibi verilerden yararlanarak, web uygulamalarına yönelik saldırılar gerçekleşmesine neden olan güvenlik açığıdır (Auger, 2009). Bilgi sızıntısına verilebilecek en yaygın örnek; kullanıcı giriş sayfalarında oluşan hata mesajlarıdır. Saldırganın rastgele kullanıcı adı ve parola denediği durumlarda eğer ekrana “geçerli bir kullanıcı bulunamadı” hatası basılırsa, saldırgan bu hatayı almayana dek yeni kullanıcı adları deneyecektir. Denediği herhangi bir kullanıcı adında ilgili hatayı almayan saldırgan, sistem üzerinde tanımlı bir kullanıcı adına erişmiş olacaktır (Sarıman, 2015).

Girdi doğrulama açıkları; kullanıcı tarafından web uygulamasına verilen girdi ifadelerinin herhangi bir kontrol ya da denetimden geçmemesinden kaynaklanmaktadır. Web uygulamasına girdiği ifadelerin herhangi bir denetimden geçmediğini gören saldırgan; bir sonraki bölümde açıklanacak olan zafiyetlere yönelik payloadları (zararlı ifadeler) web uygulamasındaki input alanlara girerek sayfanın hata vermesini ya da olağan dışı bir eylem sergilemesini bekleyecektir. Girdi doğrulama açıklarının önüne geçebilmek için kullanıcı tarafından web uygulamasına girilen verilerin işlenmeden önce çeşitli filtreleme ve kontrollerden geçirilmesi gerekmektedir.

3.2.1. Web Uygulama Zafiyetleri

Bu bölümde, saldırganların bir önceki başlıkta belirtilen açıklardan faydalanarak istismar ettiği web uygulama zafiyetleri hakkında bilgi verilecektir.

3.2.1.1. SQL Enjeksiyonu

SQL enjeksiyonu, güvenilir olmayan kaynaklardan gönderilen girdi verilerinin, kontrol edilmeden SQL sorgusu içerisinde kullanılması ve veri tabanı sunucusunda çalıştırılmasından kaynaklanmaktadır (Demir, 2013). SQL enjeksiyon saldırılarında saldırganlar, web uygulamaları üzerinde yer alan input alanlar aracılığı ile veri tabanına gönderilen sorguların yetersiz doğrulama mekanizmasından geçirilmelerini istismar ederek, kötü amaçlı veri tabanı sorguları çalıştırabilmekte, böylelikle de hedef sunucu üzerinde yer alan veri tabanını ele geçirmeye kadar varan birçok işlemi gerçekleştirebilmektedirler. (Kieyzun vd., 2009).

Web uygulamalarında kullanıcı tarafından gönderilen istekler doğrultusunda oluşturulan dinamik sorguların çalıştırılması ile veri tabanından çeşitli değerler döndürülmektedir. Kullanıcı istekleri doğrultusunda geriye döndürülen bu değerler, web uygulamasının tasarımına göre kullanıcıya çeşitli formatlarda sunulmaktadır. Ancak, web uygulamalarında saldırganlar tarafından girdi alanlarına yazılan zararlı ifadeler nedeniyle, dinamik olarak oluşturulan SQL sorguları istismar edilebilmektedir. SQL enjeksiyonunda saldırgan; tarayıcı adres çubuğu, arama alanı,

yorum bırakma alanı gibi veri girişi yapılabilen alanlara zararlı SQL komutları girerek, görüntüleme yetkisinin olmadığı verilere ulaşabilmektedir (Demirol vd., 2013).

Literatürde hata tabanlı SQL enjeksiyonu, union tabanlı SQL enjeksiyonu, kör SQL enjeksiyonu ve zaman tabanlı SQL enjeksiyonu ve http header tabanlı SQL enjeksiyonu olmak üzere 4 farklı SQL enjeksiyonu türü bulunmaktadır.

3.2.1.1.1. Hata Tabanlı (Basit) SQL Enjeksiyonu

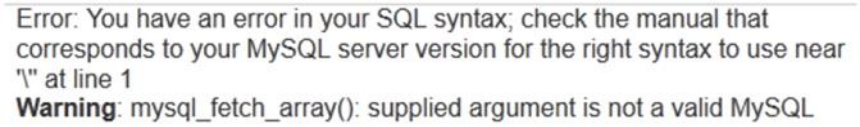
Web uygulamalarında yer alan girdi alanlarına kullanıcılar tarafından girilen veriler ile dinamik SQL cümlecikleri oluşturulmaktadır. Örneğin "SELECT * FROM bolumler" örnek SQL cümleciği basit şekilde veri tabanından bolumler tablosu içerisinde yer alan tüm verileri geriye döndürülecektir. Girdi alanlarından alınan veriler ile dinamik SQL cümlecikleri oluşturulurken, saldırganlar tarafından araya sıkıştırılan bir meta-karakter SQL Injection saldırılarına neden olabilmektedir. SQL dilinde en kritik meta karakter (') tek tırnaktır. Çünkü SQL ifadelerinde iki tek tırnağın arası String tipinde veri olarak algılanmaktadır. Bir web sayfasında yer alan girdi alanlarına, tek tırnak gibi sorgu yapısını bozacak ifadeler girildikten sonra, gelen yanıt içerisinde SQL tabanlı hata mesajları varsa, söz konusu sayfada SQL enjeksiyonu tespit edildiği anlamına gelmektedir.

Bu başlık altında anlatılacak örneklerde, bir önceki paragrafta belirtildiği gibi bir okula ait web sitesi üzerinde testler yapıldığı düşünülecektir. Söz konusu web uygulaması aracılığı ile kullanıcı belirli bir bolümde kayıtlı olan öğrencileri listelemek istemektedir. Bu listeleme işleminde URL ifadesi aşağıdaki şekilde olacaktır.

- http://www.teztest.com/ogrenciler.aspx?bolum=1

bu URL' e erişim isteğinde bulunulduğunda veri tabanında select * from ogrenciler where bolumNo=1 sorgusu çalıştırılacaktır. Belirtilen sorgu sonrasında öğrenciler tablosu içerisinde yer alan ve bolumNo değeri 1 olan tüm öğrenci kayıtları kullanıcıya döndürülecektir. Yukarıda gösterilen URL adresi, SQL enjeksiyonu için test edileceği zaman ilk olarak, URL adresinin sonuna, veri tabanında çalıştırılacak SQL sorgusunun yapısını bozacak bir meta karakter eklenmektedir. Bu karakter çoğu zaman tek tırnaktır. URL adresi sonuna söz konusu karakter eklenip çalıştırıldığında, veri tabanı sorguyu yorumlayamayacak ve ekrana kullanılan veri tabanı yönetim sistemi türüne

göre bir hata mesajı basacaktır. Şekil 3.3' de SQL enjeksiyonu barındıran bir URL adresinin sonuna tek tırnak koyulduğunda elde edilen hata gösterilmektedir.



Şekil 3.3. SQL Enjeksiyonu Zafiyetinin Tespit Edilmesi

Söz konusu hata mesajının yanıt ekranında görülmesi, ilgili sayfada hata tabanlı SQL enjeksiyonu olduğu anlamına gelmektedir. Bu aşamadan sonra saldırganlar, ek bir kontrolden geçmeyen girdi alanlarını ya da değerleri değiştirerek, veri tabanından görüntülemeleri yasak olan verileri görüntülemektedirler. Yukarıda belirtilen URL ifadesi;

- http://www.teztest.com/ogrenciler.aspx?bolum=1' or '1'='1

şeklinde değiştirildiğinde, veri tabanında çalıştırılan SQL sorgusu tabanında select * from ogrenciler where bolumNo=1' or '1'='1' şeklinde değişecektir. Bu SQL sorgusu incelendiğinde where ifadesinden sonraki bolumNo=1' or '1'='1' koşulu her zaman doğru olacaktır. Çünkü söz konusu tablo içerisinde bolumNo değeri 1 olan hiçbir kayıt olmasa bile '1'='1' koşulu her zaman doğru olacağı için ilgili tablo içerisindeki tüm öğrencilere ait kayıtlar geri döndürülecektir. İncelenen örnekteki tekniğin yanı sıra, bir web uygulamasında basit SQL enjeksiyonunun bulunup bulunmadığının tespit edilebilmesi için;

- or 1=1--
- " or 1=1--
- or 1=1--
- ' or 'a'='a
- " or "a"="a
- ') or ('a'='a gibi farklı ifadelerde kullanılmaktadır. Hedef alınan veri tabanı yönetim sisteminin türüne göre kullanılan ifadeler farklılık gösterebilmektedir.

3.2.1.1.2. Union Tabanlı SQL Enjeksiyonu

Hata tabanlı SQL enjeksiyonuna yönelik olarak alınan tedbirler sonrasında saldırganlar farklı yöntemler ile SQL enjeksiyonu saldırıları gerçekleştirmeye

başlamışlardır. Bu yöntemlerden biri de union tabanlı SQL enjeksiyonudur. Bu saldırı tekniğinde SQL dilinin 'Union' (sorgu birleştirme) komutundan yararlanılmaktadır. Söz konusu teknikte, yazılımcının sorgu çalıştırmak için kullandığı fonksiyonunun döndürdüğü verilere ve meydana gelen hatalara bakılmaktadır (Taşçı, 2014).

SQL dilinde Union komutu, birden fazla Result-Set' i birleştirmek için kullanılmaktadır. Union tabanlı SQL enjeksiyonunda dikkat edilmesi gereken nokta, zafiyet barındıran SQL sorgusunun kullanıcıya muhakkak veri tabanından çektiği verileri gösteriyor olmasıdır. Aksi taktirde, Union komutunu SQL sorgusuna eklemenin bir anlamı olmayacaktır. Union sorgularında dikkat edilmesi gereken diğer nokta ise, Union ile birleştirilen sorgular ile iki tablodan çekilen kolonların karşılıklı veri tiplerinin aynı olması gerektiğidir. Bir uygulamada Union tabanlı SQL enjeksiyon açıklığının tespit edilme süreci, basit SQL enjeksiyon zafiyetinin tespit edilmesi ile aynıdır. Eğer gönderilen sorgu sonrasında hata elde ediliyor ve elde edilen söz konusu hata içerisinde kullanıcıya veri tabanından çekilen veri gösteriliyor ise söz konusu sistem Union tabanlı SQL zafiyeti içeriyor olarak yorumlanabilmektedir. Farklı türde veri tabanları üzerinde gerçekleştirilen başarısız UNION denemelerinden alınan hata mesajları Çizelge 3.1' de gösterilmektedir (Demir, 2013).

Çizelge 3.1. VTYS' lerine göre Union sorgu hata mesajları

VTYS	**Hata Mesajı**
MSSQL	All queries combined using a UNION, INTERSECT or EXCEPT operatör must have an equal number of expressions in their target lists.
Oracle	ORA-01789: query block has incorrect number of result columns.
MySQL	The used SELECT statements have a different number of columns.
MS Access	The number of columns in the two selected tables or queries of a union query do not match.

Union tabanlı SQL enjeksiyonun mantığını daha net anlamak adına http://teztest.com/index.asp?id=10 URL adresi üzerinde testlerin yapıldığı varsayılsın.

URL de yer alan 10 değerine ek olarak, veri tabanında yer alan bir başka String tipinde veri Union ifadesi ile getirilecek olursa söz konusu URL şu şekilde bir hal alacaktır;

- http:// teztest.com /index.asp?id=10 UNION SELECT TOP 1 TABLE_NAME FROM INFORMATION_SCHEMA.TABLES—

INFORMATION_SCHEMA.TABLES tablosu, veri tabanı sunucusu üzerinde yer alan tüm tablolar hakkında bilgi içermektedir. Bu tablo içerisinde yer alan TABLE_NAME alanı ise veri tabanında yer alan tüm tabloların isimlerini içermektedir. Bu sorgu çalıştığı anda veri tabanındaki ilk tablo ismi döndürülecektir. Union komutu, ikinci sorgudan dönen string tipindeki (nvarchar) tablo ismini, rakam olan integer 10 değeri ile birleştirmek için, integer' a çevirmeye çalışacaktır. Nvarchar bir değer integer'a çevrilemediği için hata meydana gelecek ve ekrana hata basılacaktır. Oluşacak hata metni aşağıdaki gibi olacaktır;

- Microsoft OLE DB Provider for ODBC Drivers error '80040e07' [Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value **urunler** to a column of data type int. /index.asp, line 5

Elde edilen hata mesajında nvarchar değerin integer' a çevrilemediği görülmektedir. Bunun yanında hata çıktısı incelendiğinde veri tabanı sunucusunda yer alan ilk tablo isminin 'urunler' olduğu görülmektedir. Bir sonraki tablo isminin öğrenilebilmesi için ilgili URL ifadesi **http:// www.teztest.com/index.asp?id=10 UNION SELECT TOP 1 TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME NOT IN ('urunler')--** şeklinde değiştirilebilmektedir. Bu ifade çalıştırıldığı zaman veri tabanı sunucusundan döndürülen hata mesajı ise aşağıda belirtilen şekilde olacaktır;

- Microsoft OLE DB Provider for ODBC Drivers error '80040e07' [Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value '**kullanicilar'** to a column of data type int. /index.asp, line 5

Elde edilen çıktı incelendiğinde urunler tablosundan sonra veri tabanındaki bir diğer tablonun isminin kullanicilar olduğu görülmektedir. 'kullanicilar' tablosu belirlendikten sonra, bu tablo içerisindeki sütun isimleri belirlenebilecektir. Union SQL enjeksiyonu zafiyetinin test işlemleri, tablo isminin belirlenmesinden sonra,

sütun isimlerinin belirlenmesi, sütunlar içerisindeki değerlerin belirlenmesi şeklinde devam edebilmektedir.

3.2.1.1.3. Blind (Kör) SQL Enjeksiyonu

Son zamanlarda geliştirilen web uygulamaları; basit SQL enjeksiyonu ve union tabanlı SQL enjeksiyonu saldırılarını engellemek için, hata mesajlarını ayrıntılı olarak kullanıcıya sunmak yerine, özel olarak oluşturulmuş bir hata mesajını veya hata sayfasını kullanıcılara göstermektedir. Bu durumda hata mesajlarına göre SQL enjeksiyon zafiyetinin tespit edilmesi mümkün olmamaktadır. Bu gelişmeler üzerine saldırganlar tarafından keşfedilmiş olan enjeksiyon tekniği, blind SQL enjeksiyonudur.

Blind SQL enjeksiyonunda, saldırıya maruz kalan web uygulamasından hata mesajı alınamayacağı için, ilgili sayfanın/servisin manüplasyonlara verdiği cevapların true ya da false olması ile ilgilenilmektedir (Demir, 2013).

Blind SQL enjeksiyonunu daha iyi açıklayabilmek için, kullanıcı profillerinin listelendiği http://www.teztest.com/kullanicilar.php?kid=2 gibi bir URL üzerinde testlerin yapıldığı varsayılacaktır. Söz konusu URL içerisinde yer alan kid parametresinde kör SQL enjeksiyonunun varlığını test etmek için; URL üzerinde yapılacak değişikliklere göre gelecek tepkilerin true ya da false olması durumuna bakılacaktır. Söz konusu URL **http://www.teztest.com/kullanicilar.php?kid=2 AND 1=1** şeklinde değiştirilip çalıştırıldığında yine aynı kullanıcıya ait profil görüntüleniyor ise (Doğru- True) kid parametresinde blind SQL enjeksiyonu olduğu düşünülebilmektedir. Söz konusu parametrenin blind SQL enjeksiyon zafiyeti barındırdığını kanıtlamak için bu kez de URL ifadesi **http://www. teztest.com/kullanicilar.php?kid=2 AND 1=2** şeklinde değiştirilip çalıştırılır. Eğer uygulama ekranına aynı kullanıcıya ait bilgiler görüntülenemez ise kid parametresinin SQL enjeksiyonu zafiyetini kesin olarak içerdiği anlamına gelmektedir.

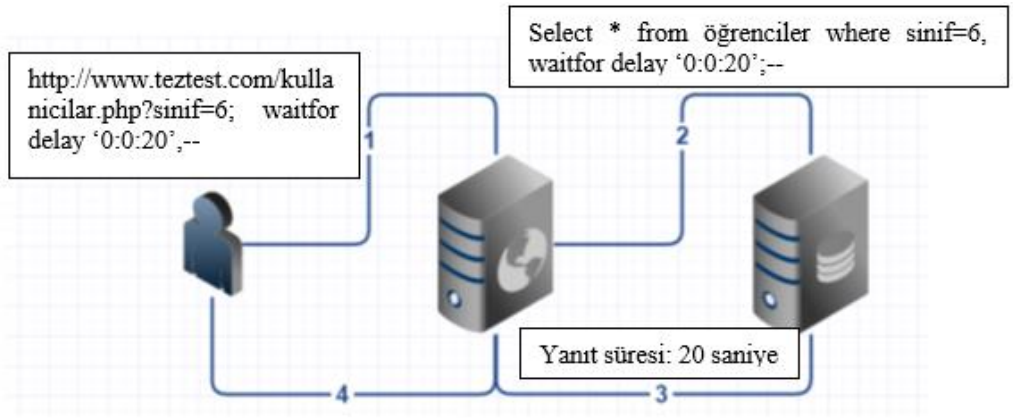
Blind SQL enjeksiyonu zafiyeti belirlendikten sonra, bu zafiyetin istismar edilmesi işlemi, diğer SQL enjeksiyon zafiyetlerine göre daha zaman alıcı fakat aynı zamanda otomatize etmeye daha müsaittir. Testlerin gerçekleştirildiği örnekte, kid

parametresinin blind SQL enjeksiyon zafiyeti içerdiği bir önceki paragrafta tespit edilmişti. Bir sonraki aşamada veri tabanında tanımlı tablonun ismi çekilmeye çalışılacaktır. Bu işlem için MySQL veri tabanı yönetim sistemlerinde, veri tabanının adının öğrenilmesini sağlayan select database() komutu kullanılacaktır. Blind SQL enjeksiyonu boolean mantığına dayandığı için sorgulamaya yapılacak ek parçalama ve eşitleme işlemleri ile boolean bir sonuç elde edilmeye çalışılacaktır. İlgili URL ifadesi, **http://www. teztest.com/kullanicilar.php?kid=2 AND ASCII (SUBSTRING((SELECT database()),1,1))=97** şeklinde değiştirilmiştir. Değiştirilen URL içerisinde yer alan sorgu incelendiğinde en iç parantez içerisinde yer alan select database ifadesi ile veri tabanı ismi çekilmiştir. Bir dış parantez içerisinde yer alan substring komutu, çekilen veritabanı isminin ilk harfinin alınması için kullanılmıştır. Ascii komutu ile elde edilen harfin ascii karşılığı bulunmuştur. Veri tabanı isminin ilk harfinin bulunmasından sonra, söz konusu değer teker teker harflerin ascii değerleri ile karşılaştırılacaktır. Yanlış karşılaştırmalarda false değeri geriye döneceği için web uygulaması üzerinde kullanıcıya ait hiçbir veri görüntülenemeyecektir. Doğru ascii değer karşılaştırması yapıldığında true geriye dönecek ve kullanıcıya ait bilgiler web uygulamasında görüntülenebilecektir. Bu yöntem ile veri tabanı isminin ilk harfi çekildikten sonra substring komutu ile diğer harflerde tek tek kontrol edilecek ve tüm işlem sonunda veri tabanının adı elde edilebilecektir (Demir 2013).

3.2.1.1.4. Zaman Tabanlı SQL Enjeksiyonu

Bu çalışmada önceki başlıklarda incelenmiş olan SQL enjeksiyonu zafiyetlerinde; gönderilen sorgu ifadeleri sonucunda alınan hata mesajları, geriye döndürülen verilerin değişmesi ya da uygulamanın sorgu sonunda true/false değerlerine göre farklılık göstermesi gibi durumlara göre farklı tiplerde SQL enjeksiyon zafiyeti tespit edilmiş ve istismar yolları incelenmiştir. Fakat günümüzde bazı web uygulamaları söz konusu davranışları sergilemese dahi SQL enjeksiyonu zafiyetini içerebilmektedirler. Zaman tabanlı SQL enjeksiyonu zafiyetinde, yapılan isteğe karşılık olarak sunucudan dönen cevabın içeriği (hata dönmesi ya da true/false değere göre doğru ya da yanlış değer dönmesi gibi) yerine, cevabın dönüş süresine bakılarak zafiyetin varlığı tespit edilebilmektedir.

Zaman tabanlı SQL enjeksiyon zafiyetinin tespit edilmesi ve istismar edilmesi için, veri tabanlarının zaman temalı fonksiyonları kullanılmaktadır. Örneğin MSSQL veri tabanları için WAITFOR DELAY komutu ilgili veri tabanında belirtilen süre kadar bekleme yaptırmak için kullanılmaktadır. Şekil 3.4' de zaman tabanlı SQL enjeksiyonunun çalışma mantığı gösterilmektedir.



Şekil 3.4. Zaman tabanlı SQL enjeksiyonu akış şeması

Şekil 3.4' de gösterilmekte olan zaman tabanlı SQL enjeksiyonu akış şeması incelendiğinde 1. Adımda kullanıcı URL alanına **http://www. teztest.com/kullanicilar.php?sinif=6'; WAITFOR DELAY '0:0:20';--** ifadesini yazarak çalıştırmaktadır. Söz konusu ifade içinde veri tabanından sinif değişkeni değeri 6 olan kayıtlara ait verilerin getirilmesi istenmektedir. Aynı zamanda bu ifadeye eklenen WAITFOR DELAY komutu ile geriye ilgili kayıtlar döndürülmeden önce 20 saniyelik bir bekleme yapılması istenmiştir. 2.adımda Web sunucu tarafından veri tabanı sunucusuna gönderilen ifade, **SELECT * from ogrenciler WHERE sinif='6'; WAITFOR DELAY '0:0:20';--** şeklinde bir dinamik SQL sorgusuna çevrilerek veri tabanı sunucusu üzerinde çalıştırılmıştır. İlgili SQL sorgusunun sonunda bulunan – ifadesi, kendisinden sonra eğer bir SQL komutu var ise bunların görmezden gelinmesi için kullanılmıştır. Çalıştırılan SQL sorgusu sonrasında, veri tabanında yer alan 6. Sınıf öğrencilerine ait kayıtlar geriye 20 saniye gecikmeli olarak gösterilecektir. İlgili işlem gerçekleştikten sonra saldırgan göndermiş olduğu isteğin 20 saniye gecikmeli olarak geldiğini gördüğünde, bu URL üzerinde zaman tabanlı SQL enjeksiyonu zafiyetinin tespit etmiş olacaktır.

Zaman tabanlı SQL enjeksiyonunun istismarı, blind SQL enjeksiyonunun istismarına çok benzemektedir. Blind SQL enjeksiyonunda gönderilen sorgulara karşılık true ya da false olmasına bakılırken, zaman tabanlı SQL enjesiyonunda gönderilen sorguya karşılık olarak dönen cevabın dönüş süresinde gecikme olup olmadığına bakılmaktadır. İlgili örnekte, kid parametresinin zaman tabanlı SQL enjeksiyon zafiyeti içerdiği bir önceki paragrafta tespit edilmişti. Bir sonraki aşamada söz konusu örnek aracılığı ile veri tabanı kullanıcısının ismi çekilmeye çalışılacaktır. Bu işlem için MSSQL veri tabanı üzerinde tanımlı komutlardan SELECT USER komutu kullanılacaktır.

İlgili örneğe bağlı olarak zaman tabanlı SQL enjeksiyonu zafiyetinin istismar edilebilmesi için için saldırgan tarafından URL ifadesinin, **http://www.teztest.com/kullanicilar.php?sinif=6'; SELECT 1 IF(ASCII(SUBSTRING((SELECT user),1,1)))= 97 WAITFOR DELAY '0:0:20';--** şeklinde değiştirilmesi gerekmektedir. İlgili URL ifadesi incelendiği zaman söz konusu ifade içerisinde 2 adet SQL sorgusunun bulunduğu görülecektir. Veri tabanında çalıştırılacak ilk sorguda sinif değerleri 6 alan kullanıcılara ait verilerin görüntülenmesi istenmiştir. İkinci sorguda ise veri tabanı kullanıcı adının ilk harfinin ascii kodunun 97 olup olmadığı kontrol edilmiştir. Tam bu noktada ilgili URL adresinin zaman tabanlı SQL enjeksiyonu zafiyeti barındırıp barındırmadığı tespit edilmektedir. Çünkü eğer kullanıcı adının ilk harfi ascii olarak 97 değerine eşit ise, 6. Sınıf öğrencilerine ait bilgiler 20 saniye gecikmeli olarak döndürülecektir. Eğer ilgili harfin ascii karakteri 97 değerine eşit değil ise, öğrencilere ait veriler hemen bekleme olmadan döndürülecektir. Bu şekilde 20 saniye bekleme gerçekleşene kadar tüm ascii değerler sırasi ile denendiğinde kullanıcı adına ait ilk harf verisi listelenebilecektir. İlk harf verisi elde edildikten sonra, kullanıcı adına ait diğer harflerin tespit edilebilmesi için aynı yöntem kullanılacaktır. Diğer harflerin kontrol edilebilmesi için sorgu içerisinde yer alan substring komutuna ait parametrelerin değiştirilerek, ismin kaçıncı harfinin tahmin edileceğinin belirtilmesi gerekmektedir.

Zaman tabanlı SQL enjeksiyonu tekniğinde, verilerin çekilmesi işlemi, çekilecek verinin uzunluğuna göre az veya çok zaman alabilmektedir. Söz konusu işlemin otomatize bir şekilde gerçekleştirilebilmesi için veri tabanından çekilecek verinin uzunluğunun elde edilmesi gerekmektedir. Çekilecek verinin uzunluğunun tespit

edilebilmesi için MSSQL veri tabanı komutlarından LEN() komutu kullanılmaktadır. LEN() komutu ile veri tabanından çekilecek verinin uzunluğunun bulunabilmesi için ilgili URL ifadesi **http://www. teztest.com/kullanicilar.php?sinif=6'; SELECT 1 IF(LEN(SELECT user)=1) WAITFOR DELAY '0:0:20';--** şeklinde değiştirilmelidir. Söz konusu ifade incelendiğinde, veri tabanı kullanıcı adının uzunluğu eğer 1 karakter ise, web uygulaması 20 saniye bekledikten sonra 6. Sınıf öğrencilerine ait bilgileri ekrana getirecektir. İlgili öğrencilere ait bilgilerin herhangi bir bekleme olmadan ekrana getirilmesi durumunda, veri tabanı kullanıcı adının 1 karakter olmadığı anlamına gelecektir. Bu şekilde yapılacak deneme işlemi ile istenen verinin uzunluğu elde edilebilecek, uzunluk verisi elde edildikten sonra, kullanıcı adının elde edilebilmesi için gerekli olan deneme sayısı hesaplanabilecektir.

3.2.1.1.5. HTTP Header tabanlı SQL Enjeksiyonu

HTTP (Hyper Text Transfer Protokol- Hiper Metin Transfer Protokolü) internet üzerinde web sayfalarının görüntülenmesini sağlayan bir protokoldür. HTTP protokolü ile istemci- sunucu arasındaki veri alışveriş kuralları tanımlanmaktadır. Bir kullanıcı internet üzerinde bir web sayfasına ulaşmak istediğinde, kullanıcının internet tarayıcısından web sunucusuna istekleri http protokolü aracılığı ile gönderilmektedir. Sunucu kendisine gelen erişim isteğine yine HTTP protokolünü kullanarak yanıt vermektedir. Söz konusu yanıt içerisinde header ve body bölümleri bulunmaktadır. Yanıtın header (üst bilgi) kısmında, sunucudan tarayıcıya gönderilen yanıta ait; boyut, son değiştirme tarihi, kullanılan şifreleme (encoding) algoritması, isteğe ait metod türü gibi bilgiler yer almaktadır (Başeğmez, 2016). Aşağıda örnek bir http isteğine ait header bilgisi verilmiştir.

- GET / HTTP/1.1
  Connection: Keep-Alive
  Accept:*/* Accept-Language: en-us
  Accept-Encoding: gzip, deflate
  User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.16) Gecko/20110319 Firefox/3.6.16 (.NET CLR 3.5.30729; .NET4.0E)
  Cookie: guest_id=v1%3A0123456789; pid=v1%3A0123456789
  Referer: https://developer.mozilla.org/en-US/docs/Web/JavaScript

HTTP header içerisindeki alanlardan biri olan User-Agent; istemci tarafından kullanılan internet tarayıcı hakkında bilgi vermektedir. Bu bilgi istatistiksel verilerin tutulması ve olası protokol ihlallerinin belirlenmesi için kullanılmaktadır. Referrer ise bir web uygulamasını ziyaret eden ziyaretçilerin nereden geldiğine (arama motoru aracılığı, banner reklamlar aracılığı vb.) dair çeşitli bilgileri sunmaktadır. HTTP header içerisinde bir de cookie alanı yer almaktadır. Cookie alanı bir web uygulamasında açılan oturumun korunması için kullanılmaktadır. Web sunucular kendilerine istek atan web tarayıcıları birbirinden ayırt etmek için birer session (oturum) açarlar. Sunucu, tarayıcıya özel üretilmiş bu session' ın id değerini, HTTP yanıtının cookie bölümüne ekleyerek tarayıcıya gönderir. Bu işlemden sonra internet tarayıcı, web sunucusuna göndereceği tüm isteklerde bu cookie değerini kullanır. Böylelikle web uygulaması üzerinde bir oturum açıldığında bu oturum hep korunacak, kullanıcı farklı safyaları ziyaret ederken devamlı oturum açmasına gerek kalmayacaktır (Şengün,2018).

Günümüzde, oturum tanımlama için HTTP cookie bilgisinin sunucular tarafından veri tabanında depolanması, HTTP headerını saldırganlar için önemli bir saldırı vektörü haline getirmektedir. Saldırganlar web sunucuya http isteği yollarken, isteğin headerında yer alan User-Agent, Referer ve Cookie alanlarını birer input alanı gibi düşünerek payloadları ilgili alanlara gömmekte ve isteklerini bu şekilde göndermektedir. Web sunucu eğer http header içerisinde yer alan payloadı herhangi bir filtreleme ve kontrolden geçirmeden veri tabanına gönderiyor ise, ortaya SQL enjeksiyonu zafiyeti çıkmaktadır. Aşağıda cookie alanına payload yerleştirilmiş bir HTTP headerı gösterilmektedir.

- GET / HTTP/1.1
  Connection: Keep-Alive
  Accept:*/* Accept-Language: en-us
  Accept-Encoding: gzip, deflate
  User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.16) Gecko/20110319 Firefox/3.6.16 (.NET CLR 3.5.30729; .NET4.0E)
  Cookie: **test' or 1/***
  Referer: https://developer.mozilla.org/en-US/docs/Web/JavaScript

3.2.1.2. Siteler Arası Betik Çalıştırma (XSS) Zafiyeti

Siteler arası betik çalıştırma zafiyetleri; saldırganlardan alınan girdinin herhangi bir denetimden geçirilmeden, kullanıcılara gösterilecek sayfanın kaynak kodunun içerisinde yerleştirilmesi ve bu işlem sonunda kullanıcı tarafında betik çalıştırılması işlemidir (Özel ve Eken, 2014).

Siteler arası kod çalıştırma saldırılarında, kullanıcı ile web sitesi arasındaki güven ilişkisi istismar edilmektedir. Söz konusu saldırılar, saldırganların belirlenen çalıştırılabilir kodu kullanıcıya göndermesi ve bu kodun kullanıcıya ait web tarayıcısında çalıştırılmasıyla gerçekleştirilmektedir. XSS kodları genellikle HTML ya da JavaScript dilinde yazılmaktadır. Fakat VBScript, ActiveX, Java, Flash veya web tarayıcılar tarafından desteklenen diger dillerde de kodlama gerçekleştirilebilmektedir (Vural, 2007).

XSS saldırıları web uygulamaları için en tehlikeli saldırıların başında gelmektedir. Programlama dilinden bağımsız olması nedeniyle farklı programlama dilleri ile gerçekleştirilen birçok web uygulamasında ortaya çıkabilmektedir. XSS saldırıları, saldırganların zararlı bir takım kodları web uygulamasına dâhil etmesi ile gerçekleştirilmektedir. XSS saldırılarında saldırganlar temelde 2 farklı yöntem kullanmaktadır. Kullanılan 1'inci yöntemde sayfada bulunan ve kullanıcıdan veri girişi isteyen girdi (input) alanları hedef alınmaktadır. Buna örnek olarak arama kutusu, yorum bırakma alanları gösterilebilmektedir. Saldırganların kullandığı ikinci yöntemde ise URL üzerinden XSS kodları girilmektedir (Çelik, 2011).

Literatürde 4 çeşit siteler arası betik çalıştırma (XSS) saldırısı bulunmaktadır. Saldırganlar tarafından kullanılan XSS saldırı türleri; reflected (yansıtılmış), stored (kalıcı), DOM tabanlı ve HTTP header tabanlı XSS' dir.

3.2.1.2.1. Yansıtılmış (Reflected) XSS

Yansıtılmış XSS saldırılarında URL alanları ya da veri girişinin yapıldığı input alanları hedef alınmaktadır. Bu saldırı, sadece XSS saldırısı yapan saldırgan tarafından görülmektedir. Web sitesinde bulunan ziyaretçiler bu saldırıyı göremezler. Bu yüzden

URL alanı üzerinde gerçekleştirilmiş yansıtılmış XSS saldırılarında saldırgan, URL adresini e-mail gibi çeşitli iletişim kanalları ile yaymaya çalışmaktadır. Saldırgan, kurban olarak belirlediği kullanıcının söz konusu linke tıklaması için çeşitli sosyal mühendislik yöntemlerini kullanmaktadır. Kullanıcı söz konusu linke tıkladığında zararlı kodlar, web sayfası içerisine eklenerek kullanıcıya geri gönderilmektedir. Bu işlemden sonra kullanıcının web tarayıcısı söz konusu zararlı kodları çalıştırılmakta ve saldırı başarı ile tamamlanmaktadır.

Yansıtılmış XSS saldırısının daha iyi anlaşılabilmesi için aşağıdaki örnek incelenmiştir. Aşağıda PHP dili ile geliştirilmiş bir web uygulamasının arama modülüne ait küçük bir kısım gösterilmektedir;

- ```
  <?php
  $gelen = $_REQUEST[‘gelen’];
  echo $gelen;
  ?>
  ```

Yukarıda gösterilen kaynak kodlar incelendiğinde kullanıcı tarafından girilen değerler $gelen isimli değişkene atanmakta ve bu değer doğrudan ekrana yazdırılmaktadır. Kullanıcı tarafından arama kontrolüne girilen ifade arama sonucunda dönen sayfada “aradığınız kelime:” şeklinde gösterilmektedir. Bu işlemden şu sonuç çıkarılabilmektedir; kullanıcı tarafından yapılan girişler hem veri tabanına sorgu olarak gönderilmekte, hem de ekrana yazdırılmaktadır. Doğal olarak SQL Injection ve XSS zafiyetlerinin bulunmasının en olası olduğu noktalar, bu tarz kullanıcı tarafından alınan girdilerin kontrol edilmeden kullanıldığı noktalardır. Gerçekleştirilen arama sonrasında oluşan URL ifadesinin;

- http://www. teztest.com/arama.php?gelen=ogrenciler

şeklinde olduğunu varsayalım. Burada gelen değişkenine atanan öğrenciler ifadesi “mali” ifadesi ile değiştirildiğinde;

- http://www. teztest.com/arama.php?gelen=’mali’

şeklini alacaktır. Bu URL çalıştırıldıktan sonra mali ifadesi eğer web ekranına basılıyorsa, burada XSS zafiyetinin olabileceği anlamına gelmektedir. Söz konusu sayfada XSS zafiyetinin olabileceğinin belirlenmesinden sonra saldırganlar söz konusu alana zararlı javascript kodları ekleyerek saldırıyı gerçekleştirmektedir.
  ```

Aşağıda söz konusu URL ifadesi içerisine zararlı javascript komutlarının gömülmesi işlemi gösterilmektedir.

- http://www.teztest.com/arama.php?gelen=<script>alert('XSS')</script>

Saldırganlar ilk olarak üstteki tarzda gerçekleştirdikleri bir değiştirme ile kaynak kodlarda tek tırnak, büyüktür, küçüktür gibi işaretlerin filtrelenip filtrelenmediğini kontrol etmektedirler. Eğer uygulamada söz konusu karakterlere yönelik bir filtreleme yoksa gerçekleştirilen işlem sonunda Şekil 3.5' te gösterildiği gibi bir küçük pencere içinde XSS yazısı gösterilecektir.

Şekil 3.5. Yansıtılmış XSS testi

Şekil 3.5' de gösterildiği gibi bir pencere ile karşılaşan saldırganlar, hedef almış oldukları URL üzerinde farklı türde parametreler kullanarak hedef kullanıcının cookie bilgisini çalmak, zararlı bir dosyayı indirmek gibi eylemler gerçekleştirerek saldırıyı daha ciddi boyutlara taşımaktadırlar.

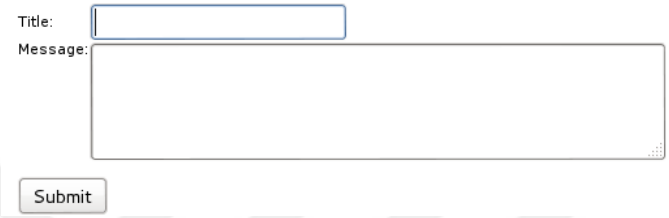
Yansıtılmış XSS saldırılarında saldırganlar, oluşturmuş oldukları payload içeren zararlı URL adreslerini, çeşitli sosyal mühendislik yöntemleri ile kurbanlarının ziyaret etmesini sağlamaktadır. Saldırganlar tarafından kullanılan en yaygın yöntem; "kredi kartı ekstreniz", "telefon faturanız","banka aidat geri ödemesi" gibi başlıklar içeren, kullanıcının merak, korku gibi duygularını istismar etmek amacı ile hazırlanan mailler içerisine zararlı URL adresinin gömülmesidir. Kendisine gönderilen mail içerisindeki zararlı URL adresini ziyaret eden kurbanlar, ilgili saldırıya maruz kalmaktadırlar.

3.2.1.2.2. Kalıcı (Stored) XSS

Saldırganlar tarafından yaygın olarak kullanılmakta olan bir diğer XSS çeşidi ise kalıcı (stored) XSS saldırılarıdır. Bu saldırı türünde daha çok forumlar, yorum alanları ve

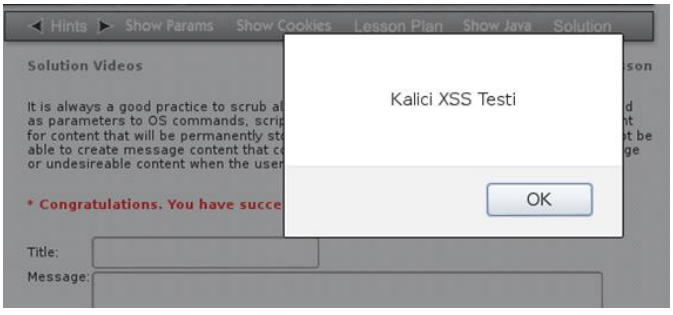
ziyaretçi defteri gibi alanlar hedef alınmaktadır. Bu alanlara saldırganlar tarafından gömülen zararlı XSS kodları veritabanına kaydedilir. Daha sonra zararlı kodların gömüldüğü sayfayı ziyaret eden her kullanıcı sayfayı görüntülediği anda XSS saldırısına maruz kalmaktadır.

Kalıcı XSS saldırısına ait test işlemleri OWASP tarafından geliştirilen WebGoat uygulaması üzerinde gerçekleştirilmiştir. Gerçekleştirilen uygulamaya ait ekran görüntüsü Şekil 3.6' da gösterilmektedir.



Şekil 3.6. WebGoat kalıcı XSS test senaryosu

WebGoat üzerinde yer alan kalıcı XSS senaryosunda günümüzde birçok web uygulamasında yaygın olarak kullanılan yorum bırakma modülü yer almaktadır. İlgili modülün senaryosunda yorum alanı kalıcı XSS zafiyeti barındırmaktadır. Söz konusu senaryoda kalıcı XSS saldırısı gerçekleştirmek için, mesaj alanına "<script> alert("Kalıcı XSS Testi")</script>" komutu eklenmiştir. Bu komutta yer alan alert("Kalıcı XSS Testi") ifadesi, mesajın bırakıldığı sayfayı ziyaret eden tüm kullanıcıların tarayıcıları tarafından yorumlanacak ve "Kalıcı XSS Testi" mesajını içeren küçük bir pencere açılacaktır (Yalçınkaya, 2015). Şekil 3.7' de bir kullanıcının, ilgili senaryodaki kalıcı XSS saldırısı ile istismar edilmiş ve içerisinde zararlı kodların bulunduğu web sayfasını ziyaret etmesi ile karşılaştığı ekran görüntüsü gösterilmektedir.



Şekil 3.7. Başarılı olmuş kalıcı XSS testi

Farklı türdeki XSS saldırıları ile saldırganlar birçok zararlı eylem gerçekleştirebilmektedirler. Örneğin; hedef web sitesinde bulunan XSS zafiyeti kullanılarak, siteyi ziyaret eden kullanıcıların oturum bilgileri (cookie) çalınabilmektedir. Ya da bir takım sosyal mühendislik saldırıları ile kurbanlar kandırılarak bilgisayarına virüs, trojan gibi zararlı yazılımlar yüklenebilmektedir. Bu zararlı yazılımlar aracılığı ile ziyaretçilerin kimlik bilgileri, şifreleri saldırganlar tarafından ele geçirilebilmektedir. Ayrıca saldırganlar söz konusu zararlı yazılımlar aracılığı ile kurbanların bilgisayarlarını bir zombi bilgisayar haline getirebilmektedir. Saldırganlar Dom tabanlı XSS saldırıları ile hedef aldıkları web uygulamasına index atabilmektedir. Sunucuya dosyalar (Shell, Backdoor gibi) yüklenebilmekte, bu sayede kredi kartı, v.s gibi bilgiler elde edilebilmektedir (Çelik, 2011).

3.2.1.2.3. DOM Tabanlı XSS

DOM (Document Object Model) "Belge Nesnesi Modeli" anlamına gelen internet tarayıcılar tarafından kullanılan bir nesnedir. İnternet tarayıcılar, ziyaret edilen her web sayfasını birer belge olarak kabul etmektedirler. Söz konusu web sayfalarında bulunan tüm bileşenler (input, resim, buton vb.) birer nesne olarak kabul edilmektedir. DOM nesnesi; ziyaret edilen web sayfalarında yer alan bu nesnelere müdahale edilmesini sağlamaktadır (Çıtak, 2016).

DOM tabanlı XSS zafiyeti bir HTML dosyasında değil, DOM (Document Object Model) nesnesinde ortaya çıkan zafiyettir. Daha önceki başlıklar atında incelenmiş olan yansıtılmış ve kalıcı XSS zafiyetlerinde, saldırı sonrasında ortaya çıkan sonuçları görüntülemek mümkünken, DOM tabanlı XSS zafiyetinde bu mümkün değildir. DOM tabanlı XSS saldırılarında Javascript kodları saldırgan tarafından manüpile edilmektedir. Söz konusu DOM tabanlı XSS zafiyetlerine yönelik saldırılar sonrasında sayfadan dönen yanıt ve HTML dosyası tamamen aynı olacaktır.

3.2.1.2.4. HTTP Header tabanlı XSS

HTTP header tabanlı XSS zafiyeti; "3.2.1.1.5. HTTP Header tabanlı SQL Enjeksiyonu" başlığında anlatılan SQL enjeksiyonu zafiyeti ile benzer özellikler

taşımaktadır. İlgili başlıkta da anlatıldığı gibi zararlı zafiyet payloadları; HTTP header'ı içerisinde yer alan User-Agent, Referer ve Cookie alanlarına enjente edilmektedir. HTTP header içerisindeki alanlara XSS payloadlarının enjekte edilmesinden sonra, oluşan yeni header kullanılarak web sayfasına istek gönderilmektedir. Test işleminde daha sonra, gönderilen isteğe dönen yanıt sayfası içerisinde payload değeri aranmaktadır. Bu bölümde incelenen diğer XSS türlerinde olduğu gibi enjekte edilen payload sayfa üzerinde başarılı bir şekilde çalışırsa, ilgili sayfada HTTP header tabanlı XSS zafiyeti vardır denilebilmektedir.

3.2.1.3. HTML Enjeksiyonu Zafiyeti

HTML, Hypertext Markup Language web sayfalarında kullanılan işaretleme dilidir. Web geliştiriciler tarafından yazılan HTML kodları, web tarayıcılar tarafından yorumlanarak web sayfalarının görsel hali oluşturulmaktadır. Web uygulama geliştiricilerin kodlama esnasında gerekli dikkat ve özeni göstermemelerinden dolayı web uygulamaları HTML enjeksiyonu zafiyeti içerebilmektedir. Bu zafiyette, saldırganlar sayfanın HTML kodlarına müdahale edip kendi yazmış oldukları payloadları sayfanın kaynak kodları içerisine yerleştirmektedir. Saldırgan tarafından sayfa içerisine enjekte edilen HTML kodları ile bir web sayfasının bütün içeriği değiştirilebilmekte, sayfayı ziyaret eden kullanıcılar zararlı web sayfasına yönlendirebilmektedirler (Weinberger, 2011). HTML enjeksiyonu zafiyetini engellemek için SQL enjeksiyonu ve XSS zafiyetlerinde olduğu gibi kullanıcılar tarafından alınan girdilerin gerekli filtreleme işlemlerine tabi tutulması gerekmektedir. HTML enjeksiyonu saldırılarında saldırganlar, zararlı HTML payloadlarını GET ve POST metodları üzerinden enjekte edebilmektedir. Saldırganlar GET üzerinden gerçekleştirdikleri saldırılarda URL adresi içerisinde yer alan değişkenlere değer olarak HTML payloadlarını eklemektedir. POST üzerinden yapılan saldırılarda ise, web uygulaması üzerinde yer alan ve kullanıcılar tarafından girdilerin yapıldığı input alanları hedef alınmaktadır. GET üzerinden yapılan saldırılara örnek olarak **http://teztest.com?sayfa=2** URL adresinin hedef alındığını varsayalım. **<h1>Tıkla</h1>** HTML payloadı incelendiğinde saldırgana ait bir web uygulamasına link verildiği görülmektedir. URL adresi içerisinde yer alan sayfa değişkenine bu payload atandığında; URL adresinin son hali; **http://teztest.com?sayfa=<a href=**

"http://www.saldirgansite.com\"><h1>Tıkla</h1> şeklini almaktadır. Eğer test edilen sayfa HTML enjeksiyonu zafiyeti barındırıyorsa, sayfa üzerinde Tıkla yazısı ile bir link oluşacak ve bu bağlantıya tıklayanları saldırgan sayfaya yönlendirecektir.

3.2.1.4. Bash Komut Enjeksiyonu (Shellshock) Zafiyeti

Shellshock zafiyeti, Unix tabanlı tüm işletim sistemleri tarafından kullanılan, BASH (Bourne-Again Shell) dilinden kaynaklanan, saldırganın hedef sisteme uzaktan erişim sağlayarak kod enjekte etmesine ve çalıştırmasına olanak sağlayan bir zafiyettir (Çetin vd. 2015).

Bash; bilgisayarlarda kullanıcılar tarafından girilen komutların yorumlanmasına ve çalıştırılmasına olanak sağlayan bir komut dili yorumlayıcısıdır. Özellikle kullanıcıların işlerini kolaylaştırmak ve tekrarlanan görevleri gerçekleştirmek için sıklıkla kullanılmaktadır. Bash script (Bash dili) sayesinde kullanıcı komutları; işletim sisteminin daha sonra çalıştıracağı basit bir metin tabanlı pencereye yazılabilmektedir. Bash, aynı zamanda web sayfaları tarafından iletilen komutların çalıştırılması için de kullanılabilmektedir (Yavuz, 2019). Shellshock zafiyetinin altında bu özellik yatmaktadır. Bash scriptte kullanılan değişken türlerinden biri çevresel değişkenlerdir. Bash içerisindeki güvenlik açığı, söz konusu çevresel değişkenler dolayısıyla ortaya çıkmaktadır. Saldırganlar sunucu BASH kabuğundaki çevresel değişkenlere zararlı kod ejekte ederek Shelshock saldırısını gerçekleştirmektedirler. Söz konusu değişkenlere zararlı kod enjekte etme işleminde HTTP headerları kullanılmaktadır. HTTP isteklerinin header alanlarına zararlı BASH payloadları ekleyen saldırganlar, uzak sunucuda kod çalıştırmaktan, denetim kazanmaya kadar birçok işlemi gerçekleştirebilmektedirler (Mary, 2015).

Shellshock saldırılarında saldırganlar zararlı kod enjekte etmek için sunucuya gönderecekleri HTTP GET ve POST isteklerinin User Agent ve Referer headerlarını kullanmaktadırlar. Örneğin saldırganın sunucu üzerindeki etc/passwd dosyasına erişmek istediği bir saldırı senaryosunda;

- () {:;}; /bin/cat /etc/passwd

payloadı User-Agent veya Referer alanlarına enjekte edilip istek sunucuya gönderilmektedir. Eğer uzak sunucu Shelshock zafiyeti barındırıyorsa sunucu üzerindeki etc/passwd dosyası okunacak ve dönen sayfa içerisine dosya içeriği de eklenecektir.

3.2.1.5. LDAP Enjeksiyonu Zafiyeti

LDAP (Lightweiht Directory Access Protocol), ilk defa 1980' lerin sonlarında ISO ve ITU-T tarafından yayımlanmış olan, dizin servislerinin sorgulama, değiştirme gibi işlemlerinin yerine getirilmesini sağlayan bir protokoldür (Sarıman, 2015). LDAP protokolü sayesinde şirketlerde personel kayıtları, adres ve telefon defteri gibi birçok farklı hizmet tek merkezden sunulabilmektedir. LDAP sunucu kullanan web uygulamaları, kullanıcılar tarafından girilen veriler doğrultusunda dinamik LDAP sorguları oluşturmakta, LDAP sunucusundan dönen sorgu yanıtını kullanıcılara göstermektedir.

Kullanıcı girdileri doğrultusunda oluşturulan dinamik sorgular, söz konusu girdilerin filtrelenmemesi durumlarında saldırıya açık konuma gelmektedir. Örnek üzerinden gitmek gerekirse; **http://teztest.com/ldapserver?pName=mali** adresinde LDAP sunucuya personel adı mali olan kullanıcıya ait kayıtların geri döndürülmesi amaçlanmaktadır. Belirtilen URL adresi ziyaret edildiğinde oluşan LDAP sorgusu aşağıdaki gibidir;

- <LDAP://ldapserver>;
  (personalName=Mali);pNo,phone,department

Yukarıda gösterilen sorgu LDAP sunucusuna iletilmekte ve sunucudan dönen yanıtlar kullanıcıya gösterilmektedir. Söz konusu web uygulamasının LDAP enjeksiyonu zafiyeti barındırıp barındırmadığını anlamanın en temel yollarından bir tanesi; URL adresinde yer alan değişkene LDAP sorgusunun yapısını bozacak payloadlar gömmektir. Örneğin yukarıda örnek olarak gösterilen web uygulamasının LDAP enjeksiyonu zafiyeti içerdiğini varsayılsın. İlgili URL adresi içerisinde yer alan pName değişkenine çok sayıda parantez içeren ")))" değeri atanırsa, bu LDAP sorgusunun yapısını bozacaktır. URL adresinin son hali **http://teztest.com/ldapserver?pName=))))))** aldığında, LDAP sonucusuna

gönderilen sorgu ifadesinde fazladan kapatma parantezi bulunduğundan, sunucu hata mesajı üretecek ve geriye döndürecektir. Sorgu sonrasında görüntülenen sayfada "supplied argument is not a valid ldap" ya da "javax\\.naming\\.NameNotFoundException" gibi hata mesajları görüntülenecektir.

3.2.1.6. PHP Kod Enjeksiyonu Zafiyeti

Kod enjeksiyonu zafiyeti, daha sonra uygulama tarafından yorumlanarak yürütülecek kodların enjekte edilmesine olanak sağlayan zafiyetdir. Diğer zafiyet türlerinde olduğu gibi bu zafiyet de güvenilmez verilerin herhangi bir kontrolden geçmeden işlenmesinden kaynaklanmaktadır.

Kod enjeksiyonu zafiyeti komut enjeksiyonu zafiyetinden farklıdır. Çünkü kod enjeksiyonu zafiteyinden yararlanmak isteyen bir saldırgan, enjekte ettiği kodun dilinin fonsiyonelliği ile sınırlandırılmıştır. PHP kod enjeksiyonu zafiyeti de saldırganların hedef aldığı web uygulamaları ve sunucuları üzerinde, izinsiz ve zararlı PHP kodu çalıştırmalarına neden olan açıklıklardır. Bir saldırganın hedef aldığı web uygulamasına PHP kodu enjekte etmesi durumunda, hedef sistem üzerinde yapabileceği işlemler PHP dilinin işlevleri ile sınırlıdır. PHP kod enjeksiyonuna örnek olarak aşağıdaki PHP kodları verilebilir;

- ```
  $myvar = "varname";
  $x = $_GET['arg'];
  eval("\$myvar = \$x;");
  ```

Yukarıdaki örnek kod bloğunda PHP eval() fonksiyonunun tehlikeli bir kullanımı görülmektedir. Söz konusu kodlarda herhangi bir input doğrulama işlemi gerçekleştirilmemesinden dolayı, PHP kod enjeksiyonu bulunmaktadır. Saldırgan tarafından sayfaya ait URL adresi /index.php?arg=1; phpinfo() olarak değiştirildiğinde **phpinfo()** komutu eval fonksiyonu tarafından yorumlanacak ve derlenecektir. Bu işlem sonunda saldırgan hedef sunucu üzerinde çalışmakta olan PHP versiyonu ve farklı birçok bilgiye ulaşabilecektir. Hedef aldığı sistem üzerinde kendisi tarafından enjekte edilen PHP kodlarının denetlenmeden çalıştırıldığını gören saldırgan, PHP dilinin işlevleri dahilinde hedef sistem üzerinde işlemler gerçekleştirebilecektir. Ayrıca, PHP kod enjeksiyonu zafiyeti aracılığı ile hedef sistem üzerinde çalışmakta
  ```

olan PHP versiyonunu öğrenen saldırganlar, ilgili PHP versiyonu için geliştirilmiş bir exploit olup olmadığını araştırmakta, eğer varsa söz konusu exploiti kullanarak farklı türde saldırılar da gerçekleştirebilmektedir (OWASP, 2013).

3.2.1.7. İşletim Sistemi Komut Enjeksiyonu

İşletim sistemi komut enjeksiyonu (OS Command Injection) zafiyeti; saldırganların web uygulamasını barındıran sunucu üzerinde çeşitli işletim sistemi komutları çalıştırmasına olanak sağlayan, uygulamayı ve verileri çok büyük bir tehlikeye sokan zafiyettir.

Komut enjeksiyonu; bir web uygulamasının kullanıcı tarafından sağlanan form, çerez (cookie), http başlıkları gibi verilerin herhangi bir denetimden geçirilmeden sunucu sistem shelli (kabuğu) üzerinde çalıştırılması ile ortaya çıkmaktadır. Bu saldırıda saldırgan tarafından gönderilen sistem komutları, zafiyeti barındıran web uygulamasının yetkisi (hakları) ile çalıştırılmaktadır. Daha önce anlatılan zafiyetlerde olduğu gibi komut enjeksiyonu zafiyeti de çoğunlukla yetersiz giriş doğrulama nedeni ile ortaya çıkmaktadır.

Komut enjeksiyonu zafiyetini, kod enjeksiyonu zafiyeti ile karıştırmamak gerekmektedir. Kod enjeksiyonunda saldırgan tarafından enjekte edilen zararlı kodlar web uygulaması tarafından çalıştırılmaktadır. Komut enjeksiyonunda ise, saldırgan tarafından gönderilen zararlı komutlar zafiyeti barındıran web uygulaması aracılığı ile hedef sistem shellinde çalıştırılmaktadır (OWASP,2018). Komut enjeksiyonuna örnek olarak, aşağıdaki URL adresinde ilgili zafiyetin bulunduğu varsayılsın;

- http://www.teztest.com/listele.php?liste=teknoloji.txt

Yukarıda verilmiş URL adresinde liste değişkenine atanan teknoloji.txt değeri, işletim sistemi tarafından cat komutu ile açılarak içeriği ekrana yazılmaktadır. Söz konusu sistemde komut enjeksiyonu zafiyetinin yer aldığını düşünen bir saldırgan URL adresini aşağıdaki gibi değiştirmektedir.

- http://www.teztest.com/listele.php?liste=teknoloji.txt; ls

Kullanıcı tarafından gönderilen değerler herhangi bir kontrol ve doğrulama mekanizmasından geçmediğinden, ilk olark teknoloji.txt, daha sonra da ilgili text dosyasının olduğu dizindeki diğer dosyaların listesi geriye döndürülecektir. Dönen yanıt içerisinde farklı dosya isimlerinin listelendiğini gören saldırgan artık çeşitli işletim sistemi komutları ile hedef sunucu üzerinde istediği her işlemi rahatlıkla gerçekleştirebilecektir.

3.2.1.8. Dosya Dahil Etme (File Inclusion) Zafiyeti

Dosya dahil etme (bu aşamadan sonra file inclusion olarak tanımlanacaktır); bir saldırgan tarafından hedef web uygulamasına dosya eklenmesine ya da hedef web uygulamasının bünyesinde barındırdığı fakat kullanıcıya sunmadığı dosyaların izinsiz görüntülenmesine olanak sağlayan zafiyettir. File inclusion zafiyeti local (yerel) file inclusion ve remote (uzak) file inclusion olmak üzere ikiye ayrılmaktadır.

3.2.1.8.1. Yerel Dosya Dahil Etme (Local File Inclusion) Zafiyeti

Local file inclusion zafiyetinde saldırganlar, web uygulamasında sunulmamış fakat web sunucu üzerinde yer alan dosyaların site üzerinden görüntülenmesine neden olmaktadır. Local file inclusion zafiyeti PHP dilinde ortaya çıkan bir zafiyettir. Söz konusı zafiyetin oluşma sebebi, web uygulaması geliştirilirken tanımlanan değişkenlere değer atama hataları yani değerlerin filtrelemeden geçirilmesidir. Local file injection zafiyeti PHP programlama dilinde yer alan “include, include_once, require, require_once” fonksiyonlarının yanlış kullanımından kaynaklanmaktadır. Söz konusu komutlar daha önce yazılmış bir PHP dosyasının, bir başka dosya içerisine çağırılmasını, içerisindeki değişken ve metodların kullanılmasını sağlamaktadır (Akcay, 2019). Local file inclusion zafiyetinin anlatımında aşağıdaki URL adresi kullanılacaktır.

- http://www.teztest.com/index.php?page=contactus.php

Yukarıdaki URL adresi çalıştığında ilk olarak index.php dosyası içerisinde aşağıdaki kod bloğu çalıştırılacaktır.

- <?php
  include($_GET[‘page’])?>

Bir önceki sayfada gösterilen kod bloğu incelendiğinde include fonksiyonu içerisine page değişkenindeki değer parametre olarak verilmektedir. Include fonksiyonu da kendisine gönderilen contactus.php sayfasının görüntülenmesini sağlamaktadır. Include fonksiyonuna verilen değerin herhangi bir filtreleme işleminden geçirilmemesi ilgili değişkene atanan isimdeki dosyaların sunucudan çekilerek web sayfasına dahil edilmesine ve görüntülenmesine neden olacaktır. İlgili URL adresi saldırgan tarafından;

- http://www.teztest.com/index.php?page= ./../../../../../../../../etc/passwd

şeklinde değiştirildiğinde, web sunucu üzerinde etc dizini altında yer alan passwd dosyasının içeriği web sayfasında görüntülenecektir.

3.2.1.8.2. Uzak Dosya Dahil Etme (Remote File Inclusion) Zafiyeti

Remote file inclusion zafiyetinde saldırganlar; web uygulamasına, uzaktaki bir web sunucusunda yer alan bir dosyayı ekleyerek web sayfası üzerinde görüntülemekte hatta zararlı kod çalıştırmaktadırlar. Remote file inclusion zafiyetinin daha iyi anlaşılabilmesi için anlatımında aşağıdaki URL adresi kullanılacaktır.

- http://www.teztest.com/index.php?page=info.php

Yukarıdaki URL adresi ziyaret edildiğinde index.php içerisinde page değişkenine atanan değer çağırılarak çalıştırılmaktadır. Bir önceki başlıkta olduğu gibi include fonksiyonuna atanan değerin herhangi bir filtrelemeden geçmediği durumlarda söz konusu zafiyet ortaya çıkmaktadır. Remote file inclusion zafiyetinden faydalanmak isteyen saldırganlar yukarıdaki URL adresini aşağıdaki şekilde değiştirmektedirler.

- http://www.teztest.com/index.php?page=http://tests.arachni-scanner.com/rfi.md5.txt

Yukarıda verilen URL adresindeki page değişkenine atanan http://tests.arachni-scanner.com/rfi.md5.txt adresi içerisinde sadece 705cd559b16e6946826207c2199bd890 verisi görüntülenmektedir. Saldırgan, yukarıda değişiklik yapılmış URL adresini ziyaret ettiğinde sayfa içeriğinde 705cd559b16e6946826207c2199bd890 verisi varsa, remote file inclusion zafiyetinin varlığını ispatlamış olacaktır. Zafiyet varlığından emin olan saldırgan uzak sunucuda kendisine ait zararlı bir dosyayı web sunucusu üzerinde çalıştırarak dosya düzenleme,

veri tabanına bağlanma, dosya indirme, dosya silme gibi birçok işlemi gerçekleştirebilecektir.

3.2.1.9. XPATH Enjeksiyonu Zafiyeti

XPath; XML dökümanları içerisindeki verilere ulaşmak için kullanılan bir sorgulama dilidir. Web uygulamalarında kullanıcı tarafından girilen veriler doğrultusunda ya da uygulamanın yapısı gereğince XML dosyalarından veri çekmek için XPath sorguları oluşturulmaktadır. Kullanıcı tarafından girilen veriler kullanılarak dinamik sorguların oluşturulması durumunda, eğer kullanıcı girdileri herhangi bir denetimden geçmiyorsa XPath enjeksiyonu ortaya çıkmaktadır. XPath enjeksiyonu temel olarak SQL enjeksiyonu ile benzerlik gösterse de, zafiyetin istismar adımı SQL enjeksiyonundan farklıdır. XPath enjeksiyonu ile izinsiz veri elde edebilmek için hedef alınan XML dosyası elemanlarının isimlerinin bilinmesi gerekmektedir (Sarıman, 2015).

3.2.1.10. SSI (Server Side Includes) Enjeksiyonu Zafiyeti

SSI (Server Side Include); web uygulamalarında yer alan statik sayfalara, dinamik içeriklerin eklenmesine imkan sağlayan betik yazma dilidir. Daha detaylandırmak gerekirse; SSI kullanılarak web uygulamalarına CGI veya çeşitli dinamik uygulamalar dahil edilerek, sunucu taraflı çalıştırılabilmektedir.

SSI enjeksiyonu zafiyeti, HTML sayfalarına komut ejekte ederek ya da uzaktan kod çalıştırarak saldırganların web uygulamasını istismar etmelerine olanak sağlamaktadır. Web uygulamalarında yer alan ve kullanıcıdan girdi eklenmesine olanak veren sayfalarda, girdilerin yeterli denetim ve filtreden geçirilmemesi durumunda söz konusu zafiyet ortaya çıkmaktadır (Karro, 1998).

Örneğin, bir web uygulamasında yer alan arama inputuna girilen verilerin herhangi bir kontrolden geçmeden işlendiği varsayılsın. Bu durumda saldırgan arama inputuna “ <!--#exec cmd="/etc/passwd" --> “ zararlı kod bloğunu girerse söz konusu payload web sunucu tarafından çalıştırılacak ve etc dizini altında yer alan passwd dosyası içeriği saldırgan tarafından elde edilecektir.

3.2.1.10. Sunucu Taraflı Template (Server Side Template) Enjeksiyonu Zafiyeti

Sunucu taraflı template enjeksiyonu (SSTI) zafiyeti; kullanıcı tarafından girilen verilerin doğrudan sayfa şablonuna eklenmesi durumlarında ortaya çıkmaktadır. Tema (Şablon) motorları; web uygulamaları tarafından, web sayfaları ya da epostalar ile dinamik veri sunmak için yaygın olarak kullanılmaktadır. Kullanıcı girdilerini şablonlara denetimden geçirmeden gömmek SSTI zafiyetinin oluşmasına neden olmaktadır. XSS' den farklı olarak SSTI zafiyeti, tespit edildiği her web uygulamasını potansiyel pivot noktasına çevirerek, web sunucusuna doğrudan saldırmak ve uzaktan kod çalıştırmak için kullanılmaktadır. SSTI zafiyeti; web uygulamasının geliştirilmesi aşamasında yapılan hatalarda ya da kullanıcılarına daha zengin işlevsellik sunmak amacıyla şablonların dikkatsiz kullanımı sonucunda ortaya çıkmaktadır.

SSTI zafiyetinin ortaya çıkmasına örnek olarak Twig tema motoru örnek gösterilebilir. Söz konusu tema motoru, kullanıcılar tarafından girilen verileri, dinamik olarak sayfa ya da eposta gibi bileşenlere eklemek için kullanılmaktadır. Örneğin, kullanıcı tarafından girilen isme göre dinamik eposta içeriğinin hazırlandığı bir uygulama olduğu varsayılsın. Kullanıcı tarafından input alanına "Mehmet Ali" ifadesi girildiğinde eposta içeriği "Merhaba Mehmet Ali" olarak başlamaktadır. Eğer kullanıcı tarafından girilen input değeri aşağıdaki kod bloğunda olduğu gibi tema içine ekleniyorsa herhangi bir zafiyet oluşmayacaktır.

- $output = $twig->render("Dear {first_name},", array("first_name" => $user.first_name));

Fakat kullanıcılar tarafından girdilerin özelleştirilmesine izin verilmesi durumunda SSTI zafiyeti ortaya çıkacaktır. Aşağıdaki kod bloğu SSTI zafiyetini barındırmaktadır.

- $output = $twig->render($_GET['custom_email'], array("first_name" => $user.first_name));

Yukarıdaki kod örneğinde kullanıcı şablonun içeriğini custom_email GET parametresi ile kontrol etmektedir. Kullanıcı verilerinin herhangi bir filtrelemeden geçmemesinden dolayı input alanına {{7*7}} ifadesi girildiğinde bu ifade tema motoru tarafından derlenecek ve 49 sonucu sayfa içeriğine eklenecektir. Sayfa içeriğine matematiksel işlemin sonucunun eklendiğini gören saldırgan bu aşamadan sonra tespit ettiği SSTI

zafiyetini komut çalıştırma ya da yerel dosya ekleme gibi saldırılarda kullanabilmektedir (Kettle,2019).

3.2.1.11. Dizin Gezinimi Zafiyeti

Güvenli bir web uygulaması ve hizmeti sağlamak için web içeriğine erişimi doğru bir şekilde kontrol etmek çok önemlidir. Dizin gezinimi (Directory Traversal, Path Traversal) zafiyeti saldırganların web sunucu üzerinde görüntülenmesi yasaklanmış dizinlere erişmesine ve uzaktan komut çalıştırmasına neden olmaktadır. Web sunucularında güvenlik mekanizması olarak kullanılan tekniklerden bir tanesi erişim kontrol listeleridir. (Access Control Lists-ACLs). Erişim kontrol listeleri yetkilendirme işlemlerinde kullanılmaktadır. Bu listeler hangi web sunucusunun yöneticisinin, hangi kullanıcılara ya da gruplara erişebileceğini, değiştirebileceğini göstermektedir. Bunun yanında, web sunucu üzerinde hangi dosyaların kimler tarafından çalıştırılabileceğini belirtmek için de kullanılmaktadır. Örneğin Windows' taki IIS sunucusunun kök dizini C:\\Inetpub\wwwroot 'dur. Normal şartlarda kullanıcılar C:\\Windows dizinine ve diğer alt dizinlerine kısıtlamalardan dolayı erişememektedirler. Web uygulamalarında kullanıcılar tarafından girilen verilerin gerekli denetimden geçirilmemesi durumunda saldıganlar, çeşitli ifade hileleri ile görüntüleme yetkileri dışındaki dizin ve dosyalara erişebilmektedirler. Dizin gezinimi saldırısı gerçekleştirmek için saldırganların bilmesi gereken şey, sistem üzerinde varsayılan dizin ve dosyaların nerede bulunduğudur. Dizin gezinimi zafiyetini daha iyi anlayabilmek adına aşağıdaki örnek incelenebilir. (Acunetix, 2019).

- http://www.teztest.com/goster.asp?page=uyeler.html

Yukarıdaki URL adresi ziyaret edildiğinde internet tarayıcı sunucuya page parametresine uyeler.html değerini atayarak göndermekte ve goster.asp dinamik sayfasını geriye döndürmesini istemektedir. Bu istek web sunucu üzerinde çalıştığında, goster.asp dosyası, web sunucunun dosya sisteminden uyeler.html dosyasını getirir ve kullanıcının görüntülemesi için internet tarayıcıya geri gönderir. Yukarıdaki URL adresinin saldırganlar tarafından aşağıdaki şekilde değiştirilmesi sonrasında, eğer page değişkenine atanan değer herhangi bir filtrelemeden geçmiyorsa dizin gezinimi zafiyeti tespit edilmiş olmaktadır.

- http://www.teztest.com/goster.asp?page=../../../../Windows/system.ini

Bu durum goster.asp dinamik sayfasının web sunucuda yer alan system.ini dosyasını geri döndürmesine ve içeriğini saldırgana sunmasına neden olacaktır. Page değişkenine atanan payload içerisinde yer alan ../ ifadesi sistem tarafından bir dizin yukarı git olarak algılanmaktadır. Yeteri kadar üst dizine çıktıktan sonra saldırgan istediği dosyayı görüntülemeyi başaracaktır.

Dizin gezinimi zafiyetini hedef alan saldırılara karşı alınabilecek önlemlere örnek olarak; kullanıcıdan alınan girdinin uzunluk kontrolüne girmesi, adres çözümleme işleminden geçirilmesi, uzantısının kontrol edilmesi gösterilebilmektedir (Sarıman, 2015).

3.2.1.12. CRLF Enjeksiyonu Zafiyeti

Bir kullanıcı internet tarayıcısı aracılığı ile web sunucusuna bir istekte bulunduğunda, sunucu yanıt olarak HTTP headerları ve istekte bulunan sayfanın içeriğini birlikte göndermektedir. Sunucu tarafından gönderilen paket içerisindeki header ve sayfa içeriği birbirinden bir takım özel karakterler aracılığı ile ayrılmaktadır. Header ve HTML içeriğini birbirinden ayıran bu karakterler carriage return (satır başı) ve line feed (yeni satır) karakterleridir. CRFL kısaltması da söz konusu karakterlerin ilk harflerinden oluşmaktadır.

Sunucu, yeni bir header başladığında daha öncekinin carriage return ve line feed ile bittiğini bilmektedir. CRLF enjeksiyonu saldırısı gerçekleştiren bir saldırgan, sunucuyu bir HTTP nesnesinin sonlandığı, diğerinin başladığı şeklinde kandırmak için input alanına carriage return, line feed ya da her ikisini eklemektedir. CRFL enjeksiyonu saldırısına örnek olarak, bir web sayfası ziyaret edildiğinde ziyaret eden IP, ziyaret edilen zaman ve ziyaret edilen sayfa bilgisinin log dosyalarında tutulduğu varsayılsın. Söz konusu log dosyasına ait girdiler aşağıdaki şekilde görülecektir.

- 192.122.155.123 - 08:20 - /index.php?page=aboutus

Saldırganlar kaydın oluşmasını sağlayan sorguya CRLF karakterlerini ekleyebilirse, log dosyası içindeki kayıtları taklit edebilmekte, hatta istediği türden kayıtlar ekleyebilmektedir. Eğer saldırgan yukarıdaki girdiyi aşağıda gösterilen şekilde değiştirebilirse log kayıtlarını isteği doğrultusunda değiştirebilecektir.

- /index.php?page=aboutus&%0d%0a127.0.0.1 -08:20- /index.php?page= aboutus &innerpage=edit

Yukarıdaki kayıtta görülen %0d ve %0a karakterleri sırası ile carriage return ve line feed karakterlerinin encode edilmiş halidir. Saldırgan söz konusu karakterleri ekledikten sonra log kayıtları aşağıdaki hali alacaktır.

- 192.122.155.123 - 08:20 - /index.php?page=aboutus
- 127.0.0.1 - 08:20 - /index.php?page= aboutus & innerpage =edit

Bu örnekte saldırgan gerçekleştireceği işlemleri gizlemek için CRLF zafiyetini kullanmıştır. Böylelikle kayıt dosyasında yer alan verilerin bir sahtesini üretmiştir.

Bu örnekte saldırganın admin parolasına sahip olduğu ve sadece admin yetkisi ile ziyaret edilebilen inner page sayfasına erişebildiği düşünülsün. Saldırgan CRLF zafiyetini kullanmadan doğrudan admin parolası ile ilgili sayfayı ziyaret ettiğinde, log kayıtlarında bilinmeyen bir IP için kayıt oluşacaktır. Bu kaydı gören admin, birşeylerin yanlış gittiğini anlayacaktır. Fakat yukarıdaki örnekte olduğu gibi CRLF zafiyetini istismar eden bir saldırgan ilgili sayfayı ziyaret edildiğinde bu eylem log sayfasına sanki localhost tarafından yapılmış gibi işlenecektir. Bu sayede admin log kayıtlarında böyle bir veri gördüğünde bu durumu yadırgamayacaktır.

CRLF enjeksiyonu üzerinden gerçekleştirilen saldırılar ilerletilirse siteler arası betik çalıştırma, önbellek zehirlemesi, HTML enjeksiyonu gibi saldırılara neden olabilmektedir.

CLRF zafiyetine karşı alınabilecek en iyi önlem, kullanıcılar tarafından headerlara doğrudan input eklenmesinin önüne geçmektedir. Bunun içinde sunucu tarafında CR ve LF karakterlerini encode eden bir fonksiyon kullanılmalıdır (Morgenroth,2017).

3.2.1.13. Bellek Taşması Zafiyeti

Bellek; çeşitli uygulamalarda verilerin geçici olarak depolanması için kullanılan birimdir. Bellek aynı zamanda aynı türde dizili veri tiplerini depolamak için kullanılan hafıza bloğu olarak da tanımlanabilmektedir. C programlama dilinde ise bellek terimi

array ifadesine karşılık gelmektedir. Array yani diziler, diğer tüm veri türlerinde olduğu gibi statik ve dinamik olarak ikiye ayrılmaktadırlar. Statik tipindeki değişkenler programların veri kısımlarına, dinamik tipteki değişkenler ise, yığın (stack) adı verilen hafızada program için özel olarak ayrılmış bölümde tutlmaktadırlar. Bellek taşması zafiyeti dinamik değişkenlere taşıyabilecekleri maksimum veri miktarından fazlasının atanması durumunda oluşmaktadır. Aşağıdaki kod bloğu bellek taşmasına basit bir örnek olarak gösterilmektedir.

- main(){
  char testArray[5];
  Strcpy (testArray,"bu veri array değişkenin taşıyabileceğinden daha büyük bir veridir.");
  return 0;}

Yukarıdaki örnek kod bloğunda görüldüğü gibi, bir değişkene taşıyabileceğinden daha yüksek bir değerin atanmasından sonra, söz konusu program için bellekte ayrılan hafıza boyutu kadar veri işlem görür. Verinin geri kalan kısmı programa ayrılmamış yığın bölgesine atanmak zorunda kalır. Gerçekleşen bu bellek taşması; programın düzgün çalışmaması, durdurulması ya da çeşitli hata mesajlarının ekrana basılmasına neden olabilmektedir. Saldırganlar tarafından bir girdi alanında bellek taşması zafiyeti tespit edildiğinde, yapılacak ilk işlem yığın alanına taşan verilerin sistemi istismar edecek şekilde yapılandırılmasıdır. Saldırgan verinin taşacak kısmını kötü amaçları doğrultusunda yapılandırırsa, hedef sistem üzerinde zararlı birçok eylemi gerçekleştirebilecektir.

Bellek taşması zafiyetini engellemek için yapılması gerekenlerden ilki, program içerisinde kullanılan fonksiyonlarda girdi denetimi yapmaktır. Bunun yanına bellek taşması kontrolü ve güvenliğini sağlayan kütüphane ve frameworklerin kullanılması da ilgili zafiyetin önüne geçmede oldukça etkili olacaktır (Akgün,2005).

3.2.1.14. URL Yönlendirme (Open Redirect) Zafiyeti

Web geliştiricileri tarafından yaygın olarak bilinen ve göz ardı edilen güvenlik açıklarından biri URL yönlendirme (open redirect) zafiyetidir. Bu zafiyet bir web uygulamasında ziyaretçi tarafından gönderilen bir input değerinin, ziyaretçiyi

yönlendirmek için kullanılmasından dolayı oluşmaktadır. Zararsız bir işlem gibi görünse de, kullanıcının hangi web sayfasına yönlendirilmek istediğini söylemesine izin vermek, ciddi güvenlik sorununa neden olabilmektedir. Aşağıda open redirect zafiyeti barındıran web sitesi için oluşturulmuş zararlı URL adresi görülmektedir.

- https://www.ornek.com/login.html?RelayState=http%3A%2F%2Fornek.com%2Fanasayfa

Yukarıda gösterilen URL adresinde RelayState parametresi, başarılı bir giriş yapan kullanıcının nereye yönlendirileceğini göstermektedir. Bu örnekte kullanıcının yönlendirileceği adres http://ornek.com/anasayfa' dır. Eğer web uygulamasında kullanıcının yönlendirileceği web sayfasının meşru ve amaçlanan sayfa olduğu kontrol edilmezse, saldırganlar söz konusu URL adresini, kullanıcıları sahte bir web sayfasına yönlendirmek için değiştirebilmektedir.

- https://www.ornek.com/login.html?RelayState=http%3A%2F%2zararliwebsitesi.com

Open redirect zafiyetine geliştiriciler yeterli önemi göstermemektedirler. Çünkü söz konusu zafiyetin web uygulamasına doğrudan bir zararı bulunmamaktadır. Fakat bu durum ilgili zafiyetin ciddi bir sorun olmadığı anlamına gelmemektedir. Bu zafiyet, web uygulaması kullanıcılarına oltalama saldırıları gerçekleştirmek için kullanılabilmektedir.

Open redirect zafiyetini engellemenin en temel yolu, kullanıcılar tarafından müdahale edilebilen, GET yöntemi ile sağlanan parametrelere dayalı yönlendirme işlemini kullanmamaktır. Eğer bu yöntem ile yönlendirme işleminin mutlaka kullanılması gerekiyorsa, bir whitelist (beyaz liste) kullanarak yönlendirilecek hedefin listesi tutulmalıdır (Trustwave, 2019).

3.2.1.15. XEE (XML External Entity) Zafiyeti

XEE zafiyeti, çok yeni olmayan fakat 2014 yılında Facebook sunucularında tespit edilmesi ile tekrar gündeme gelen bir zafiyettir. XEE aslında XML kullanımının esnekliğini arttırmak için kullanılan bir özellik olmasına rağmen, önemli güvenlik

risklerini de beraberinde getirmektedir. Söz konusu zafiyet sunucu üzerinden rastgele dosya okunmasına ya da sunucu üzerinde komut çalıştırılmasına neden olabilmektedir.

XEE zafiyeti; sunucu tarafında çalışan bir yazılım XML verisini işlerken, özel olarak tanımlanan bir XML varlığının (entity) çağırılması ile tetiklenmektedir. Web uygulamaları tarafından kullanılan bazı XML kütüphanelerinde varlık tanımlama ve çağırma özelliği varsayılan olarak açık bulunurken bazılarında kapalı olarak bulunmaktadır. XEE zafiyeti üzerinden gerçekleştirilecek bir saldırının başarılı olabilmesi için söz konusu özelliğin açık olması gerekmektedir.

Söz konusu zafiyet kullanılarak hedef sistem üzerinde yetki dışındaki dosyalar izinsiz okunabilmekte ya da uzaktan kod çalıştırılabilmektedir. Aşağıdaki kod bloğunda XML verisini parçalayıp ekrana basan bir php kodu bulunmaktadır.

- <?php$xml = simplexml_load_string(file_get_contents(“php://input”));echo $xml;?>

Bu örnekte, sunucuya basit bir XML entity içeren bir POST isteği gönderildiğinde cevap olarak entitynin değeri dönecektir. Bu durumda sunucuya gönderilen istekte, bir dosyanın içeriğini okuyacak bir entity tanımlanarak kullanılırsa istek yine başarılı olacak, saldırgan yetkisi dahilinde olmayan bir dosyaya erişecektir.

- <?xml version=”1.0″?><!DOCTYPE tag [<!ENTITY ups SYSTEM “file:///etc/passwd”>]><tag>&ups;</tag>

Örnek olarak yukarıdaki XML sunucuya gönderildiğinde, cevap olarak geriye etc dizini altındaki passwd dosyasının içeriği döndürülecektir. XEE zafiyeti üzerinden ilerlendiğinde hedef sunucu üzerinde komut çalıştırma işlemi dahi gerçekleştirilebilecektir. Örneğin, hedef sunucuda PHP’ nin expect modülü kuruluysa, söz konusu modül, komut çalıştırma işleminde kullanılabilecektir.

3.2.2. Web Uygulama Sızma Testleri

Bilgi güvenliğini sağlamada sızma testleri gerçekleştirmek en yaygın ve gerekli işlemlerin başında gelmektedir. Daha önceki bölümlerde sızma testleri, öncesinde test uzmanına sağlanan bilgi çeşitliliği ve testin yapıldığı konumlara göre beyaz, gri ve

siyah olarak gruplandırılmıştı. Söz konusu gruplar, kuruma ait farklı bileşenler üzerinde gerçekleştirilecek sızma testlerinin tamamında kullanılabilmektedir. Sızma testleri ayrıca, teste tabi tutulan ya da hedef alınan sistemlerin çeşitliliğine göre de farklılıklar göstermektedir. Kurum ağına yönelik gerçekleştirilecek sızma testlerinde; sunucular, saldırı tespit sistemleri, yönlendiriciler, güvenlik duvarları vb bileşenler hedef alınırken, kurum çalışanlarına yönelik gerçekleştirilecek sızma testlerinde çok farklı senaryolar kullanılmaktadır. Kuruma ait web uygulamasına yönelik gerçekleştirilecek sızma testi de doğal olarak farklı işlemler içermektedir.

Web uygulama sızma testlerinde yapılacak ilk işlem testin kapsamının belirlenmesidir. Kapsam terimi burada, taramaya tabi tutulacak sayfaları ifade etmek için kullanılmaktadır. Bazı durumlarda sadece kullanıcılar tarafından en sık ziyaret edilen yer olan ana sayfanın test edilmesi istenirken, bazı durumlarda sadece alt domainlerin test edilmesi istenebilmektedir. Tam kapsamlı bir web uygulama sızma testinin gerçekleştirilebilmesi için, uygulamanın sahip olduğu tüm web sayfalarının kapsama dahil edilmesi en güvenilir sonuçların elde edilmesini sağlayacaktır. Bunun yanında, kullanıcı girişine sahip uygulamalarda sadece kullanıcı giriş alanı test edilerek çeşitli bypass yöntemleri ile giriş aşaması geçilmeye çalışılmaktadır. Bu durum, siyah kutu sızma testlerine güzel bir örnektir. Web uygulama sızma testlerinde gerçekleştirilen bir diğer senaryoda ise test uzmanlarına geçerli bir kullanıcı adı parola verilmekte, kullanıcı panellerinin de teste tabi tutulması istenmektedir. Bu durum da beyaz kutu sızma testlerine bir örnektir.

Web uygulama sızma testlerinde belirlenmesi gereken bir diğer husus ise, gerçekleştirilecek testlerde URL üzerinden HTTP GET metodu mu, sayfa içerisindeki formlar aracılığı ile HTTP POST metodu mu yoksa her ikisinin de mi kullanılacağının belirlenmesidir. URL üzerinden gerçekleştirilen testlerde, URL adresi içerisindeki parametreler hedef alınmaktadır.

- https://www.teztest.com/satis.html?urun=bilgisayar

Yukarıda gösterilmekte olan URL adresi incelendiğinde, urun parametresine bilgisayar değerinin atandığı, bu atamaya göre sayfa üzerinde gösterilecek içeriğin şekillendiği görülmektedir. URL üzerinden gerçekleştirilecek testlerde urun parametresi hedef olarak alınmalı, söz konusu parametreye çeşitli zafiyetleri

tetikleyecek payloadlar atanarak sayfanın vereceği tepkiler ölçülmelidir. Örneğin, ilgili parametre aracılığı ile sayfanın SQL enjeksiyonu zafiyeti barındırıp barındırmadığını kontrol etmek için URL aşağıdaki şekilde değiştirilebilmektedir.

- https://www.teztest.com/satis.html?urun= 1\' OR \'1\'=\'1

Bu durumda kullanıcı tarafından ilgili parametreye atanan 1\' OR \'1\'=\'1 ifadesi herhangi bir filtrelemeden geçmeden veritabanı sunucusuna sorguya eklenmek üzere gönderiliyorsa, SQL enjeksiyonu zafiyeti tespit edilecektir.

Günümüzde URL rewrite engineleri(motorları) kullanılarak karmaşık URL adresleri arama motorlarının ve kullanıcıların daha net anlayacağı hale getirilmektedir. Bu işlemin bir diğer amacı da sitenin iç işleyişinin ve sorgu mantığının anlaşılmasını engellemektir. Aşağıda URL rewrite ile sadeleştirilmiş URL örneği gösterilmektedir.

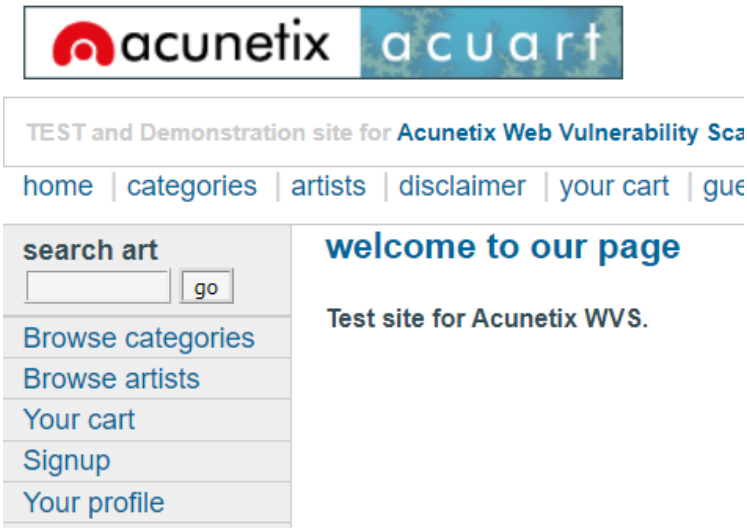
- https://www.teztest.com/satis.html?urun=bilgisayar
- https://www.teztest.com/satis.html/bilgisayar/

URL rewrite tekniği ile sadeleştirilen URL adreslerinin test edilmesi işleminde test payloadları / karakterleri arasına yerleştirilmektedir. Yukarıdaki URL adresinin SQL enjeksiyonu zafiyetine karşın test edilmesi işlemi aşağıdaki şekilde yapılmalıdır.

- https://www.teztest.com/satis.html/1\' OR \'1\'=\'1/

Yukarıdaki URL adresine istek yapıldığında, URL içerisindeki bileşenler rewrite engine tarafından ayrıştırılacak ve sorgu işleminde kullanılacaktır. Yine bir önceki örnekte olduğu gibi söz konusu web sayfası SQL enjeksiyonu barındırıyorsa ilgili payload'a tepki verecektir.

Web uygulama sızma testlerinde kullanılan bir diğer saldırı vektörü, web sayfaları üzerindeki formlarda yer alan input yani kullanıcı girdi alanlarıdır. Web uygulamaları üzerinde sızma testleri gerçekleştirilirken yapılacak işlem söz konusu girdi alanlarına çeşitli zafiyetleri tetikleyecek zararlı payloadları girerek uygulamanın tepkisini test etmektir. Şekil 3.8' de Acunetix firması tarafından geliştirilmiş içerisinde birçok zafiyeti barındıran http://testphp.vulnweb.com/ sayfasına ait arama inputu gösterilmektedir.



Şekil 3.8. http://testphp.vulnweb.com/ anasayfası

Şekil 3.8' de gösterilmekte olan http://testphp.vulnweb.com adresinin anasayfasında yer alan arama inputunun XSS zafiyetine karşı test edildiği varsayılsın. Bu işlem için girdi alanına XSS zafiyetini tetikleyici payload girilmeli, "go" butonuna tıklanmalı ve sayfanın tepkisi test edilmelidir. Eğer girilen payloadın türüne göre sayfa üzerinde bir eylem gerçekleşiyorsa bu sonuç, arama inputuna girilen değerlerin herhangi bir kontrolden geçirilmediği ve ilgili inputun XSS zafiyeti barındırdığı anlamına gelmektedir.

Web uygulamaları üzerinde gerçekleştirilecek sızma testlerinde dikkat edilmesi gereken bir diğer nokta ise, bir kontrol listesinin tutulmasıdır. Sızma testlerinde kullanılan kontrol listelerinde, test esnasında gerçekleştirilmesi gereken adımlar, kontrol edilmesi gereken alanlar, her bir adımda kullanılması gereken araçlar gibi bilgiler yer almaktadır. Bir web uygulaması üzerinde sızma testi gerçekleştirilirken kullanılacak kontrol listesi, tam kapsamlı ve eksiksiz bir test gerçekleştirilmesine olanak sağlayacaktır.

Web uygulamaları üzerinde gerçekleştirilen sızma testleri güvenlik uzmanları tarafından manuel olarak gerçekleştirilebildiği gibi, bu iş için geliştirilmiş çeşitli otomatize araçlar kullanılarak da gerçekleştirilebilmektedir. Söz konusu araçların çalışma mantığının daha iyi anlaşılabilmesi için, dinamik analiz kavramına değinilmesi gerekmektedir.

3.2.3. Dinamik Kod Analizi

Bir web uygulaması üzerinde yer alan güvenlik açıklarını belirlemek için mevcut iki yöntem, statik analiz ve dinamik analiz yöntemleridir.

Statik analiz, diğer adı ile beyaz kutu analizi yönteminde, web uygulamasının kaynak kodu incelenmekte ve olası güvenlik açıklarının tespit edilebilmesi için tüm program dallanmaları denenmektedir. Fakat statik kod analizinin dezavantajı, sadece uygulamanın çalıştırılması sırasında tespit edilebilen güvenlik açıklarını belirlemede yetersiz kalmasıdır. Ayrıca statik analiz yöntemi, dinamik analiz yöntemine göre daha fazla hatalı pozitif (false positive) sonuç üretme eğilimindedir. Bunun yanında yazılım geliştirme süreci bir döngü olduğu için statik analiz yöntemi bu döngü içerisinde yetersiz kalmaktadır.

Web uygulaması üzerinde zafiyetlerin tespiti için kullanılan bir diğer yöntem dinamik analiz, diğer adı ile siyah kutu analizidir. Dinamik analiz yönteminde ise, aktif olarak hizmet vermekte olan bir web uygulamasına yönelik zararlı girdiler gönderilerek uygulamanın yanıtı analiz edilmekte, elde edilen yanıtlara göre güvenlik açıkları tespit edilmektedir. Dinamik analiz yöntemi ile elde edilen sonuçlar, uygulamanın çalışma zamanı (runtime) davranışlarına dayanarak üretildiği için, yüksek oranda false pozitif sonuç üreten statik analiz yöntemine göre daha güvenilirdir. Bunun yanında uygulamanın tüm alanları test edilemediğinden dolayı, statik analize göre daha az sonuç ortaya çıkabilmektedir (Deepa ve Thilagam, 2016).

Dinamik analiz yönteminin bir diğer avantajı, yayında olan bir web sayfası üzerinde gerçekleştirildiği için, gerçek bir siber saldırının birebir simülasyonunun yapılmasına imkân tanımasıdır. Bir web uygulamasına yönelik olarak gerçekleştirilen bir siber saldırıda, salgırganların önünde hedef alacakları bir web sitesi, saldırı payloadlarını yükleyecekleri input alanları bulunmaktadır. Dolayısıyla dinamik analiz tekniği ile bir web uygulamasını test eden bir güvenlik uzmanı, saldırgan gibi düşünmek ve onun gibi hareket etmek zorundadır. Bu da tam kapsamlı bir güvenlik testi yapılmasına olanak sağlamaktadır.

Günümüzde web uygulamaları üzerinde gerçekleştirilen sızma testlerinin büyük çoğunluğu bir önceki paragrafta belirtilen sebeplerden dolayı dinamik analiz yöntemi ile gerçekleştirilmektedir. Güvenlik uzmanlarının azımsanmayacak kadarı söz konusu testleri manuel olarak gerçekleştirmektedir. İlgili testlerde güvenlik uzmanları, kurumun sahip olduğu web sayfası üzerinden yola çıkarak kuruma ait ulaşabildiği tüm alt sayfalar (subdomain) üzerinde çeşitli saldırı teknikleri kullanarak, web uygulamasının ilgili zafiyetleri barındırıp barındırmadığını tespit etmekte ve raporlamaktadır. Fakat, söz konusu süreç güvenlik uzmanı tarafından elle yürütüldüğünden dolayı uzun zaman alabilmekte, bunun yanında bazı alt sayfaların test edilmesi unutulabilmektedir. Bahsedilen dez avantajlardan dolayı günümüzde web uygulama testlerinde otomatik zafiyet tarayıcıların kullanımı oldukça yaygınlaşmıştır. Otomatik zafiyet tarayıcıların işleyişi kullanıcının tercihlerine göre şekillenmektedir. Tam kapsamlı çalışma için ayarlanmış bir zafiyet tarayıcısı ilk olarak, kendisine verilen URL adresinden yola çıkarak kuruma ait tüm sub domain ve alt URL adreslerini kaydetmektedir. Daha sonra kaydedilen her bir URL adresi tek tek ziyaret edilerek, tercihe göre GET ya da POST metodları üzerinden zafiyet testlerini gerçekleştirmektedir. Söz konusu zafiyet tarayıcılar, test ettiği zafiyet çeşitliliği, test edebildiği metod türü (GET ya da POST), oturum açma ve kaydetme gibi birçok yönden farklılık göstermektedir.

3.3. Rastgele Orman (Random Forest) Algoritması

Rastgele orman (random forest) algoritması, denetimli (supervised) bir makine öğrenmesi algoritmasıdır. Random forest algoritmasında, adından da anlaşılacağı üzere, eğitilmiş karar ağaçları topluluğu olan rastgele bir orman oluşturulmaktadır. Random forest algoritmasında farklı öğrenme modellerinden elde edilen sonuçlar, daha doğru ve istikrarlı bir tahmin elde edebilmek için kombine edilmektedir. Random forest algoritmasının daha iyi anlaşılabilmesi için toplu sınıflandırma yöntemlerine değinmek gerekmektedir.

Rastgele orman algoritması sınıflandırılma modeli olarak kullanıldığında; pek çok veri seti için, Adaboost ve Destek Vektör Makineleri algoritmalarına göre daha kesin sonuçlar sunmaktadır. Ayrıca söz konusu algoritma çok kısa sürede sonuç vermektedir. Rastgele orman algoritmasının bir diğer avantajı; içerisinde binlerce

değişken ve çok sayıda sınıf etiketi içeren, dengesiz bir dağılıma sahip veri setlerinde dahi yüksek doğruluk oranına sahip sonuçlar üretmesidir (Yılmaz, 2014).

Toplu sınıflandırma yöntemleri, tek bir sınıflandırıcı yerine, birçok farklı türde sınıflandırıcı üreten ve o sınıflandırıcılardan elde edilen tahminler doğrultusunda veriyi sınıflandıran algoritmalardır. Günümüzde en yaygın kullanılan toplu sınıflandırma algoritmaları; torbalama, hızlandırma ve rastgele orman algoritmalarıdır (Akar, 2012).

Torbalama (bagging) algoritmasında, veri setinden birden çok eğitim seti oluşturulmaktadır. Oluşturulan her bir veri seti için bir ağaç üretilmektedir. Ağaçların üretilmesinden sonra, genel modelin tahmin işleminde "en çok oy" baz alınmaktadır (Liaw ve Wiener, 2012).

Günümüzde yaygın olarak kullanılmakta olan bir diğer toplu sınıflandırma yöntemi Hızlandırma'dır. Hızlandırma algoritmasında tekrarlı eğitim tekniği kullanılmaktadır. Bu modelde yanlış tahmin edilen örneklerin bir sonraki iterasyonda ağırlıkları arttırılmaktadır. Hızlandırma algoritması çoğu durumda torbalama algoritmasına göre daha iyi sonuçlar üretmektedir. Fakat yavaş olması ve gürültüye karşı aşırı hassas olması, söz konusu algoritmanın dezavantajlarıdır (Gislason vd. 2006).

Rastgele orman algoritması, ağaç tipi sınıflandırıcılardan oluşan bir topluluk olarak düşünülebilmektedir. Söz konusu algoritma, rastgelelik özelliği eklenmiş ve daha da geliştirilmiş bir torbalama (bagging) versiyonudur. Rastgele orman algoritması; eldeki tüm değişkenler içerisinden en iyi dalı seçmek yerine, her düğümde rastgele seçilmiş değişkenlerden en iyisini kullanarak her düğümü dallara ayırmaktadır. Rastgele orman algoritmasındaki tüm veri setleri, orijinal veri setinden yer değiştirerek üretilmektedir. Veri setlerinin üretilmesinden sonra, rastgele öznitelik seçilerek ağaçlar geliştirilmektedir. Geliştirilen ağaçların budanmaması, rastgele orman algoritmasının doğruluk oranının artmasını sağlamaktadır. Rastgele orman algoritmasının bir diğer avantajı, çok hızlı bir algoritma olmasıdır (Akar, 2012).

Rastgele orman algoritmasının çalıştırılması için ilk olarak kullanıcıdan iki adet parametre tanımlaması istenmektedir. Bu değişkenler; her bir bölümde kullanılacak

değişkenlerin sayısını ifade eden (m) ve geliştirilecek toplam ağaç sayısını ifade eden N' dir. Bu işlemden sonraki ilk adım, eğitim ve test setlerinin belirlenmesidir. Kullanılan veri setinden bootstrap (hızlandırma) yöntemi ile n adet örnekleme seçilmektedir. Seçilen her örnekleme verisinin 2/3 oranındaki kısmı ağaç oluşturmak için kullanılmaktadır. Bu kısma eğitim seti, diğer adı ile inbag denilmektedir. Örnekleme verisinden geriye kalan 1/3' lük kısım ise hata oranını hesaplamada kullanılacak olan test setidir. Test setine aynı zamanda out-of-bag denilmektedir.

Oluşturulan eğitim seti içerisinden rastgele olarak seçilen değişkenler içinde bilgi kazancı en yüksek değişken, dallara ayırıcı değişken olarak kullanılmaktadır. Bu yöntem literatürde; classification and regression tree (CART) algoritması olarak geçmektedir. Bilgi kazancının belirlenmesinde kullanılan formül denklem 3.1' de gösterilmektedir.

$$Bilgi\ Kazanç\ Değeri(D,X) = E(D) - \sum_{i=}^{n} p(Di)E(Di) \qquad (3.1)$$

Denklem 3.1' de gösterilmekte olan denkleme ait bileşenlerin ne anlama geldiği aşağıda listelenmiştir.

- X= Bölünecek olan değişken,
- D=Bölünecek olan özelliğe ait alt kümeler,
- E(D)=X değişkenine göre bölünme öncesi entropi değeri,
- E(Di)=X değişkenine göre bölünme sonrası i. alt kümeye ait entropi değeri.

Rastgele orman algoritmasında dallara ayırma için gerekli eşik değeri; gini indeksi ile belirlenmektedir. Rastgele orman algoritmasında, terminal düğüm elde edilene kadar, anlatılan işlemler tekrar edilmektedir.

Rastgele orman algoritmasında, yeni bir verinin hangi sınıfa ait olduğunun tahmin edilmesi için, n örneklemeden oluşturulmuş, n tane ağacın ürettiği tahminler birleştirilir. Birleştirme sonunda yapılan oylama ile en çok oyu alan sınıf, veri için tahmin edilen sınıf olmaktadır. Bu oy, test setinin hata oranı ile belirlenmektedir.

Bu algoritmada, eğitim veri seti kullanılarak oluşturulan her ağaç üzerinde, test veri seti uygulanarak hata oranı hesaplanmaktadır. Bu hata oranı; ormandaki her ağacın kendi hata oranına ve aralarındaki kolerasyona bağlı olarak hesaplanmaktadır. Geliştirilen modelde; her ağaçta verilerin ve değişkenlerin birbirinden farklı olmasından dolayı, elde edilen hata oranı düşük çıkmaktadır. Elde edilen hata oranı düştükçe, kullanılan sınıflandırma algoritmasının performansı artmaktadır (Korkem, 2013).

3.4. Apriori Algoritması

Appriori algoritması, 1994 yılında, Agrawal ve Srikant tarafından geliştirilmiş bir birliktelik çıkarım algoritmasıdır. Apriori terimi; algoritmanın sık geçen küme elemanlarının önsel bilgisinin kullanmasından, yani bilgileri bir önceki adımdan almasından dolayı, "bir önceki" - "prior" ifadesinden gelmektedir (Chen vd. 2006).
Apriori algoritmasının temel mantığına göre; eğer k-öge kümesi, belirlenmiş minimum destek eşiğini aşıyorsa, söz konusu kümenin alt kümeleri de ilgili minimum eşik değerini sağlar. Bu ifadede bahsedilen küme ifadesi, bir ya da daha fazla elemandan oluşan bir kümedir. K-öge kümesi ifadesi ise, içerisinde k tane eleman bulunan küme anlamına gelmektedir (Ay ve Çil, 2008).

Bir veri üzerinde apriori algoritmasının çalıştırılması sonrasında elde edilen her bir kural yani ögeler arasındaki birliktelik kuralı; destek ve güven kavramları ile açıklanmaktadır. Destek yani support ifadesi, kural sonucu ilişki bulanan ögeler arasındaki birlikteliğin sıklığı anlamına gelmektedir. Güven yani confidence ifadesi ise, iki öge arasındaki birlikteliğin doğruluğunu ifade etmektedir (Güngör vd. 2013).

Apriori algoritmasının çalışma mantığının anlaşılabilmesi için bir örnek üzerinden devam edilecektir. A ve B ürünleri, bir markette satılan birbirinden farklı birer üründür. A ürünü için destek eşiği, A ürününün satın alındığı sepetlerin, tüm sepetlere oranıdır. A ürünü için destek değeri, denklem 3.1' de gösterilmektedir.

$$Destek(A) = \frac{A \text{ ürününü } içeren\ sepet\ sayısı}{Tüm\ sepetlerin\ sayısı} \tag{3.2}$$

A ve B ürünleri için destek eşiği ise, her iki ürünün beraber alındığı sepet sayısının, tüm sepet sayısına oranı şeklinde hesaplanmaktadır. Denklem 3.2' de söz konusu destek değerinin hesaplaması gösterilmektedir.

$$Destek(A,B) = \frac{A\ ve\ B\ \text{ürün}lerini\ içeren\ sepet\ sayısı}{Tüm\ sepetlerin\ sayısı} \tag{3.3}$$

B ürününün hangi olasılıkla A ürününün alındığı sepette olacağı ise güven değeri ile ifade edilmektedir. Bu güven yani confidence değeri denklem 3.3 ya da 3.4' te gösterilmekte olan işlemlerden bir tanesi ile hesaplanabilmektedir.

$$Güven(A,B) = \frac{A\ ve\ B\ \text{ürün}lerini\ içeren\ sepet\ sayısı}{A\ \text{ürününü}\ içeren\ sepetlerin\ sayısı} \tag{3.4}$$

$$Güven(A,B) = \frac{Destek(A,B)}{Destek(A)} \tag{3.5}$$

Apriori algoritması kullanılarak elde edilen kuralların ne kadar kullanışlı, doğru ve güvenilir olduğu, ilgili denklemlerde bahsedilen destek ve güven kriterleri ile belirlenmektedir. Apriori algoritmasından elde edilen ilişkilerin güvenirliğini test etmede kullanılan bir diğer kriter ise "Lift" kriteridir. Lift değeri, arasında ilişki bulunan iki ürün arasındaki kolerasyonu ifade etmektedir. Lift değeri 1' den büyükse iki ürün arasında pozitif kolerasyon var anlamına gelmektedir. Lift değeri 1' den küçük ise iki ürün arasında bulunan ilişkinin bir anlam ifade etmediğini göstermektedir.

Bir veri üzerinde Apriori algoritması ile birliktelik analizi yapılacağında, kullanıcıdan ilk olarak, elde edilecek kuralların geçerliliğini belirlemek amacıyla destek ve güven eşiğini girmesi istenmektedir. İki ürün arasındaki ilişkinin önemli olması için hem destek hem de güven kriterlerinin yüksek olması gerekmektedir. Yüksek destek eşiği belirlemek algoritmanın daha hızlı çalışmasını sağlarken aynı zamanda elde edilen kural sayısında düşüşe neden olacaktır. Örneğin, yapılan alışverişlerde satın alınan ürünlerin aynı sayıda alınmaması, yüksek tutulan destek eşiği ile birleştiğinde, söz konusu ürünler ile ilgili birliktelik kuralı elde edilememesine neden olacaktır. Her bir

ürün için bir birliktelik kuralı elde etmek isteyen bir araştırmacının destek eşiğini aşağıya çekerek, sonuçları yüksek güven değerine göre filtrelemesi gerekmektedir.

4. ARAŞTIRMA BULGULARI

Bu tez çalışmasında yöntemlerin uygulanması ve gerçekleştirilen işlemlerin detayları aşağıda başlıklar halinde detaylı olarak anlatılmıştır. Çalışmada, dinamik analiz ve yapay zeka tabanlı web zafiyet tarayıcısı geliştirilmiştir. Söz konusu araç Pycharm ortamında Python programlama dili kullanılarak konsol tabanlı geliştirilmiştir. Web zafiyet tarayıcısının geliştirilmesinden sonra, zafiyet tarama ve veri toplama işlemlerinin gerçekleştirilebilmesi için PHP ve ASP tabanlı web projeleri kullanılarak bir web sızma testi laboratuvarı oluşturulmuştur. Oluşturulan laboratuvar üzerinde geliştirilen araç kullanılarak elde edilen veri seti, sayfa sınıflandırma ve zafiyetler arası birliktelik analizi işlemlerinde kullanılmıştır. Bu bölümün ilerleyen başlıklarında gerçekleştirilen işlemler detaylı olarak anlatılmıştır.

4.1. Yapay Zekâ ve Dinamik Analiz Tabanlı Web Zafiyet Tarayıcısı

Bu tez çalışması kapsamında geliştirilen web zafiyet tarayıcısı Pycharm derleyicisi üzerinde Python programlama dili ile kodlanmıştır. Geliştirilen web zafiyet tarayıcısı; ana modül, kapsam genişletme modülü, zafiyet test modülü, öznitelik çıkarma modülü, raporlama modülü, sınıflandırma modülü olmak üzere 7 farklı bileşene ayrılmaktadır.

4.1.1. Web Zafiyet Tarayıcısı Ana Modülü

Çalışma kapsamında geliştirilen web zafiyet tarayıcısı, kullanım kolaylığı ve güvenlik araçlarındaki yaygınlığı sebebiyle komut ekranı üzerinden çalışmaktadır. Web zafiyet tarayıcısına birçok farklı modül ve özelliğin eklenmiş olması, beraberinde farklı amaç ve modlarda çalışma imkânı da sunmaktadır. Kimi kullanıcılar sadece web sitesinden URL adreslerinin çıkarılmasını isterken, kimi kullanıcılar sadece tek bir URL adresinin taranmasını isteyebilmektedirler. Ya da, bazı kullanıcılar sadece subdomainlerin taranmasını isterken, bazıları ise gizlenmiş, kaba kuvvet yöntemi ile tespit edilebilecek URL adreslerinin elde edilmesini isteyebilmektedirler. Bu durumda her bir farklı senaryo için kullanıcıya seçme imkânının verilmesi gerekmektedir. Yine kullanıcının tercihine göre, testlerin GET metodu mu yoksa POST metodu üzerinden mi gerçekleştirileceği, tek thread mi yoksa multi thread mi kullanılacağı gibi işlemlerin

belirlenmesi de argümanlar üzerinden gerçekleştirilmektedir. Şekil 4.1' de geliştirilen araç için tanımlanmış argüman listesi verilmiştir.

```
parser = argparse.ArgumentParser()
parser.add_argument('-u','--url', dest='url',  required=True, action='store', help= "T
parser.add_argument('-sd', '--subdomain', dest='subdomain', required=False, action='st
parser.add_argument('-c', '--crawl', dest='crawl', required=False, action='store_true'
parser.add_argument('-m', '--method', dest='method', required=False, action='store',he
parser.add_argument('-a', '--attack', dest='attack', required=False, action='store_tru
parser.add_argument('--threading', dest='threading', required=False, action='store_tru
parser.add_argument('-fs', '--fullscan', dest='fullscan', required=False, action='stor
parser.add_argument('--username', dest='username', required=False, action='store',help
parser.add_argument('--password', dest='password', required=False, action='store',help
parser.add_argument('--exception', dest='exception', required=False, action='store',he
parser.add_argument('-b', '--brute', dest='attack', required=False, action='store_true
```

Şekil 4.1. Web zafiyet tarayıcısı kullanım argümanları

Şekil 4.1' de gösterilen argümanlar ve kullanım amaçları Çizelge 4.1' de gösterilmektedir.

Çizelge 4.1. Subdomain bulma modülü çalışma adımları

Argüman	Kullanım Amacı
-u ya da --url	Taranması istenen URL adresi girilmesi için kullanılmaktadır.
-c ya da --crawl	Kullanıcı tarafından girilen URL adresinden yola çıkarak tüm sitenin taranması (crawl edilmesi) için kullanılmaktadır.
-sd ya da --subdomain	Daha önce verilmiş URL adresine ait subdomain adreslerinin arama motorları aracılığı ile toplanması için kullanılmaktadır.
-b ya da --brute	Kullanıcı tarafından verilen web sayfası içerisinde adreslenmemiş gizli web sayfalarının brute force tekniği ile toplanması için kullanılmaktadır.
-m ya da --method	Zafiyet testinin hangi HTML metodu üzerinden gerçekleşeceğini belirtmek için kullanılmaktadır.
-a ya da --attack	Kullanıcı tarafından girilen web sitesi üzerinde zafiyet testlerinin başlatılması için kullanılmaktadır.

Çizelge 4.1. Subdomain bulma modülü çalışma adımları (Devam)

--username ve --password	Kullanıcı girişi gereken web sitelerinin taranmasında başarılı kullanıcı girişi yaparak, login arkasında kalan sayfaların da test edilebilmesi için kullanılmaktadır.
-f ya da --fullscan	-c ve -a argümanlarının aynı anda kullanıldığı durumlar için tanımlanmıştır.
-h ya da --help	Yardım menüsünün görüntülenmesi için kullanılmaktadır.
--threading	Argümanı taramalar esnasında aracın tek thread mi yoksa multi thread mi çalışacağının belirlenmesi için kullanılmaktadır.
--exception	zafiyet taraması esnasında, taranması istenmeyen bir URL adresi varsa, onu belirtmek için kullanılmaktadır.

Çizelge 4.1' de gösterilen argümanlar kullanılarak, örnek http://teztest.com adresi üzerinde farklı türde yapılacak tarama örnekleri aşağıda verilmiştir.

1. python sansar.py -u http:// teztest.com -c
2. python sansar.py -u http:// teztest.com -c -a -m both
3. python sansar.py -u http:// teztest.com -c -a -m both –username admin --password parola –threading

Yukarıdaki örnekler incelendiğinde ilk örnekte web zafiyet tarayıcısına -u argümanı ile verilen URL adresinden yola çıkarak tüm siteyi dolaşması ve elde ettiği adresleri kaydetmesi söylenmiştir. İkinci kullanım örneğinde, yine aynı URL adresinden yola çıkarak tüm adresleri toplaması, daha sonra toplamış olduğu URL adreslerine sırayla hem GET hem de POST metodları üzerinden zafiyet taraması gerçekleştirmesi söylenmiştir. Üçüncü kullanım örneğinde ise, ikinci kullanım örneğine ek olarak ziyaret ettiği sayfalarda karşılaştığı girdi alanlarına kendisine verilen kullanıcı adı ve parola bilgilerini girmesi söylenmiştir. Böylelikle zafiyet tarayıcısı sayfaları dolaşırken bir login sayfası ile karşılaştığında, kendisine verilen kullanıcı adı ve parola bilgilerini kullanarak oturum açabilecek, kullanıcı girişi gerektiren sayfaları dolaşma ve test etme imkânı elde edecektir.

Web zafiyet tarayıcısı, yukarıda listelenmiş kullanım örneklerinden 3.sü gerçekleştirildiğinde ilk olarak -c parametresinden dolayı kapsam genişletme modülünü çalıştıracaktır. Kapsam genişletme modülü çalıştıktan sonra elde edilen URL listesi saldırı modülüne verilecektir. Saldırı modülü toplanan listedeki her bir URL adresine sayfa sınıflandırma ve birliktelik analizi kapsamında belirlenen test işlemleri gerçekleştirecek, zafiyet tepsit edildiği anda öznitelik çıkarma modülünü çalıştıracaktır. Öznitelik çıkarma modülü zafiyet tespit edilen sayfa, form ve inputa ait öznitelikleri çıkardıktan sonra test işlemi devam edecektir. Listedeki son URL adresi de test edildikten sonra, öznitelik çıkarma modülü tarafından toplanan veriler raporlama modülüne gönderilecektir. Raporlama modülü kendisine gönderilen verileri taramanın yapıldığı gün ve saat bilgisi ile bir excel dosyasına yazacaktır. Böylelikle tarama sonunda tarama raporu elde edilmiş olacaktır. Bu işlemlerden sonra zafiyet tarayıcısı çalışmayı sonlandıracaktır. Şekil 4.2' de web zafiyet tarayıcısının akış şeması verilmiştir.

Şekil 4.2. Web zafiyet tarayıcısı akış şeması

4.1.2. Web Zafiyet Tarayıcısı Kapsam Genişletme Modülü

Web uygulama sızma testlerinde en önemli noktalardan biri yapılacak testin kapsamının belirlenmesidir. Kapsam belirleme işlemi kullanıcı tarafından verilen argümanlar doğrultusunda gerçekleştirilmektedir. Eğer kullanıcı tarafından argüman olarak verilen URL adresinden yola çıkarak diğer URL adreslerinin toplanması amacıyla bir argüman verilmediyse, araç kendisine verilen tek URL adresi üzerinde test işlemlerini gerçekleştirmektedir. Kullanıcılar tarafından girilen argümanlar içerisinde, kapsam genişletme modülü bileşenleri tarafından kullanılanlar sırası ile, -sd, -b, -c, --username, --password argümanlarıdır. Kullanıcı tarafından girilen argümanlara göre kapsam genişletme modülü altında çeşitli sınıflar çalıştırılmaktadır.

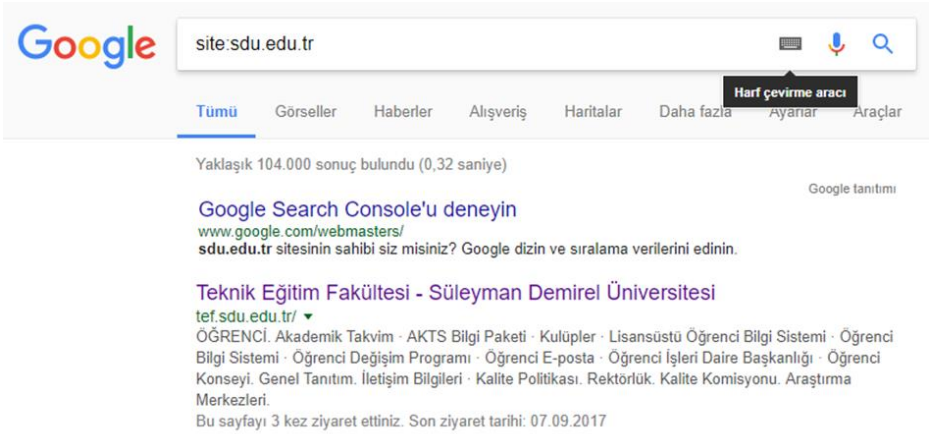
Kapsam genişletme modülü içerisinde yer alan bileşenler; subdomain bulma modülü, brute force (kaba kuvvet) modülü, örümcek (crawler) modülüdür.

4.1.2.1. Web Zafiyet Tarayıcısı Alt Domain Bulma Modülü

Bir web uygulamasına yönelik olarak sızma testi gerçekleştirileceği zaman test uzmanlarına çoğunlukla tek bir web adresi verilmektedir. Böyle durumlarda güvenlik uzmanlarından, söz konusu web adresinden yola çıkarak diğer alt adresleri elde etmesi ve söz konusu tüm adreslere yönelik testler gerçekleştirmesi istenmektedir. Bir sızma testi ilk olarak keşif adımı ile başlamakta, daha sonra tarama, sızma ve raporlama gibi adımlarla devam etmektedir. Alt domain bulma işlemi, keşif adımının en önemli parçalarından biridir. Günümüzde güvenlik uzmanları tarafından alt domain bulma işleminde izlenen çeşitli yollar ve kullanılan çeşitli araçlar bulunmaktadır.

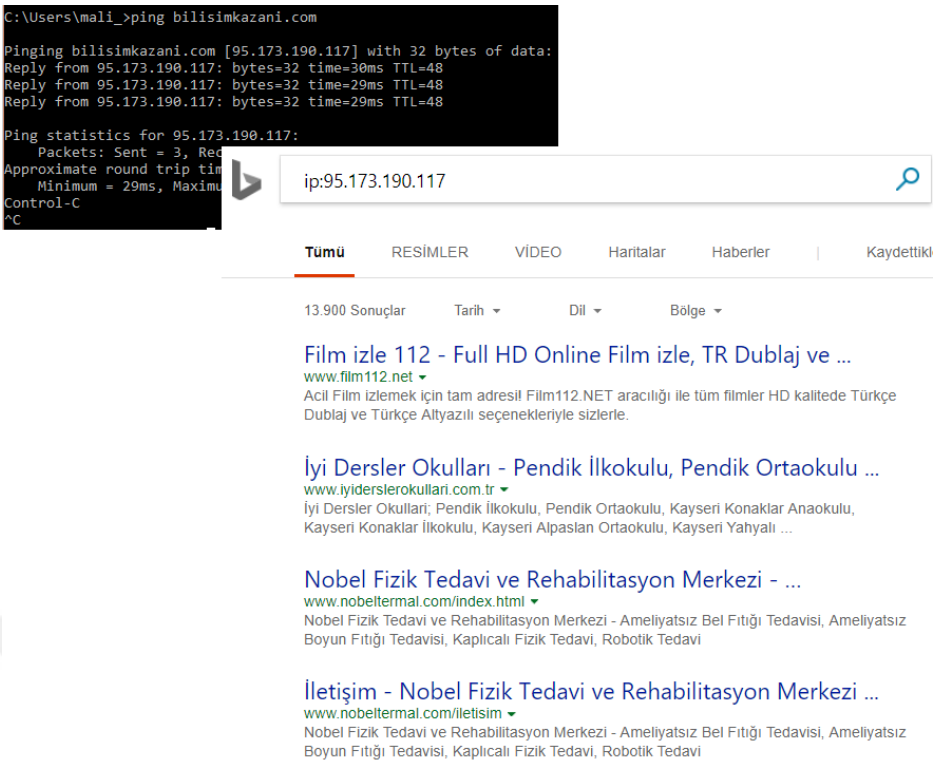
Güvenlik uzmanları tarafından alt domain bulma işlemi için kullanılan yöntemlerden biri arama motorlarıdır. Google, Baidu, Yahoo, Ask, Bing gibi arama motorları, arama sorgularını hassaslaştırmak için çeşitli gelişmiş arama operatörlerini desteklemektedir. Google için bu operatörler Google Dorks olarak adlandırılmaktadır (Hudak, 2017). Google' ın bir alan adı için bulduğu tüm alt alanları bulmak için Google aramasında "site" operatörü kullanılabilmektedir. Örnek olarak, sdu.edu.tr uzantılı domain adreslerinin listelenmesi için gerçekleştirilen google araması Şekil 4.3' de gösterilmiştir. Günümüzde google arama motoru üzerinden alt domain bulma işlemi

için google arama motorunu kullanan araçlara theHarvester aracı örnek olarak gösterilebilmektedir.



Şekil 4.3. Google üzerinden site operatörü ile alt domain bulma işlemi

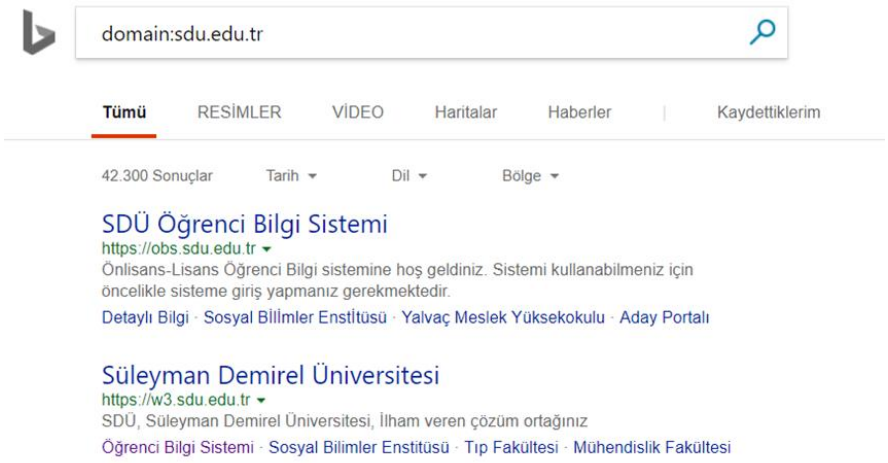
Bing arama motoru da bazı gelişmiş arama operatörlerini desteklemektedir. Bu çalışmada bir domain adresine ait alt domainlerin elde edilebilmesi için bing arama motoru kullanılmıştır. Bing arama motorunun kullanılan ilk operatörü "ip" operatörüdür. Söz konusu işlemde öncelikle alt domaini bulunacak olan domain adresine ping atma işlemi gerçekleştirilmiştir. Böylelikle domain adresinin yayınlandığı web sucununun ip adresi elde edilmiştir. Daha sonra elde edilen ip adresi, bing operatörürün "ip" operatörü ile taranmış ve söz konusu ip adresinden yayını yapılan tüm alt domainler elde edilmeye çalışılmıştır. Gerçekleştirilen işlem, kendi web sunucusuna sahip kurumların adreslerini bulmada sorunsuz çalışırken, sunucu hizmetini satın almış olan kurumlarda olumsuz sonuç vermektedir. Şekil 4.4' de bilisimkazani.com adresi için söz konusu işlemin adımları gösterilmiştir.



Şekil 4.4. Bing üzerinden ip operatörü ile alt domain bulma işlemi

Şekilde görüldüğü gibi, ilgili adresin yayınının yapıldığı IP adresi ilk aşamada bulunmuştur. Fakat "ip" operatorü ile söz konusu IP adresi üzerinden yayınlanan web siteleri listelenmek istendiği zaman, aynı firma üzerinden sunucu hizmeti alan tüm web siteleri listelenmiştir. Bunun sebebi, aynı sunucu üzerinden birden fazla web sitesinin yayın yapmasıdır.

Gerçekleştirilen çalışma kapsamında daha sonra Bing arama motorunun "domain" operatörü kullanılmıştır. Bing arama motoru içerisinde "domain:sdu.edu.tr" şeklinde arama yapıldığında, sdu.edu.tr alt domainlerine ait kayıtların döndüğü görülmüştür. Gerçekleştirilen arama işlemine ait ekran görüntüsü aşağıdaki Şekil 4.5' de gösterilmiştir.



Şekil 4.5. Bing üzerinden domain operatörü ile alt domain bulma işlemi

Bing üzerinden subdomain bulma işleminde ziyaret edilecek URL adresi dinamik olarak oluşturulmakta ve sürekli olarak değiştirilmektedir.

- https://www.bing.com/search?q=domain%3Asdu.edu.tr&first=10

Yukarıdaki URL adresi incelendiğinde search?q=domain%3 ifadesinden sonra aranacak domain adres bilgisi yer almaktadır. &first= ifadesinden sonra ise arama sonunda dönen sayfada, ilk kaç kaydın gösterileceği bilgisi yer almaktadır. Geliştirilen subdomain bulma modülü ilgili URL kalıbı içerisine taranacak domain eklenmekte, daha sonra kayıtlar 10’ ar tane listelenecek şekilde, kayıt bulunamayana kadar güncellenmektedir. Geliştirilen subdomain bulma modülünün çalışma altyapısı çizelge 4.2’ de gösterilmiştir.

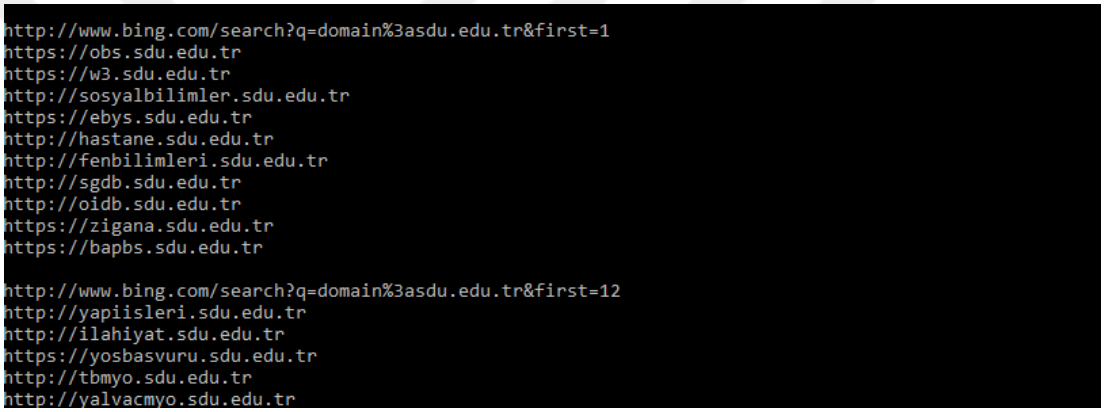
Çizelge 4.2. Subdomain bulma modülü çalışma adımları

No	Subdomain Toplama İşlemi Adımları
1	Subdomainleri toplanacak domain adının alınması
2	Domain adının Bing arama string kalıbı içerisine yerleştirilmesi
3	Oluşturulan dinamik URL adresine istek yapılması ve dönen sayfa kaynağının alınması
4	Sayfa kaynağı içerisindeki alt domain adreslerinin Pyton Beautifulsoup kütüphanesi yardımı ile ayıklanarak kaydedilmesi
5	Oluşturulan dinamik URL adresinin sonraki 10 kaydı gösterecek şekilde güncellenmesi ve tekrar istek atılması

Çizelge 4.2. Subdomain bulma modülü çalışma adımları (Devam)

6	İstek atılan yeni URL adresinden dönen sayfa kaynağının alınması ve 4. adımın tekrar edilmesi
7	5 numaralı adımın gösterilecek kaynak kalmayana kadar tekrar edilmesi

Geliştirilen modül kullanılarak gerçekleştirilen tarama ile alt domain adreslerinin elde edilmesinden sonra söz konusu adresler, geliştirilen modelin bulunduğu klasör içerisinde oluşturulmuş olan subDomain_Listesi.txt dosyasına kaydedilmektedir. Şekil 4.6' da geliştirilmiş olan aracın çalıştırılması ve çalışma zamanında alt domainlerin elde edilmesi işlemi gösterilmiştir.



Şekil 4.6. Geliştirilmiş subdomain bulma modülünün çalıştırılması

4.1.2.2. Web Zafiyet Tarayıcısı Kaba Kuvvet Modülü

Günümüzde web uygulama geliştiriciler, ziyaretçiler tarafından görülmesini istemedikleri admin paneli ya da robots.txt sayfası gibi sayfaları gizli tutmakta, site içerisinde link vermemektedirler. Uygulama geliştiriciler tarafından gizlenen bu sayfalar saldırganlar için büyük önem taşımaktadır. Çünkü admin paneline yapılacak izinsiz bir giriş tüm site yönetiminin zararlı ellere düşmesi anlamına gelmektedir. Oluşabilecek zararın boyutları düşünüldüğünde, söz konusu gizli sayfaların tespit edilmesi ve test edilmesi, güvenlik açısından büyük önem taşımaktadır. Gerçekleştirilen çalışma kapsamında geliştirilen web zafiyet tarayıcısının kapsam genişletme modülü içerisine, site içerisinde link verilmemiş gizli sayfaları brute force tekniği ile tespit eden bir modül eklenmiştir. Geliştirilen modül içerisinde yer alan,

brüte force tekniğinde kaynak olarak kullanılan wordlist' e ait ekran görüntüsü Şekil 4.7' de gösterilmektedir.

Şekil 4.7. Kaba kuvvet yönteminde kullanılan kelime listesi

Geliştirilen modülde test edilecek domain ve subdomain adreslerinin sonuna SANSAR ifadesi eklenmektedir. Daha sonra Şekil 4.7' de gösterilmekte olan liste içerisinde yer alan uzantılar, SANSAR terimi ile yer değiştirmektedir. Örneğin test edilen alt domain adresi; muhendislik.sdu.edu.tr/SANSAR ise ve listede test edilecek sıradaki kelime 'admin' ise test edilecek URL adresi; muhendislik.sdu.edu.tr/admin olmaktadır. Şekil 4.8' de söz konusu işlemin gerçekleştirildiği sınıfa ait kurucu metoda ait kodlar verilmiştir.

```
22  class bruteForceModule(Thread):
23      def __init__( self,word,url):
24          Thread.__init__(self)
25          try:
26              self.word = word.split("\n")[0]
27              self.urly = url.replace('SANSAR',self.word)
28              self.url = self.urly
29          except Exception as e:
30              print(e)
```

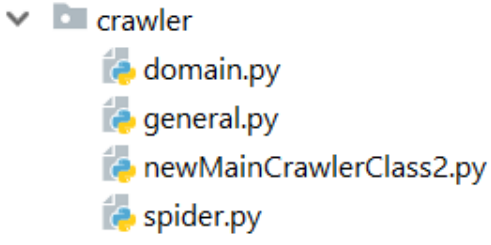
Şekil 4.8. bruteForceModule sınıfı kurucu metodu

Test edilecek URL adreslerinin oluşturulmasından sonra, söz konusu adreslerin geçerli birer adres olup olmadığı test edilmiştir. Bu işlem için Python requests kütüphanesi kullanılmıştır. Gerçekleştirilen işlemde oluşturulan URL adreslerine GET metodu kullanılarak bağlantı isteğinde bulunulmuştur. Bu işlemden sonra gerçekleştirilen bağlantı isteğinin durum kodu kontrol edilmiştir. Ziyaret edilen URL adreslerinin geçerli olması durumunda, 200 durum kodu elde edilmektedir. Böylelikle hangi URL adresinin geçerli olup hangisinin geçerli olmadığı tespit edilmektedir.

4.1.2.3. Web Zafiyet Tarayıcısı Örümcek (Crawler) Modülü

Bir web uygulamasına yönelik olarak sızma testi gerçekleştirileceği zaman, ana domain ve alt domainlerin listesi kimi durumlarda test uzmanı için yeterli olmayabilmektedir. Ana domain ve alt domainler içerisinde bulunmayan bir zafiyet, söz konusu domainler içerisinde yer alan alt adreslerde ortaya çıkabilmektedir. Proje kapsamında geliştirilen örümcek modülü; kendisine parametre olarak gönderilen URL adresinden başlayarak tüm web sayfasını baştan aşağıya gezmekte, gezdiği URL adreslerini kaydetmektedir. Böylelikle web sitesinin tüm haritası elde edilmektedir.

Örümcek modülü kendi içerisinde 4 farklı sınıftan oluşmaktadır. Şekil 4.9' da Örümcek (crawler) modülü sınıfları gösterilmiştir.



Şekil 4.9. Örümcek modülü sınıfları

Örümcek modülü içerisinde yer alan ana sınıf, newMainCrawlerClass.py sınıfıdır. Bu sınıf; diğer sınıfların çağırıldığı, toplanan URL adreslerinin text dosyasına kaydedildiği ana sınıftır. Şekil 4.10' da söz konusu sınıfa ait kurucu metoda ait kodlar verilmiştir.

```
def __init__(self,url,username,password,contentOfLoginPage,session,exception):
    self.isFinished=False
    self.PROJECT_NAME = 'scanReport'
    self.HOMEPAGE = url
    self.DOMAIN_NAME = get_domain_name(url)
    self.QUEUE_FILE = self.PROJECT_NAME + '/kuyruk.txt'
    self.CRAWLED_FILE = self.PROJECT_NAME + '/bulunan_linkler.txt'
    self.queue = Queue()
    self.general_headers = {'User-Agent': 'Mozilla/5.0 (Windows; U; Windows NT 6.1; rv:2.2) Gecko/20110201',
               'Accept-Language': 'en-US;',
               'Accept-Encoding': 'gzip, deflate',
               'Accept': 'text/html,application/xhtml+xml,application/xml;',
               'Connection': 'close'}
    # self.loginUrl = "http://testphp.vulnweb.com/login.php"
    self.session = session
    self.listOfLines=set()
    self.username=username
    self.password=password
    self.finalListOfURLs=set()
    self.contentOfLoginPage= contentOfLoginPage
    self.exception=exception
```

Şekil 4.10. Örümcek modülü ana sınıfı kurucu metodu

Örümcek nesnesinin web sitesini nasıl gezdiğine değinmeden önce parametre olarak aldığı oturum (session) nesnesine değinmek gerekmektedir. Bu tez çalışması kapsamında geliştirilen web zafiyet tarayıcısı, web sayfalarını gezerken ve kaydedilmiş URL adreslerine zafiyet testi yaparken aynı oturum nesnesini kullanmaktadır. Bunun sebebi, web uygulaması üzerinde o oturum nesnesi ile birkez oturum açıldığında, gerçekleştirilecek tüm bağlantıları o oturum ile gerçekleştirmektir.

Günümüzde kullanılan web uygulama tarayıcılar kullanıcı girişi gereken ve giriş sonrası görüntülenebilecek sayfaların gezilmesi ve test edilmesi için kullanıcılardan ilk olarak bir cookie bilgisi istemektedir. Kendilerine verilen cookie ile oturum elde eden araçlar, gezme ve test işlemlerini bu oturum üzerinden gerçekleştirmektedir. Fakat bu yöntemin bir dezavantajı bulunmaktadır. Kullanıcılar bir web sayfasını test etmeden önce ilk olarak o sayfayı kendi tarayıcılarından açmalı, kullanıcı girişi yapmalı ve elde ettikleri cookie değerini bir şekilde alarak programa vermelidir. Bu süreç hem uzun hem acemi kullanıcılar için zor, hem de hata yapmaya çok müsaittir.

Bu tez çalışması kapsamında geliştirilen web zafiyet tarayıcısına, kullanıcı girişinin gerektiği web uygulamalarını test ederken geçerli bir kullanıcı adı ve parola vermek

yeterlidir. Geliştirilen araç web uygulamasını sayfa sayfa gezerken kullanıcı girişi formu ile karşılaştığında, kendisine parametre olarak gönderilen kullanıcı adı ve parola bilgilerini forma girerek başarılı bir şekilde giriş yapmaktadır. Login sayfası üzerinde kullanıcı girişinin yapılması işleminde Python bünyesindeki Selenium Kütüphanesi kullanılmaktadır. Gerçekleştirilen tüm bağlantı işlemleri aynı oturum nesnesi üzerinden gerçekleştirildiği için tek sefer giriş yapıldıktan sonra aynı oturum ile tüm sayfalar başarılı bir şekilde ziyaret edilebilmekte ve test edilebilmektedir.

Geliştirilen araç bünyesindeki örümcek modülünün bir diğer avantajı, daha önce de bahsedildiği üzere sayfa üzerinde gördüğü formları varsayılan değerler ile doldurmasıdır. Örneğin, bir web sayfasında arama formunun olduğu kabul edilsin. Sayfa üzerindeki linkleri toplayan fakat formlar üzerinde bir işlem yapmayan bir örümcek bu formu dikkate almadan işlemine devam edecektir. Fakat bu çalışma kapsamında geliştirilmiş örümcek söz konusu form ile karşılaştığında, arama inputu içerisine bir değer girmekte ve formu submit etmekte yani göndermektedir. Bu işlemin adımları sırası ile şu şekilde gerçekleşecektir; ziyaret web sayfasına ait kaynak kodları elde edildikten sonra, kaynak kodlar içerisindeki <form> taglarını elde etmektedir. Form tagları elde edildikten sonra, ilk olarak forma ait ‘action’ değeri çekilerek bir değişkene atılmaktadır. Böylelikle POST isteği gönderilecek URL adresi elde edilmektedir. Daha sonra form içerisinde yer alan ‘<input> tagları elde edilmektedir. Böylelikle form içerisinde yer alan inputlara ait ‘name’, ‘type’ gibi özelliklere ait değerler kullanılabilmektedir. Form içerisindeki inputlar elde edildikten sonra her bir inputa türüne göre tanımlanmış varsayılan değerler atanmaktadır. Bu işlemi gerçekleştiren metoda ‘FormScraper’ ismi verilmiştir. FormScraper metodu tüm bu işlemlerden sonra örümcek modülüne ziyaret etmesi için bir POST nesnesi göndermektedir. Şekil 4.11’ de formlar içerisinde tespit edilen inputlara, türlerine göre atanacak varsayılan değerler gösterilmiştir.

```
defaults = {
    "checkbox": "default",
    "color": "fffffff",
    "date": "2019-02-02",
    "datetime": "2019-03-19T19:03:34.32",
    "datetime-local": "2019-03-19T19:03",
    "email": "sansar%40mailinator.com",
    "mail": "sansar%40mailinator.com",
    "file": ["pix.gif", "GIF89a", "image/gif"],
    "hidden": "default",
    "month": "2019-03",
    "number": "1337",
    "password": "parolametni",
    "radio": "beton",
    "range": "37",
    "search": "aramametni",
    "submit": "submit",
    "tel": "0555555555",
    "text": "textmetni",
    "time": "19:03",
    "url": "https://akademik.ahievran.edu.tr/site/mehmetaliyalcinkaya",
    "week": "2019-W03"
}
```

Şekil 4.11. Formlar içerisindeki inputlara atanacak varsayılan değerler

Formscraper metodu tarafından geriye döndürülen POST medotu, Python içerisindeki request kütüphanesi kullanılarak, mevcut session nesnesi aracılığı ile gönderilmektedir. Şekil 4.12' de geliştirilen formscraper aracının kullanımı sonrası üretilen POST metodları gösterilmiştir.

Şekil 4.12. FormScrapaer metodu tarafından POST isteklerinin üretilmesi

Şekil 4.12’ de gösterilen web sayfası incelendiğinde; sayfa içerisinde arama işlemi için oluşturulmuş formun olduğu görülmektedir. FormScrapaer aracına söz konusu URL adresi parametre olarak verildiğinde, çıktı olarak bir POST kalıbı döndürmüştür. Döndürülen POST kalıbı incelendiğinde, form içerisinde yer alan inputlara ait isimlerin belirlendiği ve inputların türlerine göre varsayılan değerlerin atandığı görülmektedir.

Geliştirilen web zafiyet tarayıcısı kapsam genişletme modülü içerisindeki oturum yönetimi ve otomatik form doldurma özelliklerine değindikten sonra bir web uygulamasının baştan aşağı nasıl örümcek tarafından gezildiğine değinmek gerekmektedir. Kapsam genişletme modülü içerisindeki ana sınıf, kendisine gönderilen URL, kullanıcı adı, parola gibi bilgileri, kapsam genişletme modülü içerisindeki bir diğer sınıf olan spider.py sınıfına göndermektedir.

Spider.py sınıfı ilk olarak kendisine parametre olarak gönderilen başlangıç sayfasına bağlanmaktadır. Başlangıç sayfasına gerçekleştirilen bağlantı sonrasında yapılan ilk işlem, sayfa kaynağını almaktır. Ziyaret edilen sayfaya ait kaynak kodlar bir Beautifulsoup nesnesine dönüştürülerek, sayfa kaynağı içerisindeki URL adreslerinin ve form bilgilerinin kazınmasını sağlayan “extractURLsFromSoup” isimli bir metoda gönderilmektedir.

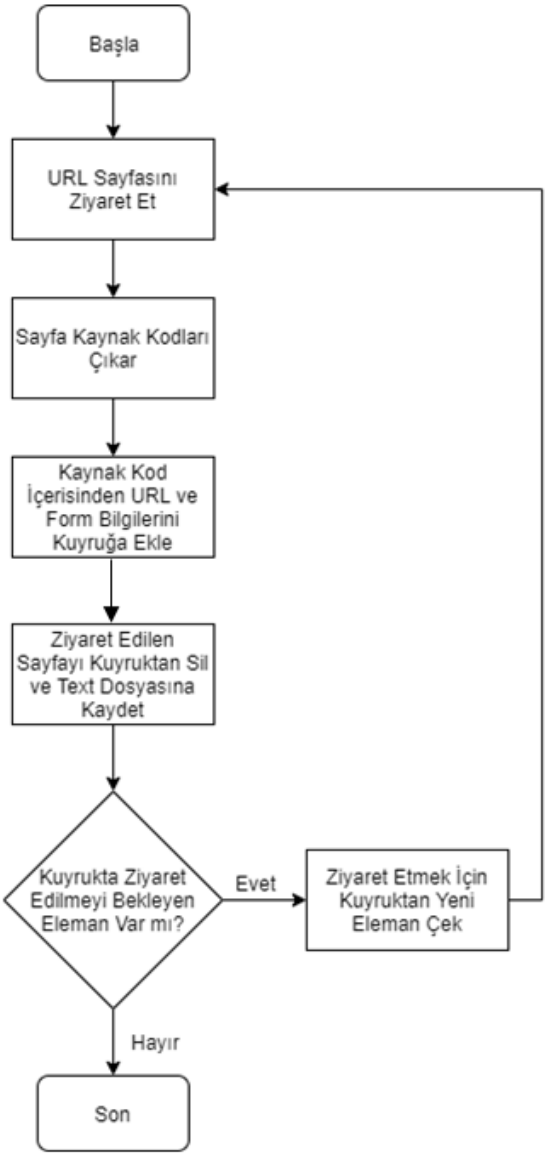
extractURLsFromSoup methodu, kendisine gönderilen sayfa kaynağı içerisinde ilk olarak URL toplama işlemi gerçekleştirmektedir. Bu işlemde, sayfa kaynağında yer alan “script”, “iframe”,” a”, “frame”, “iframe”, “button” ve “meta” HTML tagları çıkartılmaktadır. Bu adımdan sonra söz konusu taglar içerisinde yer alan; “src”, “href”, “action”, “hhtp-equiv” taglarına atanan URL adresleri toplanmaktadır. Toplanan URL adreslerinin gezilen sayfa domainine ait olup olmadığı kontrol edildikten sonra ziyaret edilecek URL adresleri listesine eklenmektedir. Bu işlem dizisi sonunda bir web sayfası içerisinde verilmiş tüm linkler başarı ile toplanmaktadır.

extractURLsFromSoup metodunun yaptığı ikinci işlem, sayfa kaynağını daha önce bahsedilen FormScraper methoduna yollamaktır. FormScraper metodu tarafından kaynak kodları parçalanacak, sayfa içerisindeki formlar çıkartılıp varsayılan değerler ile doldurulacak ve geriye POST istek metodları olarak döndürülecektir. Sayfa

kaynağından kazıdığı URL adreslerine, FormScraper metodundan gönderilen POST metodlarını da ekleyen extractURLsFromSoup metodu, elindeki ziyaret edilecekler listesini geriye döndürmektedir.

Spider.py sınıfı bünyesindeki extractURLsFromSoup metodundan geriye döndürülen, ziyaret edilecek URL ve istek atılacak POST nesneleri listesi kuyruğa eklendikten sonra, bu verileri elde etmesini sağlayan URL adresini bir text dosyasına kaydetmektedir. Böylelikle ziyaret edilip kaynak kodu parçalanan bir adresin tekrar ziyaret edilmesi engellenmektedir. Bu işlemden sonra Spider.py sınfı elindeki kuyruktan bir elemanı çekerek, aynı ziyaret et- kaynak kodu al- parçala- URL ve POST çıkar işlemlerini tekrar etmektedir. Bu işlem döngüsü kuyrukta ziyaret edilecek eleman kalmayana kadar devam etmektedir. Bu sayede bir web uygulaması içerisindeki tüm sayfalar tek tek ziyaret edilecektir.

Spider.py sınıfında değinilmesi gereken son konu, sayfalar üzerinde tespit edilen formlar varsayılan değerler ile doldurulduktan sonra nasıl submit edildiğidir. Spider sınıfı sayfaları gezerken bir form ile karşılaştığında, ilgili form daha önce de anlatıldığı gibi varsayılan değerler ile doldurulmaktadır. Bu adımdan sonra Python Requests Kütüphanesi kullanılarak doldurulan formun URL adresine bir POST isteği gönderilmektedir. İlgili istek gönderiliken data parametresine hazırlanan POST verisi eklenmektedir. Şekil 4.13’ de bu başlık altında işlenen örümcek modülünün çalışması özetlenmiştir.



Şekil 4.13. Örümcek modülü çalışma diagramı

Bu çalışma kapsamında geliştirilen web zafiyet tarayıcısının sahip olduğu kapsam genişletme modülü; web güvenliği alanında yaygın olarak kullanılan zafiyet tarayıcıların kapsam genişletme modülleri ve özellikle web uygulamalarındaki URL adreslerinin çıkarılması için geliştirilmiş crawler araçlar ile karşılaştırılmıştır. Söz konusu işlemde kullanılan ilk araç; sızma testlerinde en yaygın kullanılan araçlardan olan Metasploit' tir. Gerçekleştirilen test işleminde Metasploit içerisinde yer alan 'msfcrawler' modülü kullanılmıştır. Söz konusu karşılaştırma işlemi, web uygulama sızma testlerinde yaygın olarak kullanılan ve açık kaynak kodlu zafiyet tarayıcılardan olan WASCAN, Wapiti ve Skipfish ile gerçekleştirilmiştir. Bunun yanında web uygulama sızma testlerinde kapsam belirleme amacıyla yaygın olarak kullanılmakta olan Httrack ve BlackWidow araçları da karşılaştırma işlemine dahil edilmiştir.

Gerçekleştirilen işlemde her bir araç ile ‘http://testphp.vulnweb.com’ adresi parametre olarak verilmiş ve söz konusu adresden yola çıkarak tüm URL adreslerinin bulunması istenmiştir. Gerçekleştirilen tarama işlemleri sonrasında, bu çalışma kapsamında geliştirilen web zafiyet tarayıcısının kapsam genişletme modülü, Httrack aracı ile aynı sayıda ve en çok URL adresini tespit etmiştir. Çizelge 4.3’ de kullanılan araçlar ve tespit ettikleri URL adres sayıları gösterilmektedir.

Çizelge 4.3. Kapsam genişletmede kullanılan araçlar ve performans verileri

Kullanılan Araç	URL Adres Sayısı
Sansar ve Httrack	75
Metasploit (msfcrawler)	66
Wapiti	57
Skipfish	50
BlackWidow	40
Wascan	12

4.1.3. Web Zafiyet Tarayıcısı Zafiyet Test Modülü

Bu tez çalışması kapsamında geliştirilen “Dinamik Analiz ve Yapay Zekâ Tabanlı Web Zafiyet Tarayıcısı” aracının bir diğer önemli modülü; zafiyet test modülüdür. Zafiyet test modülü ilk olarak ana modül tarafından çalıştırılan kapsam genişletme modülü tarafından toplanmış test edilecek URL adreslerinin listesini parametre olarak almaktadır. Bunun yanında test işlemlerinde gerçekleştirilecek bağlantılarda kullanılmak üzere, kapsam genişletme modülü tarafından da kullanılmış olan oturum (session) nesnesini de parametre olarak almaktadır. Bu sayede kapsam genişletme modülü tarafından oturum açıldıysa, bu oturum nesnesi sayesinde test edilecek tüm sayfalar başarılı bir şekilde görüntülenebilecektir. Eğer zafiyet test modülünde yeni bir oturum nesnesi kullanılsaydı, kapsam genişletme modülü tarafından kullanıcı girişi yapıldıktan sonra ziyaret edilip listeye eklenen URL adresleri, zafiyet testleri için yeni bir oturum nesnesi ile tekrar ziyaret etmeye çalışıldığında, sayfalar görüntülenemeyecekti. Çünkü yeni nesne kullanıcı giriş yetkisine sahip olmayacaktı.

Zafiyet test metodu tarafından alınan bir diğer parametre ise, kullanıcı adı ve parola parametresidir. Zafiyet test işlemi için yazılmış olan her bir sınıf, ilk olarak tarayacağı URL adresine erişip erişemediğini kontrol etmektedir. Eğer ziyaret etmesi gereken URL adresine gitmeye çalışırken web sayfası aracı kullanıcı girişine yönlendiriyorsa, bu bazı sebeplerden dolayı oturumun sonlandığı anlamına gelmektedir. Web zafiyet tarayıcısı böyle bir durumda karşılaştığında, kullanıcı adı ve parola parametrelerini kullanarak sisteme tekrar kullanıcı girişi yapmakta, böylelikle taraması gereken URL adresine ulaşarak başarılı taramalar gerçekleştirebilmektedir.

Geliştirilen zafiyet test modülü testlerini, kullanıcının tercihine göre GET, POST ya da her iki metod üzerinden gerçekleştirmektedir. Her bir zafiyet için yazılmış test sınıflarına geçmeden önce; zafiyet testlerinin GET ve POST metodları üzerinden nasıl gerçekleştirildiğine detaylı olarak değinmek gerekmektedir.

GET metodunda formlara girilen veriler, sunucuya gönderilirken adres çubuğunda görünmektedirler. Aşağıda GET metodu kullanımının bir örneği gösterilmektedir.

- http://www.teztest.com/sayfa.php?kullaniciNo=veri1&parola=veri2

Yukarıda örnek olarak gösterilen URL adresi için zafiyet testi yapılacağı düşünüldüğünde ilk olarak, URL içerisinde yer alan kullaniciNo ve parola değişkenlerinin tespit edilmesi gerekmektedir. Çünkü bu iki değişkene atanan değerler (bu örnekte veri1 ve veri2 değerleri) veri tabanında çalıştırılacak sorgu içerisine eklenmektedir. Sorgu içerisine eklenen değerlerin herhangi bir filtrelemeden ve kontrolden geçmeden sorguya eklenmesi durumunda çeşitli zafiyetler ortaya çıkmaktadır. Bu nedenle zafiyet test modülü geliştirilirken ilk olarak URL adresi içerisinde yer alan değişkenlerin ve değerlerin elde edilmesi işlemi kodlanmıştır.

Gerçekleştirilen işlemde URL içerisinde yer alan değişken ve değerler tek tek ayrıştırılmaktadır. Daha sonra değişkenlere sırası ile SANSAR terimi eklenmektedir. Bu işlem bir dizi döngü ve koşul içerisinde gerçekleştirildiği için tüm değişkenlere sırası ile SANSAR anahtar kelimesi atanmış olacaktır. Yukarıda örnek verilmiş URL adresi ele alınırsa ayrıştırma sonunda;

- http://www.teztest.com/sayfa.php?kullaniciNo=SANSAR&parola=veri2
- http://www.teztest.com/sayfa.php?kullaniciNo=veri1&parola=SANSAR

URL adresleri elde edilecektir. Böylelikle ilgili URL adresinde bir zafiyet tespit edilirse, zafiyetin hangi değişkende tespit edildiği de kolaylıkla görülecektir. Şekil 4.14'de URL içerisindeki değişkenleri tespit ederek, bu değişkenlere sırası ile 'SANSAR' ifadesini atayan metodun bir kısmı gösterilmiştir.

```
if "=" in urlAdress:
    try:
        parsedURL = urllib.parse.urlparse(urlAdress)  # url parse modülü ile url adre
        parametersOfURL = urllib.parse.parse_qsl(parsedURL.query)  # ayrılmış URL' in
        listOfParameters = []
        listOfValues = []
        countOfParameters = 0
        for params, values in parametersOfURL:
            listOfParameters.extend([str(params + "=")])
            listOfValues.extend([str(urllib.parse.quote_plus(values))])
            countOfParameters = countOfParameters + 1
        dynamic_url = urlAdress.find("?")
        mainUrlAdress = str(urlAdress[:dynamic_url + 1])  # sayfaya ait url adresi de
        activeVariable = 1
        i = 1
        while i <= countOfParameters and activeVariable <= countOfParameters:
            # döngü içerisinde yapılan işlem içerisinde URL içerisinde yer alan param
            if (i < countOfParameters and i == activeVariable):
                # print("Parameter currently being tested: " + listOfParameters[i - 1
                mainUrlAdress += listOfParameters[i - 1] + payload + "&"  # url adres
                i = i + 1  # sonraki parametrelerin de test edilmesi için i bir ötele

            elif (i == countOfParameters and i == activeVariable):
                mainUrlAdress += listOfParameters[i - 1] + payload
                activeVariable = activeVariable + 1
                i = i + 1
                urlsWillBeExamined.add(mainUrlAdress)
                mainUrlAdress = str(urlAdress[:dynamic_url + 1])  # yüklenen payload

            elif (i == countOfParameters and i != activeVariable):
                mainUrlAdress += listOfParameters[i - 1] + listOfValues[i - 1]
                activeVariable = activeVariable + 1
                i = 1
                urlsWillBeExamined.add(mainUrlAdress)
                mainUrlAdress = str(urlAdress[:dynamic_url + 1])

            elif (i == countOfParameters):
                mainUrlAdress += listOfParameters[i - 1] + listOfValues[i - 1]
                activeVariable = activeVariable + 1
                i = 1
                urlsWillBeExamined.add(mainUrlAdress)
                mainUrlAdress = str(urlAdress[:dynamic_url + 1])
            else:
                # print "Rebuilding Original Parameter Values"
                mainUrlAdress += listOfParameters[i - 1] + listOfValues[i - 1] + "&"
                i = i + 1
    except:
```

Şekil 4.14. getParser metodu içerisinde bir kod bloğu

Bu tez kapsamında geliştirilen web zafiyet tarayıcısı içerisinde 21 farklı türde web zafiyeti test edilmektedir. Her bir zafiyet türü için ayrı birer zafiyet test sınıfı oluşturulmuştur. Geliştirilen her bir sınıf, GET üzerinden test işlemleri gerçekleştireceği zaman, ilk olarak yukarıda anlatılan işlemi gerçekleştirmekte, daha sonra URL adresi içerisindeki SANSAR terimini, test edilecek zafiyete özgü payloadlar ile yer değiştirerek test işlemini başlatmaktadır. Örneğin, aşağıdaki URL adresi üzerinde hata tabanlı SQL enjeksiyonunun test edildiği düşünülsün;

- http://www.teztest.com/sayfa.php?kullaniciNo=veri1&parola=SANSAR

Hata tabanlı SQL enjeksiyonu payloadlarından olan ' or '1=1 terimi SANSAR terimi ile yer değiştirildiğinde URL adresinin son hali aşağıdaki gibi olmaktadır.

- http://www.teztest.com/sayfa.php?kullaniciNo=veri1&parola=' or '1=1

Bu durumda parola değişkenine ' or '1=1 değeri atanmaktadır. Eğer bu değer herhangi bir filtrelemeden geçmeden veri tabanında çalıştırılacak sorgu içerisine eklenirse SQL enjeksiyonu zafiyeti tespit edilebilecektir.

Günümüzde birçok web uygulamasında URL kısaltmak, arama motorları tarafından daha kolay indekslenmek ve güvenlik seviyesini arttırmak gibi sebeplerden dolayı URL rewrite tekniği kullanılmaktadır. Bu çalışma kapsamında geliştirilen web zafiyet tarayıcısı, URL rewrite kullanılmış web sayfalarında GET üzerinden zafiyet testlerini de başarı ile gerçekleştirebilmektedir. Aşağıda URL rewrite tekniği ile oluşturulmuş bir URL adresi gösterilmektedir.

- http://www.teztest.com/sayfa.php/kullaniciNo/veri1

Yukarıda gösterilmekte olan URL adresi incelendiğinde URL içerisinde onu parçalamaya ve değişkenleri tespit etmeye yarayan '?', '=' gibi işaretler bulunmamaktadır. Bu tür URL adreslerinin test edilmesinde; adres / karakterine göre parçalanmalı, domainden sonra gelen her çift / karakter arasına payload gömülmelidir. Geliştirilen araç URL rewrite ile oluşturulmuş bir URL adresi ile karşılaştığında ilk olarak / karakterlerini tespit ederek, çift / karakterlerin arasını SANSAR terimi ile değiştirmektedir. Yukarıda gösterilen URL adresi bu işleme tabi tutulduğunda elde edilen sonuçlar aşağıdaki gibi olacaktır.

- http://www.teztest.com/sayfa.php/SANSAR/veri1
- http://www.teztest.com/sayfa.php/kullaniciNo/SANSAR

Bu adımdan sonra her bir sonuç içerisindeki SANSAR ifadeleri test sınıfı tarafından payload ile değiştirilerek test işlemleri başlatılmaktadır. Aşağıda, verilen örneğin hata tabanlı SQL enjeksiyonu payloadları ile değiştirilmesi gösterilmiştir.

- http://www.teztest.com/sayfa.php/kullaniciNo/' or '1=1
- http://www.teztest.com/sayfa.php/' or '1=1/veri1

Şekil 4.15’ de rewrite metodu ile oluşturulmuş URL adreslerinin parçalanması ve içerisine SANSAR terimlerinin eklenmesi işlemini yapan kod bloğu gösterilmiştir.

```
parsedURL = urllib.parse.urlparse(urlAdress)  # url parse modülü ile url adresi scheme netloc gibi pa
baseURL = parsedURL.scheme + "://" + parsedURL.netloc
path = parsedURL.path

if path == '' or path == '/':
    mainUrlAdress = baseURL + '/SANSAR'
    urlsWillBeExamined.add(mainUrlAdress)

else:
    indexesOfSlash = []
    for index in range(0, len(path)):
        if path[index] == '/':
            indexesOfSlash.append(index)

    counterOfSlash = 0
    numberOfSlash = len(indexesOfSlash)
    while counterOfSlash < numberOfSlash:
        if counterOfSlash == numberOfSlash - 1:
            newURL = baseURL + path.replace(path[indexesOfSlash[counterOfSlash] + 1:], 'SANSAR')
            urlsWillBeExamined.add(newURL)
            break
        else:
            newURL = baseURL + path.replace(path[indexesOfSlash[counterOfSlash]
                                                 + 1: indexesOfSlash[counterOfSlash + 1]], 'SANSAR')
            urlsWillBeExamined.add(newURL)
            counterOfSlash = counterOfSlash + 1
```

Şekil 4.15. getParser metodu içerisinde URL rewrite parçalama işlemi

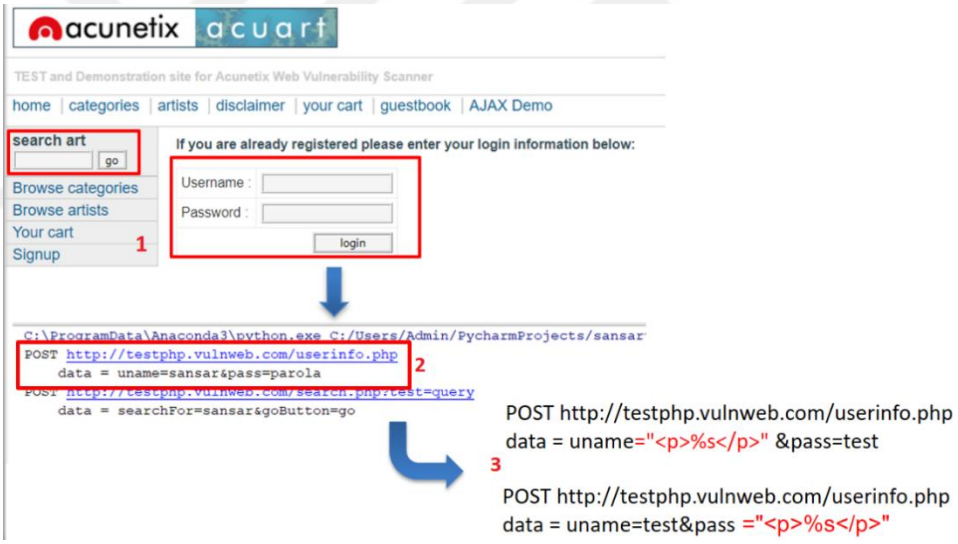
GET metodu üzerinden gerçekleştirilen testlerin mantığının açıklamasından sonra, POST metodu üzerinden gerçekleştirilen testlerin açıklanması gerekmektedir. POST üzerinden test gerçekleştirme ifadesinin anlamı, ziyaret edilen web sayfası üzerindeki form ve inputların test edilmesidir. Bir web sayfası üzerindeki form ve inputların zafiyet testinin nasıl yapıldığı, “Materyal ve Yöntem” başlığının “Web Uygulama Sızma Testleri” alt başlığında anlatılmıştır. Geliştirilen web zafiyet tarayıcısı tarafından bir URL adresindeki form ve inputlar test edileceğinde ilk olarak, ilgili sınıfa parametre olarak gönderilen URL adresi, formScraper metoduna parametre olarak gönderilmektedir. formScarper metodu web sayfası içerisinde yer alan formları tespit ederek geriye formlar tarafından gönderilecek POST kalıplarını döndürmektedir. Bu adımdan sonra, geliştirilen aracın payloads klasörü altında yer alan payloads.py sınıfı içerisindeki zafiyete özgü payloadlar bir döngü yardımı ile çekilmektedir. Daha sonra çekilen payload, formScraper metodu tarafından döndürülen POST kalıbı içerisinde yer alan ‘sansar’ değerleri ile sırayla yer değiştirmektedir. Örnek üzerinden gidilecek olursa, formScraper tarafından aşağıdaki post kalıbı gönderilmiş olsun;

- POST http://testphp.vulnweb.com/userinfo.php
  data = uname=mali&pass=parola

ise ve listeden çekilen payload; " %s " ise, gerçekleştirilen yer değiştirme işlemi sonrasında üretilen POST kalıpları;

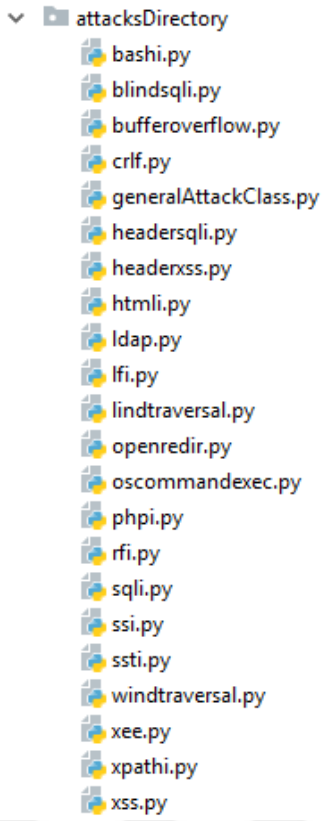
- POST http://testphp.vulnweb.com/userinfo.php
  data = uname=" %s " &pass=parola
- POST http://testphp.vulnweb.com/userinfo.php
  data = uname=mali&pass=" %s "

olacaktır. Daha sonra her bir POST kalıbı ve sınıfa parametre olarak gönderilen session nesnesi kullanılarak POST isteği gönderilmektedir. Tıpkı GET üzerinden yapılan testlerde her bir değişkenin ayrı ayrı test edilebildiği gibi, POST üzerinden gerçekleştirilen testlerde de sayfada yer alan tüm input alanları (hidden türünde olanlar da dahil) tek tek test edilmektedir. Şekil 4.16' da test edilen sayfa üzerindeki formların tespit edilmesi (1), FormScraper metodu tarafından POST kalıplarının oluşturulması (2), oluşturulan kalıplara sırayla payloadların eklenmesi (3) gösterilmiştir.



Şekil 4.16. FormScraper methodu ile POST isteklerinin üretilmesi ve test edilmesi

Bu tez çalışması kapsamında geliştirilen web zafiyet tarayıcısı bünyesinde toplam 21 farklı zafiyet test sınıfı bulunmaktadır. Şekil 4.17' de zafiyet test modülü içerisinde yer alan sınıflar gösterilmiştir.



Şekil 4.17. Zafiyet test modülü sınıfları

Bu tez kapsamında geliştirilmiş zafiyet test sınıfları içerisinde; bir tane kurucu metod, GET üzerinden testlerin yapılmasını sağlayan get isminde bir metod, POST üzerinden testlerin yapılmasını sağlayan post isminde bir metod, oturum kopması durumunda login sayfasına giderek tekrar oturum açma işlemi gerçekleştiren loginProcess isminde bir metod ve son olarak hangi durumlarda hangi metodun çalışacağını belirten go isminde bir metod içermektedir.

Şekil 4.17' de gösterilen her bir zafiyetin mantığı, neden ortaya çıktığı, ne gibi sonuçlara neden olduğu gibi bilgiler "Materyal ve Yöntem" başlığı altında verildiği için, bu başlık altında tekrar değinilmeyecektir. Bu bölümde sadece Şekil 4.17' de gösterilen sınıfların çalışma mantığı ve iş akışı incelenecektir.

Geliştirilen zafiyet test modülü içerisindeki **bashi.py** sınıfı **Shellshock** zafiyetinin test edilmesinde kullanılmaktadır. İlgili sınıfın işleyişi Çizelge 4.4' te gösterilmektedir.

Çizelge 4.4. Shellshock zafiyeti test sınıfı işleyişi

No	Shellshock zafiyeti test adımları
1	Parametre olarak gönderilen test edilecek URL, oturum, login sayfa adresi, kullanıcı adı, parola değerleri alınır ve sınıf değişkenlerine atanır.
2	Test edilecek sayfaya ulaşılmaya çalışılır. Ulaşılmak istenen sayfa kullanıcı giriş sayfasına yönlendiriyorsa, kullanıcı girişi yapılarak test işlemleri başlatılır.
3	Payloads.py dosyası içerisinden Shellshock payloadları alınır.
4	Alınan payloadlar bir test header verisine enjekte edilir.
5	Oluşturulan test headerı kullanılarak sayfaya get isteği gönderilir.
6	Gönderilen isteğe karşılık gelen yanıt içerisinde zafiyet olduğunu ispatlayan string verisi aranır.
7	Eğer zafiyet bulunduysa, zafiyet adı, bulunduğu adres, oturum gibi veriler öznitelik çıkarma modülüne gönderilir.
8	Öznitelik çıkarma modülünden gönderilen öznitelikler raporlama sınıfına gönderilir.

Geliştirilen web zafiyet tarayıcısı içerisinde yer alan bir diğer zafiyet test sınıfı **sqli.py** sınıfıdır. Bu sınıf adından da anlaşılacağı üzere web sayfalarında hata tabanlı **SQL enjeksiyonu** aramaktadır. Çizelge 4.5' de söz konusu sınıfın işleyişi gösterilmiştir.

Çizelge 4.5. Hata tabanlı SQL enjeksiyonu zafiyeti test sınıfı işleyişi

No	Hata tabanlı SQLi zafiyeti test adımları
1	Parametre olarak gönderilen test edilecek URL, oturum, login sayfa adresi, kullanıcı adı, parola değerleri alınır ve sınıf değişkenlerine atanır.
2	Kullanıcı isteğine göre get ve post metodları çalıştırılır. (Her iki metodun işleyiş mantığı aynı olduğu için tek süreç olarak anlatılacaktır.)
3	Test edilecek sayfaya ulaşılmaya çalışılır. Ulaşılmak istenen sayfa kullanıcı giriş sayfasına yönlendiriyorsa, kullanıcı girişi yapılarak test işlemleri başlatılır.
4	Payloads.py dosyası içerisinden SQL enjeksiyonu payloadları alınır.

Çizelge 4.5. Hata tabanlı SQL enjeksiyonu zafiyeti test sınıfı işleyişi (Devam)

5	Errors klasörü içerisinden enjeksiyon sonucu görüntülenmesi muhtemel hata listesi alınır.
6	Alınan payloadlar eğer get metodu çalışıyorsa 'getParser' metodu tarafından hazırlanan URL kalıplarına, eğer post metodu çalışıyorsa 'FormScraper' metodu tarafından hazırlanan POST kalıplarına enjekte edilir.
7	Oluşturulan URL ya da POST kalıpları kullanılarak istek gönderilir.
8	Gönderilen isteğe karşılık gelen yanıt içerisinde zafiyet olduğunu ispatlayan hata mesajlarının olup olmadığı kontrol edilir.
9	Eğer dönen yanıt içerisinde hata mesajı ile karşılaşıldıysa; zafiyet adı, bulunduğu adres, oturum gibi veriler öznitelik çıkarma modülüne gönderilir.
10	Öznitelik çıkarma modülünden gönderilen öznitelikler raporlama sınıfına gönderilir.

Hata tabanlı SQL enjeksiyonu zafiyetinden sonra, zafiyet tarama modülü içerisinde geliştirilen bir diğer sınıf **blindsqli.py** sınıfıdır. Bu sınıf web sayfaları üzerinde **zaman tabanlı kör SQL enjeksiyon** testini gerçekleştirmektedir. Çizelge 4.6' da söz konusu sınıfın işleyişi gösterilmiştir.

Çizelge 4.6. Zaman tabanlı kör SQL enjeksiyonu zafiyeti test sınıfı işleyişi

No	**Zaman tabanlı kör SQL enjeksiyonu zafiyeti test adımları**
1	Parametre olarak gönderilen test edilecek URL, oturum, login sayfa adresi, kullanıcı adı, parola değerleri alınır ve sınıf değişkenlerine atanır. Varsayılan timeout süresi 3 saniye olarak belirlenir.
2	Kullanıcı isteğine göre get ve post metodları çalıştırılır. (Her iki metodun işleyiş mantığı aynı olduğu için tek süreç olarak anlatılacaktır.)
3	Test edilecek sayfaya ulaşılmaya çalışılır. Ulaşılmak istenen sayfa kullanıcı giriş sayfasına yönlendiriyorsa, kullanıcı girişi yapılarak test işlemleri başlatılır.
4	Payloads klasörü içerisindeki "blindSQLipayloads.txt" dosyası içerisindeki test payloadları alınır.

Çizelge 4.6. Zaman tabanlı kör SQL enjeksiyonu zafiyeti test sınıfı işleyişi (Devam)

5	Enjekte edilecek payloadlar içerisine uyuma süresi olarak 10 saniye değeri eklenir.
6	Alınan payloadlar eğer get metodu çalışıyorsa 'getParser' metodu tarafından hazırlanan URL kalıplarına, eğer post metodu çalışıyorsa 'FormScraper' metodu tarafından hazırlanan POST kalıplarına enjekte edilir.
7	Oluşturulan URL ya da POST kalıpları kullanılarak istek gönderilir.
8	Gönderilen isteğe sonrasında yanıt dönmesi için geçen süreye bakılır. Eğer geçen süre, daha önce belirlenen timeout süresinden (3 Saniye) uzunsa zafiyet tespit edilmiş anlamına gelmektedir.
9	Eğer zafiyet tespit edildiyse; zafiyet adı, bulunduğu adres, oturum gibi veriler öznitelik çıkarma modülüne gönderilir.
10	Öznitelik çıkarma modülünden gönderilen öznitelikler raporlama sınıfına gönderilir.

Geliştirilen zafiyet test modülü içerisindeki bir diğer SQL enjeksiyonu test sınıfı **headersqli.py** sınıfıdır. Bu sınıf, "Materyal ve Yöntem başlığı altında anlatılmış "**HTTP Header tabanlı SQL Enjeksiyonu**" zafiyetinin test edilmesinde kullanılmaktadır. İlgili sınıfın işleyişi Çizelge 4.7' de gösterilmiştir.

Çizelge 4.7. Header SQL enjeksiyonu zafiyeti test sınıfı işleyişi

No	**Header SQL enjeksiyonu zafiyeti test adımları**
1	Parametre olarak gönderilen test edilecek URL, oturum, login sayfa adresi, kullanıcı adı, parola değerleri alınır ve sınıf değişkenlerine atanır.
2	Test edilecek sayfaya ulaşılmaya çalışılır. Ulaşılmak istenen sayfa kullanıcı giriş sayfasına yönlendiriyorsa, kullanıcı girişi yapılarak test işlemleri başlatılır.
3	Errors klasörü içerisinden enjeksiyon sonucu görüntülenmesi muhtemel hata listesi alınır.
4	Payloads.py dosyası içerisinden SQL enjeksiyonu payloadları alınır.

Çizelge 4.7. Header SQL enjeksiyonu zafiyeti test sınıfı işleyişi (Devam)

5	Alınan payloadlar bir test header içerisindeki ‘Cookie’, ‘Referer’ ve ‘User-Agent’ alanlarına sırası ile enjekte edilir.
6	Oluşturulan test headerları kullanılarak sayfaya get istekleri gönderilir.
7	Gönderilen isteğe karşılık gelen yanıt içerisinde zafiyet olduğunu ispatlayan hata mesajlarının olup olmadığı kontrol edilir.
8	Eğer dönen yanıt içerisinde hata mesajı ile karşılaşıldıysa; zafiyet adı, bulunduğu adres, oturum gibi veriler öznitelik çıkarma modülüne gönderilir.
9	Öznitelik çıkarma modülünden gönderilen öznitelikler raporlama sınıfına gönderilir.

Geliştirilen zafiyet test modülü içerisindeki bir diğer test sınıfı **bufferoverflow.py** sınıfıdır. Bu sınıf, adından da anlaşılacağı üzerine **bellek taşması (buffer overflow)** zafiyetinin test edilmesinde kullanılmaktadır. İlgili sınıfın işleyişi Çizelge 4.8’ de gösterilmiştir.

Çizelge 4.8. Bellek taşması zafiyeti test sınıfı işleyişi

No	**Bellek taşması zafiyeti test adımları**
1	Parametre olarak gönderilen test edilecek URL, oturum, login sayfa adresi, kullanıcı adı, parola değerleri alınır ve sınıf değişkenlerine atanır.
2	Kullanıcı isteğine göre get ve post metodları çalıştırılır. (Her iki metodun işleyiş mantığı aynı olduğu için tek süreç olarak anlatılacaktır.)
3	Test edilecek sayfaya ulaşılmaya çalışılır. Ulaşılmak istenen sayfa kullanıcı giriş sayfasına yönlendiriyorsa, kullanıcı girişi yapılarak test işlemleri başlatılır.
4	Errors klasörü içerisinden test sonrası görüntülenmesi muhtemel hata listesi alınır.
5	Metodlar içerisinde tanımlanan payloadlar eğer get metodu çalışıyorsa ‘getParser’ metodu tarafından hazırlanan URL kalıplarına, eğer post metodu çalışıyorsa ‘FormScraper’ metodu tarafından hazırlanan POST kalıplarına enjekte edilir.

Çizelge 4.8. Bellek taşması zafiyeti test sınıfı işleyişi (Devam)

6	Oluşturulan URL ya da POST kalıpları kullanılarak istek gönderilir.
7	Gönderilen isteğe karşılık, gelen yanıt içerisinde zafiyet olduğunu ispatlayan hata mesajlarının olup olmadığı kontrol edilir.
8	Eğer dönen yanıt içerisinde hata mesajı ile karşılaşıldıysa; zafiyet adı, bulunduğu adres, oturum gibi veriler öznitelik çıkarma modülüne gönderilir.
9	Öznitelik çıkarma modülünden gönderilen öznitelikler raporlama sınıfına gönderilir.

Bu tez çalışması kapsamında geliştirilen web zafiyet tarayıcısı içerisindeki bir diğer zafiyet test sınıfı; **Carriage return- line feed (CRLF)** zafiyetinin test edilmesi için yazılmış olan **crlf.py**' dir. Bu sınıf da benzer diğer sınıflarda olduğu gibi payloadı ilgili yerlere enjekte ettikten sonra yapılan isteklere dayanmaktadır. Çizelge 4.9' de CLRF zafiyeti test adımları gösterilmiştir.

Çizelge 4.9. CLRF zafiyeti test sınıfı işleyişi

No	**CLRF zafiyeti test adımları**
1	Parametre olarak gönderilen test edilecek URL, oturum, login sayfa adresi, kullanıcı adı, parola değerleri alınır ve sınıf değişkenlerine atanır.
2	Kullanıcı isteğine göre get ve post metodları çalıştırılır. (Her iki metodun işleyiş mantığı aynı olduğu için tek süreç olarak anlatılacaktır.)
3	Test edilecek sayfaya ulaşılmaya çalışılır. Ulaşılmak istenen sayfa kullanıcı giriş sayfasına yönlendiriyorsa, kullanıcı girişi yapılarak test işlemleri başlatılır.
4	Payloads.py dosyası içerisinden CLRF zafiyeti payloadları alınır.
5	Alınan payloadlar eğer get metodu çalışıyorsa 'getParser' metodu tarafından hazırlanan URL kalıplarına, eğer post metodu çalışıyorsa 'FormScraper' metodu tarafından hazırlanan POST kalıplarına enjekte edilir.

Çizelge 4.9. CLRF zafiyeti test sınıfı işleyişi (Devam)

6	Oluşturulan URL ya da POST kalıpları kullanılarak istek gönderilir.
7	Gönderilen isteğe karşılık gelen yanıtın headerının Set-Cookie alanı içerisinde enjekte edilen payload aranır.
8	Eğer 8.maddede belirtilen koşul sağlandıysa zafiyet bulundu anlamına gelmektedir. Bu durumda; zafiyet adı, bulunduğu adres, oturum gibi veriler öznitelik çıkarma modülüne gönderilir.
9	Öznitelik çıkarma modülünden gönderilen öznitelikler raporlama sınıfına gönderilir.

CRLF zafiyetinden sonra zafiyet test modülü içerisine eklenen bir sonraki test sınıfı **html enjeksiyonu (HTML injection)** sıfınıdır. Bu sınıfın ismi projede **htmli.py** olarak geçmektedir. Çizelge 4.10' da htmli.py sınıfının HTML enjeksiyonunu test etme aşamaları verilmiştir.

Çizelge 4.10. HTML enjeksiyonu zafiyeti test sınıfı işleyişi

No	**HTML enjeksiyonu zafiyeti test adımları**
1	Parametre olarak gönderilen test edilecek URL, oturum, login sayfa adresi, kullanıcı adı, parola değerleri alınır ve sınıf değişkenlerine atanır.
2	Kullanıcı isteğine göre get ve post metodları çalıştırılır. (Her iki metodun işleyiş mantığı aynı olduğu için tek süreç olarak anlatılacaktır.)
3	Test edilecek sayfaya ulaşılmaya çalışılır. Ulaşılmak istenen sayfa kullanıcı giriş sayfasına yönlendiriyorsa, kullanıcı girişi yapılarak test işlemleri başlatılır.
4	Payloads.py dosyası içerisinden HTML enjeksiyonu zafiyeti payloadları alınır.
5	Alınan payloadlar eğer get metodu çalışıyorsa 'getParser' metodu tarafından hazırlanan URL kalıplarına, eğer post metodu çalışıyorsa 'FormScraper' metodu tarafından hazırlanan POST kalıplarına enjekte edilir.

Çizelge 4.10. HTML enjeksiyonu zafiyeti test sınıfı işleyişi (Devam)

6	Oluşturulan URL ya da POST kalıpları kullanılarak istek gönderilir.
7	Gönderilen isteğe karşılık gelen yanıtın içerisinde enjekte edilen payload aranır.
8	Eğer 8.maddede belirtilen koşul sağlandıysa zafiyet bulundu anlamına gelmektedir. Bu durumda; zafiyet adı, bulunduğu adres, oturum gibi veriler öznitelik çıkarma modülüne gönderilir.
9	Öznitelik çıkarma modülünden gönderilen öznitelikler raporlama sınıfına gönderilir.

Zafiyet test modülüne eklenen bir diğer test sınıfı **ldap.py** sınıfıdır. Bu **sınıf LDAP enjeksiyonu (LDAP Injection)** zafiyetinin test edilmesi için geliştirilmiştir. Çizelge 4.11' de LDAP enjeksiyon test sınıfının çalışma adımlar gösterilmiştir.

Çizelge 4.11. LDAP enjeksiyonu zafiyeti test sınıfı işleyişi

No	**LDAP enjeksiyonu zafiyeti test adımları**
1	Parametre olarak gönderilen test edilecek URL, oturum, login sayfa adresi, kullanıcı adı, parola değerleri alınır ve sınıf değişkenlerine atanır.
2	Kullanıcı isteğine göre get ve post metodları çalıştırılır. (Her iki metodun işleyiş mantığı aynı olduğu için tek süreç olarak anlatılacaktır.)
3	Test edilecek sayfaya ulaşılmaya çalışılır. Ulaşılmak istenen sayfa kullanıcı giriş sayfasına yönlendiriyorsa, kullanıcı girişi yapılarak test işlemleri başlatılır.
4	Payloads.py dosyası içerisinden LDAP enjeksiyonu payloadları alınır.
5	Errors klasörü içerisinden enjeksiyon sonucu görüntülenmesi muhtemel hata listesi alınır.
6	Alınan payloadlar eğer get metodu çalışıyorsa 'getParser' metodu tarafından hazırlanan URL kalıplarına, eğer post metodu çalışıyorsa 'FormScraper' metodu tarafından hazırlanan POST kalıplarına enjekte edilir.

Çizelge 4.11. LDAP enjeksiyonu zafiyeti test sınıfı işleyişi (Devam)

7	Oluşturulan URL ya da POST kalıpları kullanılarak istek gönderilir.
8	Gönderilen isteğe karşılık gelen yanıt içerisinde zafiyet olduğunu ispatlayan hata mesajlarının olup olmadığı kontrol edilir.
9	Eğer dönen yanıt içerisinde hata mesajı ile karşılaşıldıysa; zafiyet adı, bulunduğu adres, oturum gibi veriler öznitelik çıkarma modülüne gönderilir.
10	Öznitelik çıkarma modülünden gönderilen öznitelikler raporlama sınıfına gönderilir.

Geliştirilen bir diğer zafiyet tarama sınıfı **yerel dosya dahil etme (local file inclusion)** zafiyetini test eden **lfi.py** sınıfıdır. LFI zafiyetinin test edilme işleminde temel olarak, girdi alanlarına payloadlar yüklendikten sonra, dönen sayfa yanıtı içerisinde hata mesajı örnekleri aranmaktadır. Çizelge 4.12' de LFI zafiyetinin test aşamaları gösterilmiştir.

Çizelge 4.12. Yerel dosya dahil etme zafiyeti test sınıfı işleyişi

No	**Yerel dosya dahil etme zafiyeti test adımları**
1	Parametre olarak gönderilen test edilecek URL, oturum, login sayfa adresi, kullanıcı adı, parola değerleri alınır ve sınıf değişkenlerine atanır. Varsayılan timeout süresi 3 saniye olarak belirlenir.
2	Kullanıcı isteğine göre get ve post metodları çalıştırılır. (Her iki metodun işleyiş mantığı aynı olduğu için tek süreç olarak anlatılacaktır.)
3	Test edilecek sayfaya ulaşılmaya çalışılır. Ulaşılmak istenen sayfa kullanıcı giriş sayfasına yönlendiriyorsa, kullanıcı girişi yapılarak test işlemleri başlatılır.
4	Payloads klasörü içerisindeki "localfilepayloadShort.txt" dosyası içerisindeki test payloadları alınır.
5	Alınan payloadlar eğer get metodu çalışıyorsa 'getParser' metodu tarafından hazırlanan URL kalıplarına, eğer post metodu çalışıyorsa 'FormScraper' metodu tarafından hazırlanan POST kalıplarına enjekte edilir.

Çizelge 4.12. Yerel dosya dahil etme zafiyeti test sınıfı işleyişi (Devam)

6	Oluşturulan URL ya da POST kalıpları kullanılarak istek gönderilir.
7	Gönderilen isteğe karşılık olarak gönderilen yanıt içerisinde çeşitli hata mesajları aranmaktadır. Eğer ilgili hata mesajları bulunursa zafiyet tespit edildi anlamına gelmektedir.
8	Eğer zafiyet tespit edildiyse; zafiyet adı, bulunduğu adres, oturum gibi veriler öznitelik çıkarma modülüne gönderilir.
9	Öznitelik çıkarma modülünden gönderilen öznitelikler raporlama sınıfına gönderilir.

Yerel dosya dahil etme zafiyeti için test sınıfının yazılmasından sonra, **uzak dosya dahil etme (remote file inclusion)** zafiyeti için test sınıfı yazılmıştır. İlgili sınıfın ismi **rfi.py** olarak tanımlanmıştır. Çizelge 4.13'de RFI zafiyeti için yazılmış test metodunun çalışma adımları yer almaktadır.

Çizelge 4.13. Uzak dosya dahil etme zafiyeti test sınıfı işleyişi

No	**Uzak dosya dahil etme zafiyeti test adımları**
1	Parametre olarak gönderilen test edilecek URL, oturum, login sayfa adresi, kullanıcı adı, parola değerleri alınır ve sınıf değişkenlerine atanır.
2	Test edilecek sayfaya ulaşılmaya çalışılır. Ulaşılmak istenen sayfa kullanıcı giriş sayfasına yönlendiriyorsa, kullanıcı girişi yapılarak test işlemleri başlatılır.
3	getParser metodu tarafından hazırlanan URL kalıbı içerisine yönlendirilecek sayfanın URL adresi eklenir.
4	Oluşturulan test URL' i kullanılarak sayfaya get isteği gönderilir.
5	Gönderilen isteğe karşılık gelen yanıt içerisinde yönlendirme yapılmak istenen sayfanın içeriğinin olup olmadığı kontrol edilir.
6	Eğer zafiyet bulunduysa, zafiyet adı, bulunduğu adres, oturum gibi veriler öznitelik çıkarma modülüne gönderilir.
7	Öznitelik çıkarma modülünden gönderilen öznitelikler raporlama sınıfına gönderilir.

Uzaktan ve yerel dosya dahil etme zafiyetleri için test sınıflarının yazılmasından sonra, **dizin gezinimi zafiyeti (directory traversal**) için test sınıflı yazılmıştır. Windows ve Linux sunucular üzerinde dizin gezinimi zafiyeti için **'windtraversal.py'** ve **'lindtraversal.py'** olmak üzere iki ayrı test sınıfı geliştirilmiştir. Söz konusu sınıfların birbirinden farkı, kullanılan payloadlar ve yapılan istek sonrasında dönen yanıt içerisinde aranan verilerdir. Çizelge 4.14' te dizin gezinimi zafiyeti için yazılmış sınıfların çalışma aşamaları verimiştir.

Çizelge 4.14. Dizin gezinimi zafiyeti test sınıfı işleyişi

No	Dizin gezinimi zafiyeti test adımları
1	Parametre olarak gönderilen test edilecek URL, oturum, login sayfa adresi, kullanıcı adı, parola değerleri alınır ve sınıf değişkenlerine atanır. Varsayılan timeout süresi 3 saniye olarak belirlenir.
2	Kullanıcı isteğine göre get ve post metodları çalıştırılır. (Her iki metodun işleyiş mantığı aynı olduğu için tek süreç olarak anlatılacaktır.)
3	Test edilecek sayfaya ulaşılmaya çalışılır. Ulaşılmak istenen sayfa kullanıcı giriş sayfasına yönlendiriyorsa, kullanıcı girişi yapılarak test işlemleri başlatılır.
4	Kullanılacak payloadlar sınıf türüne göre, 'linuxdtravpayloads.txt' ve 'windtravpayloads.txt' dosyalarından çekilir.
5	Alınan payloadlar eğer get metodu çalışıyorsa 'getParser' metodu tarafından hazırlanan URL kalıplarına, eğer post metodu çalışıyorsa 'FormScraper' metodu tarafından hazırlanan POST kalıplarına enjekte edilir.
6	Oluşturulan URL ya da POST kalıpları kullanılarak istek gönderilir.
7	Gönderilen isteğe karşılık olarak gönderilen yanıt içerisinde çeşitli veri örnekleri aranmaktadır. (Linux sistemlerde 'root' vb, Windows sistemlerde 'boot' vb.)
8	Eğer zafiyet tespit edildiyse; zafiyet adı, bulunduğu adres, oturum gibi veriler öznitelik çıkarma modülüne gönderilir.
9	Öznitelik çıkarma modülünden gönderilen öznitelikler raporlama sınıfına gönderilir.

Dizin gezinimi zafiyeti için test sınıflarının yazılmasından sonra, **URL yönlendirme (Open redirect)** zafiyeti için test sınıfı yazılmıştır. Söz konusu sınıfa ait işlem adımları çizelge 4.15' te gösterilmiştir.

Çizelge 4.15. URL yönlendirme zafiyeti test sınıfı işleyişi

No	URL yönlendirme zafiyeti test adımları
1	Parametre olarak gönderilen test edilecek URL, oturum, login sayfa adresi, kullanıcı adı, parola değerleri alınır ve sınıf değişkenlerine atanır.
2	Test edilecek sayfaya ulaşılmaya çalışılır. Ulaşılmak istenen sayfa kullanıcı giriş sayfasına yönlendiriyorsa, kullanıcı girişi yapılarak test işlemleri başlatılır.
	Payloads klasörü içerisindeki "openredirectpayloads.txt" dosyası içerisindeki test payloadları alınır.
3	getParser metodu tarafından hazırlanan URL kalıbı içerisine sıradaki payload eklenir.
4	Oluşturulan test URL' i kullanılarak sayfaya get isteği gönderilir.
5	Gönderilen isteğe karşılık gelen yanıt içerisinde yönlendirme yapılmak istenen sayfanın içeriğinin olup olmadığı kontrol edilir.
6	Eğer zafiyet bulunduysa, zafiyet adı, bulunduğu adres, oturum gibi veriler öznitelik çıkarma modülüne gönderilir.
7	Öznitelik çıkarma modülünden gönderilen öznitelikler raporlama sınıfına gönderilir.

Zafiyet test modülü içerisine eklenen bir diğer test sınıfı **oscommandexec.py** sınıfıdır. Söz konusu sınıf **işletim sistemi komut enjeksiyonu (command execution)** zafiyetinin test edilmesi için geliştirilmiştir. Çizelge 4.16' da ilgili sınıfın çalışma aşamaları gösterilmiştir.

Çizelge 4.16. İşletim sistemi komut enjeksiyonu zafiyeti test sınıfı işleyişi

No	İşletim sistemi komut enjeksiyonu zafiyeti test adımları
1	Parametre olarak gönderilen test edilecek URL, oturum, login sayfa adresi, kullanıcı adı, parola değerleri alınır ve sınıf değişkenlerine atanır.
2	Kullanıcı isteğine göre get ve post metodları çalıştırılır. (Her iki metodun işleyiş mantığı aynı olduğu için tek süreç olarak anlatılacaktır.)
3	Test edilecek sayfaya ulaşılmaya çalışılır. Ulaşılmak istenen sayfa kullanıcı giriş sayfasına yönlendiriyorsa, kullanıcı girişi yapılarak test işlemleri başlatılır.
4	Payloads.py dosyası içerisinden command execution zafiyeti payloadları alınır.
6	Alınan payloadlar eğer get metodu çalışıyorsa ‘getParser’ metodu tarafından hazırlanan URL kalıplarına, eğer post metodu çalışıyorsa ‘FormScraper’ metodu tarafından hazırlanan POST kalıplarına enjekte edilir.
7	Oluşturulan URL ya da POST kalıpları kullanılarak istek gönderilir.
8	Gönderilen isteğe karşılık gelen yanıtın içerisinde enjekte edilen payload çeşitli koşullar ile birlikte aranır.
9	Eğer 8.maddede belirtilen koşul sağlandıysa zafiyet bulundu anlamına gelmektedir. Bu durumda; zafiyet adı, bulunduğu adres, oturum gibi veriler öznitelik çıkarma modülüne gönderilir.
10	Öznitelik çıkarma modülünden gönderilen öznitelikler raporlama sınıfına gönderilir.

Komut çalıştırma zafiyeti test sınıfının eklenmesinden sonra, aracın geliştirilmesine **PHP kod enjeksiyonu (PHP code injection)** zafiyeti test sınıfının kodlanması ile devam edilmiştir. Söz konusu zafiyete ait **phpi.py** ismindeki test sınıfının çalışma aşamaları Çizelge 4.17’ de gösterilmiştir.

Çizelge 4.17. PHP kod enjeksiyonu test sınıfı işleyişi

No	PHP kod enjeksiyonu zafiyeti test adımları
1	Parametre olarak gönderilen test edilecek URL, oturum, login sayfa adresi, kullanıcı adı, parola değerleri alınır ve sınıf değişkenlerine atanır.
2	Kullanıcı isteğine göre get ve post metodları çalıştırılır. (Her iki metodun işleyiş mantığı aynı olduğu için tek süreç olarak anlatılacaktır.)
3	Test edilecek sayfaya ulaşılmaya çalışılır. Ulaşılmak istenen sayfa kullanıcı giriş sayfasına yönlendiriyorsa, kullanıcı girişi yapılarak test işlemleri başlatılır.
4	Payloads.py dosyası içerisinden code injection zafiyeti payloadları alınır.
6	Alınan payloadlar eğer get metodu çalışıyorsa 'getParser' metodu tarafından hazırlanan URL kalıplarına, eğer post metodu çalışıyorsa 'FormScraper' metodu tarafından hazırlanan POST kalıplarına enjekte edilir.
7	Oluşturulan URL ya da POST kalıpları kullanılarak istek gönderilir.
8	Gönderilen isteğe karşılık gelen yanıtın içerisinde, "root\:\/bin\/bash|" ifadesi, çeşitli karakterlerden ayıklanmış payload ile birlikte aranır.
9	Eğer 8.maddede belirtilen koşul sağlandıysa zafiyet bulundu anlamına gelmektedir. Bu durumda; zafiyet adı, bulunduğu adres, oturum gibi veriler öznitelik çıkarma modülüne gönderilir.
10	Öznitelik çıkarma modülünden gönderilen öznitelikler raporlama sınıfına gönderilir.

Bu tez çalışması kapsamında geliştirilen web zafiyet tarayıcısı içerisine eklenen 15. zafiyet test sınıfı; **server side includes (ssi)** zafiyetinin test edilmesi için kodlanmış **ssi.py** sınıfıdır. Söz konusu sınıfa ait çalışma adımları Çizelge 4.18' de gösterilmiştir.

Çizelge 4.18. SSI zafiyeti test sınıfı işleyişi

No	SSI zafiyeti test adımları
1	Parametre olarak gönderilen test edilecek URL, oturum, login sayfa adresi, kullanıcı adı, parola değerleri alınır ve sınıf değişkenlerine atanır.

Çizelge 4.18. SSI zafiyeti test sınıfı işleyişi (Devam)

2	Kullanıcı isteğine göre get ve post metodları çalıştırılır. (Her iki metodun işleyiş mantığı aynı olduğu için tek süreç olarak anlatılacaktır.)
3	Test edilecek sayfaya ulaşılmaya çalışılır. Ulaşılmak istenen sayfa kullanıcı giriş sayfasına yönlendiriyorsa, kullanıcı girişi yapılarak test işlemleri başlatılır.
4	Payloads.py dosyası içerisinden server side includes zafiyeti payloadları alınır.
6	Alınan payloadlar eğer get metodu çalışıyorsa 'getParser' metodu tarafından hazırlanan URL kalıplarına, eğer post metodu çalışıyorsa 'FormScraper' metodu tarafından hazırlanan POST kalıplarına enjekte edilir.
7	Oluşturulan URL ya da POST kalıpları kullanılarak istek gönderilir.
8	Gönderilen isteğe karşılık gelen yanıtın içerisinde, "root:/bin/[bash|sh]" ifadesi aranır.
9	Eğer 8.maddede belirtilen koşul sağlandıysa zafiyet bulundu anlamına gelmektedir. Bu durumda; zafiyet adı, bulunduğu adres, oturum gibi veriler öznitelik çıkarma modülüne gönderilir.
10	Öznitelik çıkarma modülünden gönderilen öznitelikler raporlama sınıfına gönderilir.

Server side include zafiyeti için test sınıfı yazılmasından sonra, zafiyet test modülünün geliştirilmesine, **sunucu taraflı template enjeksiyonu (server side template injection)** zafiyeti için test sınıfı yazılarak devam edilmiştir. Söz konusu zafiyerin testi için yazılmış sınıfın adı **ssti.py** olarak belirlenmiştir. Çizelge 4.19' da ssti.py sınfınıın çalışma adımları gösterilmiştir.

Çizelge 4.19. Sunucu taraflı template enjeksiyonu zafiyeti test sınıfı işleyişi

No	**Sunucu taraflı template enjeksiyonu zafiyeti test adımları**
1	Parametre olarak gönderilen test edilecek URL, oturum, login sayfa adresi, kullanıcı adı, parola değerleri alınır ve sınıf değişkenlerine atanır.
2	Kullanıcı isteğine göre get ve post metodları çalıştırılır. (Her iki metodun işleyiş mantığı aynı olduğu için tek süreç olarak anlatılacaktır.)

Çizelge 4.19. Sunucu taraflı template enjeksiyonu zafiyeti test sınıfı işleyişi (Devam)

3	Test edilecek sayfaya ulaşılmaya çalışılır. Ulaşılmak istenen sayfa kullanıcı giriş sayfasına yönlendiriyorsa, kullanıcı girişi yapılarak test işlemleri başlatılır.
4	Zafiyet testi için matematiksel işlemler içeren payloadları tanımlanır.
6	Tanımlanan payloadlar eğer get metodu çalışıyorsa 'getParser' metodu tarafından hazırlanan URL kalıplarına, eğer post metodu çalışıyorsa 'FormScraper' metodu tarafından hazırlanan POST kalıplarına enjekte edilir.
7	Oluşturulan URL ya da POST kalıpları kullanılarak istek gönderilir.
8	Gönderilen isteğe karşılık gelen yanıtın içerisinde, payload içindeki matematiksel işlemin sonucu aranır.
9	Eğer 8.maddede belirtilen koşul sağlandıysa zafiyet bulundu anlamına gelmektedir. Bu durumda; zafiyet adı, bulunduğu adres, oturum gibi veriler öznitelik çıkarma modülüne gönderilir.
10	Öznitelik çıkarma modülünden gönderilen öznitelikler raporlama sınıfına gönderilir.

Zafiyet test modülü içerisine eklenen diğer test sınıflarından ilki **XML external entity (XEE)** zafiyeti için kodlanan **xee.py** sınıfıdır. Zafiyet testi için geliştirilen bir diğer sınıf ise **siteler arası betik çalıştırma (cross site scripting)** zafiyeti için geliştirilen **xss.py** sınıfıdır. Söz konusu zafiyetler için geliştirilmiş test sınıflarının çalışma adımları HTML enjeksiyonu zafiyeti için geliştirilen test metodu ile aynı adımları içermektedir. Sınıflar arasındaki tek fark zafiyeti test etmek için kullanılan payloadların birbirinden farklı olmasıdır. Zafiyet test modülüne eklenen diğer sınıflardan **headerxss.py**, **HTTP header tabanlı XSS** zafiyetinin testi için geliştirilmiştir. Söz konusu test sınıfı header tabanlı SQL enjeksiyonu zafiyetinin testi için geliştirilmiş sınıf ile aynı aşamaları içermektedir. İki sınıf arasındaki fark, diğer sınıflarda olduğu gibi farklı payloadların kullanılmasıdır. Zafiyet test modülü için geliştirilen son sınıf **XPATH enjeksiyonu** zafiyetinin testi için geliştirilen **xpathi.py** sınıfıdır. Söz konusu sınıfın çalışma prensibi, SQL enjeksiyonu ve LDAP enjeksiyonunda olduğu gibi kullanılan payloadlar sonrasında dönen yanıt içinde çeşitli

hata mesajları aranması prensibine dayanmaktadır. Söz konusu sınıfların işleyişi arasındaki fark kullanılan payloadlar ve kontrol edilen hata mesajlarıdır.

4.1.4. Web Zafiyet Tarayıcısı Öznitelik Çıkarma Modülü

Bu tez çalışmasında geliştiren web zafiyet tarayıcısı bir web sayfasını tararken, sayfa üzerinde bir zafiyet tespit ettiğinde, söz konusu zafiyetin tespit edildiği web sayfasına, forma ve inputa ait çeşitli özniteliklerin tespit edilmesi gerekmektedir. Bunu gerçekleştirmekteki amaç; elde edilen verileri kullanarak zafiyetlerin ne tür sayfalarda, formlarda ya da inputlarda ortaya çıktığının tespit edilmesidir. Tespit edilen her zafiyet için söz konusu veriler toplandığında ortaya özgün bir veri seti çıkmıştır. Söz konusu modül kullanılarak toplanan verilerden oluşan bu veri seti, çalışmanın ilerleyen kısımlarında sayfa sınıflandırma ve zafiyetler arasında birliktelik analizi gerçekleştirmek için kullanılmıştır.

Zafiyet tespit edilen sayfaya ait özniteliklerin çıkarılmasında ilk olarak wappalyzer kütüphanesi kullanılmıştır. Söz konusu kütüphane sayesinde zafiyet tespit edilen sayfaya ait; CMS, database managers, javascript frameworks, search engines, programming languages, web server extensions gibi birçok bilgi elde edilebilmektedir. Şekil 4.18' de wappalyzer kütüphanesi aracılığı ile öznitelik çıkarımına olanak sağlayan wapplyzerFeatures metoduna ait kodlar verilmiştir.

```
41      def wappalyzerFeatures(self,url):
42          wappalyzer = Wappalyzer.latest()
43          webpage = WebPage.new_from_url(url)
44          categories = wappalyzer.analyze_with_categories(webpage)
45          keyWillBeAdded=''
46          valueWillBeAdded=''
47          for key,value in categories.items():
48              keyWillBeAdded=key
49              for key, value in value.items():
50                  valueWillBeAdded=value[0]
51              self.features[valueWillBeAdded]=keyWillBeAdded
```

Şekil 4.18. wappalyzerFeatures metodu

Şekil 4.18 de gösterilmekte olan metod incelendiğinde; öznitelik çıkartılacak olan URL adresi wappalyzer sınıfına parametre olarak gönderilmekte, geriye içerisinde

öznitelikleri barındıran bir sözlük döndürülmektedir. Geriye döndürülen değerler, features ismindeki sözlüğe eklenmektedir.

Zafiyetin tespit edildiği sayfaya ait yapısal özniteliklerin wappalyzer kütüphanesi ile elde edilmesinden sonra, sayfaya, forma ve inputa ait öznitelik değerlerinin elde edilmesini sağlayan metodlar geliştirilmiştir. Söz konusu metodların çalışma mantığı, zafiyetin tespit edildiği sayfanın kaynak kodlarının parçalanması ile gerçekleştirilmektedir. Bu işlem için Python bünyesindeki Beautifulsoup Kütüphanesi kullanılmaktadır.

Öznitelik çıkarma modülünün geliştirilmesi işlemine, sayfaya ait diğer özniteliklerin elde edilmesine olanak sağlayan page features metodunun geliştirilmesi ile devam edilmiştir. Geliştirilen metod sayesinde sayfaya ait; title, decription, keywords, generator, content-type, form sayısı, input sayısı, parola giriş alanı sayısı, arama alanı sayısı, select ve radio buton sayısı, email alanı sayısı, text area sayısı, text sayısı, file input sayısı ve buton sayısı gibi veriler elde edilmektedir.

Sayfa içerisinden öznitelik çıkarımı için geliştirilen bir diğer metod ise; zafiyetin tespit edildiği forma ait verilerin çıkarılmasını sağlayan formFeatures metodudur. Söz konusu metod sayesinde, form içerisindeki input sayısı, form içindeki parola alanı sayısı, form içindeki arama alanı (search area) sayısı, form içindeki select, radio button ve option sayısı, form içindeki buton sayısı, forma ait action bilgisi ve form içerisindeki text verileri elde edilmektedir.

Sayfa içerisinden öznitelik çıkarımı için geliştiren son metod ise, zafiyetin tespit edildiği inputa ait çeşitli verilerin çıkartılmasını sağlayan input features metodudur. Geliştirilen metod sayesinde zafiyetin tespit edildiği inputa ait; name, size, type, src, required, value, max, min, maxlength, minlenght ve accept gibi özellikler elde edilmektedir.

Tüm metodların çalışması sonrasında 70 farklı öznitelik elde edilmektedir. FeatureExtractor sınıfının çalışması sonrasında geriye sözlük tipinde bir değişken döndürülmektedir. Döndürülen sözlükte anahtarlar öznitelik isimleri, anahtarlara

karşılık atanan değerler ise, methodlar tarafından elde edilen verilerdir. Çizelge 4.20' de öznitelik çıkarma modülü tarafından üretilen öznitelikler gösterilmiştir.

Çizelge 4.20. Öznitelik çıkarma modülü tarafından üretilen öznitelikler

Öznitelik Türü	**Öznitelikler**
Wappalyzer kütüphanesi öznitelikleri	'Payload', 'CMS', 'Message Boards', 'Database Managers', 'Documentation Tools','Widgets', 'Ecommerce','Hosting Panels', 'Javascript Frameworks', 'Comment Systems', 'Font Scripts', 'Web Frameworks', 'Editors', 'LMS', 'Web Servers', 'Rich Text Editors', 'Javascript Graphics', 'Mobile Frameworks', 'Programming Languages', 'Operating Systems', 'Search Engines', 'Web Mail', 'Marketing Automation', 'Web Server Extensions', 'Databases', 'Payment Processors', 'Dev Tools', 'Live Chat'
Sayfa öznitelikleri	'description', 'keywords', 'generator', 'content-type', 'title', 'Num Of Forms', 'Num Of Inputs', 'Num Of Password Areas', 'Num Of Search Areas', 'Num Of Select', 'Num Of Radio', 'Num Of Option', 'Num Of Checkbox', 'Num Of Text', 'Num Of Email', 'Num Of Text Areas','Num Of File', 'Num Of Buttons'
Form öznitelikleri	Num Of Inputs In Vulnerable Form',' Types of Inputs In Vulnerable Form', 'Num Of Password Areas In Vulnerable Form', 'Num Of Search Areas In Vulnerable Form', 'Num Of Text Areas In Vulnerable Form', 'Num Of Select In Vulnerable Form', 'Num Of Radio In Vulnerable Form', 'Num Of Option In Vulnerable Form','Num Of Checkbox In Vulnerable Form', 'Num Of Buttons In Vulnerable Form', 'Action Of Form', 'Text of Form'
Input öznitelikleri	'name', 'type', 'src', 'size', 'required', 'value', 'max', 'min', 'maxlength', 'accept'

4.1.5. Web Zafiyet Tarayıcısı Raporlama Modülü

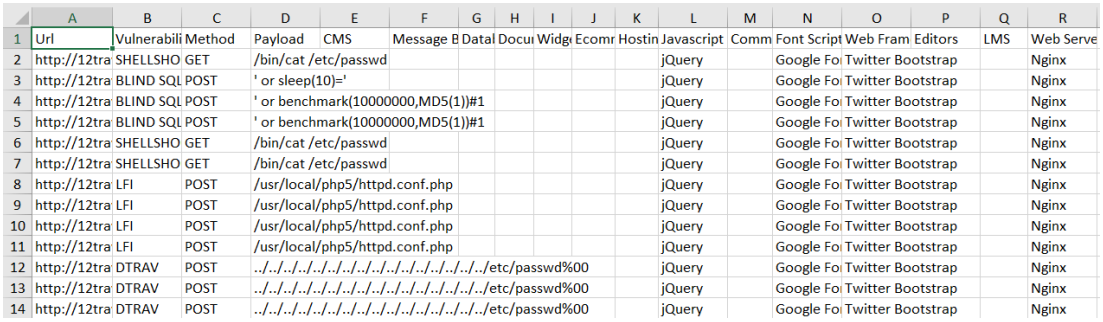
Geliştirilen zafiyet tarayıcısı tarafından bir web sayfasında bir güvenlik açığı tespit edildiğinde, öznitelik çıkarma modülü çalıştırılarak zafiyetin tespit edildiği web sayfasına ait öznitelikler çıkarılmaktadır. Daha sonra zafiyet test sınıfı tarafından bu öznitelikler raporlama modülüne gönderilmektedir.

Raporlama modülü sayesine bir web sayfası test edildikten sonra; tespit edilen tüm zafiyetler, hangi sayfalarda tespit edildiği, hangi payload kullanıldığı, hangi HTML metodu üzerinden testin gerçekleştiği gibi bilgiler ve birçok farklı öznitelik kaydedilmektedir.

```
65  def writeDatatoXls(gelenVeriler,today):
66      filename = realpath + str(today) + ".xlsx"
67      for c in columnNames:
68          dataDictionary[c]=None
69      for d in dataDictionary:
70          if d in gelenVeriler:
71              dataDictionary[d]=gelenVeriler[d]
72      datas=[]
73      for d in dataDictionary:
74          datas.append(dataDictionary[d])
75      kitap=load_workbook(filename)
76      sayfa = kitap.active
77      try:
78          sayfa.append(datas)
79      except:
80          for i in range(0,len(datas)):
81              datas[i]=str(datas[i]).encode('unicode_escape').decode('utf-8')
82          sayfa.append(datas)
83      kitap.save(filename)
84      kitap.close()
```

Şekil 4.19. generateReport.py sınıfı writeDatatoXls metodu

Raporlama modülü web sayfasına ait tarama işlemi tamamlandıktan sonra, içerisinde öznitelik sözlükleri barındıran bir listeyi parametre olarak almaktadır. Daha sonra liste içerisindeki her bir elemanı excel dosyasına yazarak raporlama işlemini tamamlamaktadır. Şekil 4.19' da raporlama modülü içerisinde yer alan writeDatatoXls metodunun kodları verilmiştir. Şekil 4.19'da gösterilen kodlar incelendiğinde, metoda parametre olarak gelen veriler öncelikle datas isimli listeye eklenmekte daha sonra da excel dosyasına yazılmaktadır. Parametre olarak gelen öznitelikler içerisinde tespit edilmemiş öznitelik varsa, ona karşılık değer olarak None ifadesi atanmaktadır. Şekil 4.20' de bir tarama sonrasında raporlama modülü tarafından üretilmiş bir test raporu gösterilmiştir.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	Url	Vulnerabili	Method	Payload	CMS	Message B	Datal	Docu	Widg	Ecomr	Hostin	Javascript	Comm	Font Script	Web Fram	Editors	LMS	Web Serve
2	http://12tra	SHELLSHO	GET	/bin/cat /etc/passwd								jQuery		Google Fo	Twitter Bootstrap			Nginx
3	http://12tra	BLIND SQL	POST	' or sleep(10)='								jQuery		Google Fo	Twitter Bootstrap			Nginx
4	http://12tra	BLIND SQL	POST	' or benchmark(10000000,MD5(1))#1								jQuery		Google Fo	Twitter Bootstrap			Nginx
5	http://12tra	BLIND SQL	POST	' or benchmark(10000000,MD5(1))#1								jQuery		Google Fo	Twitter Bootstrap			Nginx
6	http://12tra	SHELLSHO	GET	/bin/cat /etc/passwd								jQuery		Google Fo	Twitter Bootstrap			Nginx
7	http://12tra	SHELLSHO	GET	/bin/cat /etc/passwd								jQuery		Google Fo	Twitter Bootstrap			Nginx
8	http://12tra	LFI	POST	/usr/local/php5/httpd.conf.php								jQuery		Google Fo	Twitter Bootstrap			Nginx
9	http://12tra	LFI	POST	/usr/local/php5/httpd.conf.php								jQuery		Google Fo	Twitter Bootstrap			Nginx
10	http://12tra	LFI	POST	/usr/local/php5/httpd.conf.php								jQuery		Google Fo	Twitter Bootstrap			Nginx
11	http://12tra	LFI	POST	/usr/local/php5/httpd.conf.php								jQuery		Google Fo	Twitter Bootstrap			Nginx
12	http://12tra	DTRAV	POST	../../../../../../../../../../../../etc/passwd%00								jQuery		Google Fo	Twitter Bootstrap			Nginx
13	http://12tra	DTRAV	POST	../../../../../../../../../../../../etc/passwd%00								jQuery		Google Fo	Twitter Bootstrap			Nginx
14	http://12tra	DTRAV	POST	../../../../../../../../../../../../etc/passwd%00								jQuery		Google Fo	Twitter Bootstrap			Nginx

Şekil 4.20. Raporlama modülü tarafından üretilen bir tarama raporu

4.1.6. Web Test Laboratuvarının Oluşturulması ve Test İşlemlerinin Gerçekleştirilmesi

Bu tez çalışması kapsamında, hem geliştirilen zafiyet tarayıcısının geliştirme esnasında zafiyet testlerindeki yeterliliğini ölçmek, hem de geliştirme sonrasında yapılacak testler ile veri toplayarak veri seti oluşturmak için bir web test laboratuvarı oluşturulmuştur. Web test laboratuvarı geliştirilmesinin bir diğer zorunluluğu ise; bir web sayfası üzerinde zafiyet tarama işleminin yapılabilmesi için söz konusu sayfaya sahip kişi ya da kurum ile, zafiyet testinin ne zaman yapılacağı, zafiyet testinin kapsamının ne olacağı, zafiyet testi esnasında hangi test sınıflarının çalıştırılacağı gibi bilgileri içeren bir sözleşme yapılması gerekliliğidir. Bu kriterleri kurum ile beraber belirlemenin çok fazla zaman alması, sahip olduğu web sayfasının test edilmesini isteyen kişi/kurum sayısının azlığı gibi sebeplerden dolayı, bir web test laboratuvarının oluşturulma zorunluluğu ortaya çıkmıştır.

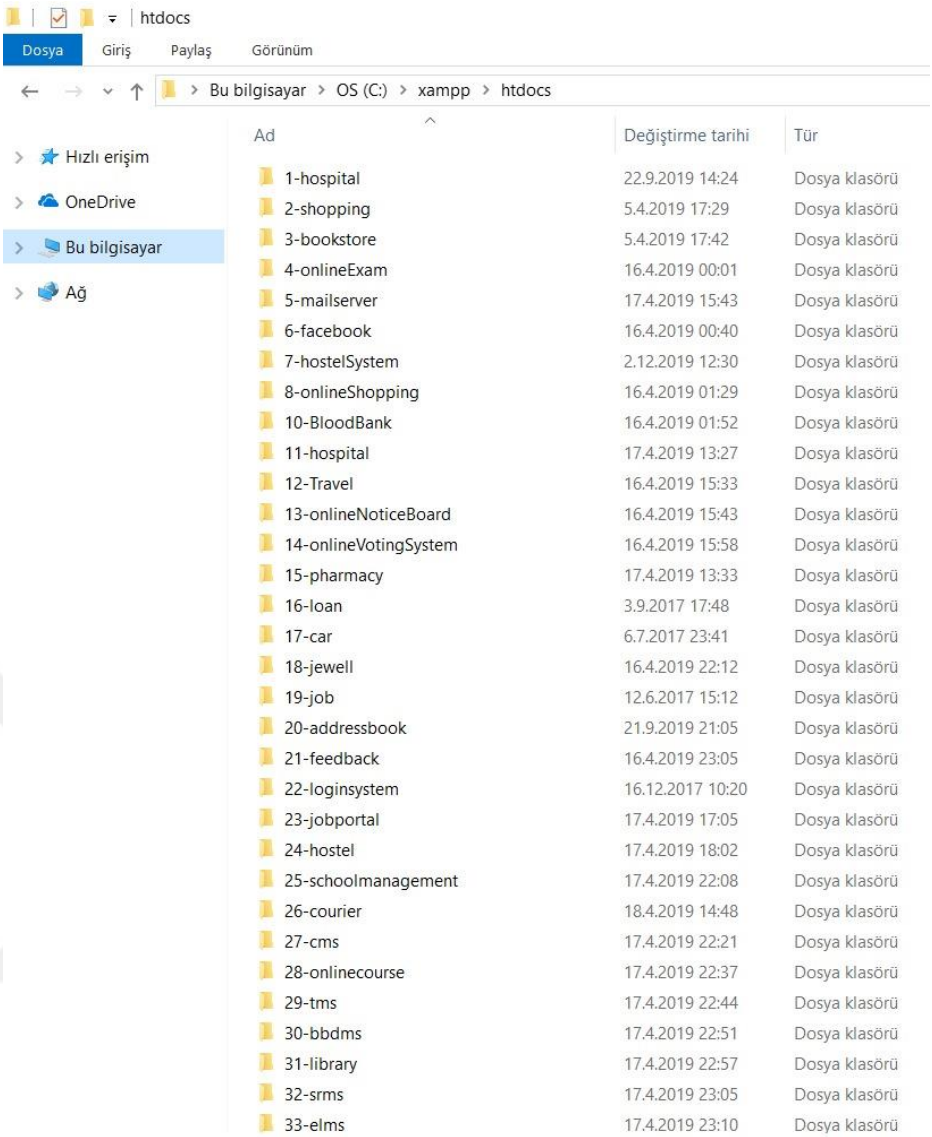
Web test laboratuvarı oluşturma işleminin bir diğer avantajı ise farklı türde yapılar içeren, farklı amaçlar için hazırlanmış web uygulamalarının dâhil edilebilmesidir. Bu tez çalışmasında oluşturulan web test laboratuvarında PHP ve ASP tabanlı web uygulamaları kullanılmıştır. Web test laboratuvarı oluşturulmasında kullanılan web uygulamaları; Github, Gitlab ve Sourceforge platformlarında geliştiriciler tarafından paylaşılmış açık kaynak kodlu projelerdir. Web laboratuvarı bünyesinde toplam 262 farklı web uygulaması yer almaktadır. Laboratuvar içerisinde yer alan web uygulamalarından birkaçı aşağıda listelenmiştir;

- Hastane yönetim sistemi,
- Online alışveriş uygulamaları,
- Online sınav uygulaması,

- Çeşitli sosyal medya uygulamaları,
- Otel ve hostel yönetim sistemleri,
- Online kan bağış uygulaması,
- Araç, ev kiralama uygulamaları,
- Kargo takip uygulaması,
- İçerik yönetim sistemleri,
- Haber uygulamaları,
- Online iş başvuru uygulaması,
- Konferans kayıt ve takip uygulaması,
- Online oy kullanma uygulaması,
- Stok takip uygulamaları,
- Sinema, otobüs, uçak ve tren bilet alım uygulamaları,
- Online bankacılık uygulamaları.

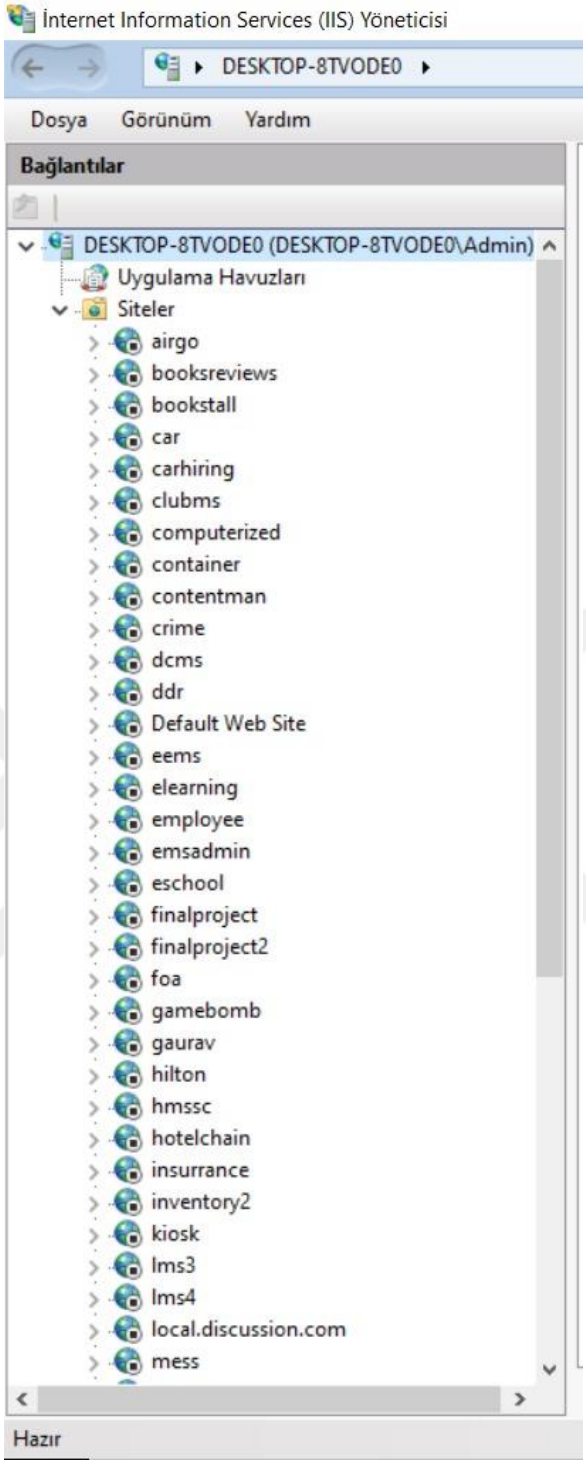
Web test laboratuvarı oluşturulurken, çeşitli amaçlar için geliştirilmiş sayfaların seçilmesine dikkat edilmiştir. Böylelikle geliştirilen web zafiyet tarayıcısının gerçek hayatta karşılaşması muhtemel uygulamalar üzerinde test edilmesi imkânı sağlanmıştır. Oluşturulan test laboratuvarı, sahip olduğu uygulama sayısı bakımından literatürdeki en büyük test laboratuvarı olma özelliği taşımaktadır.

Oluşturulan web test laboratuvarı bünyesinde 207 tane birbirinden farklı PHP tabanlı web uygulaması bulunmaktadır. PHP tabanlı web uygulamalarının yerel sunucuya kurulup yapılandırılması işleminde XAMPP ve WAMP sunucuları kullanılmıştır. XAMPP sunucunun isminde yer alan harflerden X; platform bağımsız bir uygulama olduğunu, A; apache sunucusu içerdiğini, M; MYSQL veri tabanı sunucusu içerdiğini, P; PHP yorumlayıcısı (interpreter) içerdiğini, P; perl yorumlayıcısı içerdiğini ifade etmektedir. WAMP ise; Windows, Apache, MySQL ve PHP ifadelerinin kısaltmasından oluşmaktadır. Her iki yazılım, kullanıcılarına bir web sunucu, bir veri tabanı sunucu ve bir web dili yorumlayıcı sunarak, web uygulamalarını yerel sunucuda çalıştırma imkânı vermektedir (Agrawal ve Gupta, 2014). Şekil 4.21’ de Xammp uygulamasının htdocs klasörü altında yer alan PHP tabanlı web uygulamalarının bir bölümü gösterilmiştir.



Şekil 4.21. XAMMP üzerinde yapılandırılmış web uygulamaları

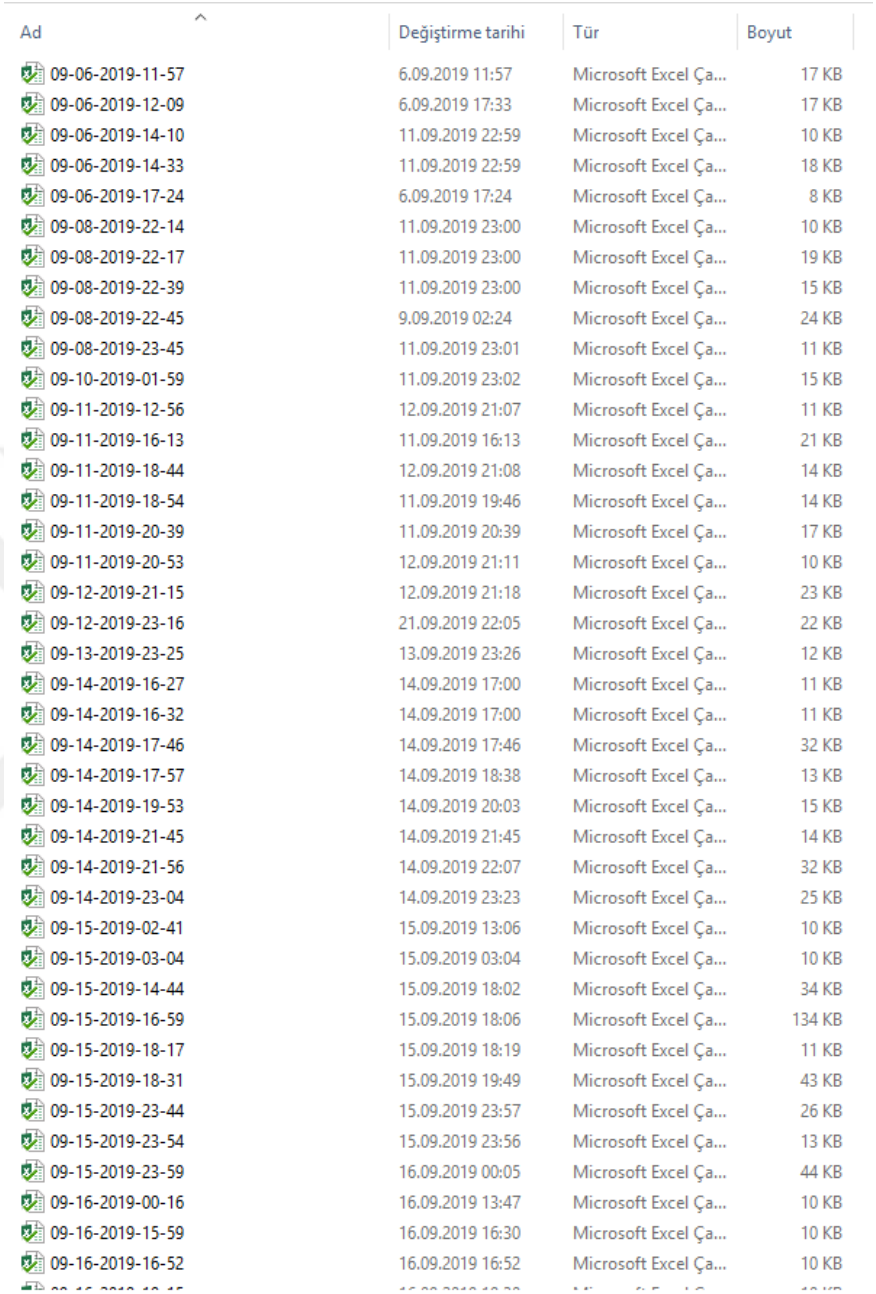
Oluşturulan web laboratuvarı bünyesinde ASP tabanlı 55 farklı uygulama yer almaktadır. ASP tabanlı uygulamaların yerel sunucuda çalıştırılarak teste hazır hale getirilmesinde Microsoft IIS ve Microsoft SQL Server kullanılmıştır. IIS (Internet Information Services); bir web sunucu olarak kullanıldığında, kullanıcılarına web uygulamalarını barındırma ve yönetmek için bir platform sağlamaktadır (Khalid, 2012). Şekil 4.22' de IIS üzerinde yapılandırılmış ASP tabanlı web uygulamalarının bir bölümü gösterilmiştir.



Şekil 4.22. IIS üzerinde yapılandırılmış web uygulamaları

Web zafiyet test aracının geliştirilmesi ve web test laboratuvarının tamamlanmasından sonra, geliştirilen araç kullanılarak zafiyet test işlemlerine başlanmıştır. Zafiyet testleri laboratuvar içerisinde kurulumu gerçekleştirilen tüm web uygulamalarının test edilmesi ile tamamlanmıştır. Gerçekleştirilen her teste ait sonuçlar, testin tamamlandığı gün, ay, yıl, saat ve dakika bilgileri ile bir excel dosyasına

kaydedilmiştir. Gerçekleştirilen test işlemleri sonrasında elde edilen sonuçların bir kısmının gösterildiği ekran görüntüsü Şekil 4.23' de verilmiştir.



Ad	Değiştirme tarihi	Tür	Boyut
09-06-2019-11-57	6.09.2019 11:57	Microsoft Excel Ça...	17 KB
09-06-2019-12-09	6.09.2019 17:33	Microsoft Excel Ça...	17 KB
09-06-2019-14-10	11.09.2019 22:59	Microsoft Excel Ça...	10 KB
09-06-2019-14-33	11.09.2019 22:59	Microsoft Excel Ça...	18 KB
09-06-2019-17-24	6.09.2019 17:24	Microsoft Excel Ça...	8 KB
09-08-2019-22-14	11.09.2019 23:00	Microsoft Excel Ça...	10 KB
09-08-2019-22-17	11.09.2019 23:00	Microsoft Excel Ça...	19 KB
09-08-2019-22-39	11.09.2019 23:00	Microsoft Excel Ça...	15 KB
09-08-2019-22-45	9.09.2019 02:24	Microsoft Excel Ça...	24 KB
09-08-2019-23-45	11.09.2019 23:01	Microsoft Excel Ça...	11 KB
09-10-2019-01-59	11.09.2019 23:02	Microsoft Excel Ça...	15 KB
09-11-2019-12-56	12.09.2019 21:07	Microsoft Excel Ça...	11 KB
09-11-2019-16-13	11.09.2019 16:13	Microsoft Excel Ça...	21 KB
09-11-2019-18-44	12.09.2019 21:08	Microsoft Excel Ça...	14 KB
09-11-2019-18-54	11.09.2019 19:46	Microsoft Excel Ça...	14 KB
09-11-2019-20-39	11.09.2019 20:39	Microsoft Excel Ça...	17 KB
09-11-2019-20-53	12.09.2019 21:11	Microsoft Excel Ça...	10 KB
09-12-2019-21-15	12.09.2019 21:18	Microsoft Excel Ça...	23 KB
09-12-2019-23-16	21.09.2019 22:05	Microsoft Excel Ça...	22 KB
09-13-2019-23-25	13.09.2019 23:26	Microsoft Excel Ça...	12 KB
09-14-2019-16-27	14.09.2019 17:00	Microsoft Excel Ça...	11 KB
09-14-2019-16-32	14.09.2019 17:00	Microsoft Excel Ça...	11 KB
09-14-2019-17-46	14.09.2019 17:46	Microsoft Excel Ça...	32 KB
09-14-2019-17-57	14.09.2019 18:38	Microsoft Excel Ça...	13 KB
09-14-2019-19-53	14.09.2019 20:03	Microsoft Excel Ça...	15 KB
09-14-2019-21-45	14.09.2019 21:45	Microsoft Excel Ça...	14 KB
09-14-2019-21-56	14.09.2019 22:07	Microsoft Excel Ça...	32 KB
09-14-2019-23-04	14.09.2019 23:23	Microsoft Excel Ça...	25 KB
09-15-2019-02-41	15.09.2019 13:06	Microsoft Excel Ça...	10 KB
09-15-2019-03-04	15.09.2019 03:04	Microsoft Excel Ça...	10 KB
09-15-2019-14-44	15.09.2019 18:02	Microsoft Excel Ça...	34 KB
09-15-2019-16-59	15.09.2019 18:06	Microsoft Excel Ça...	134 KB
09-15-2019-18-17	15.09.2019 18:19	Microsoft Excel Ça...	11 KB
09-15-2019-18-31	15.09.2019 19:49	Microsoft Excel Ça...	43 KB
09-15-2019-23-44	15.09.2019 23:57	Microsoft Excel Ça...	26 KB
09-15-2019-23-54	15.09.2019 23:56	Microsoft Excel Ça...	13 KB
09-15-2019-23-59	16.09.2019 00:05	Microsoft Excel Ça...	44 KB
09-16-2019-00-16	16.09.2019 13:47	Microsoft Excel Ça...	10 KB
09-16-2019-15-59	16.09.2019 16:30	Microsoft Excel Ça...	10 KB
09-16-2019-16-52	16.09.2019 16:52	Microsoft Excel Ça...	10 KB

Şekil 4.23. Test işlemlerinden elde edilen sonuç raporları

Tarama işlemlerinin tamamlanması sonrasında elde edilen test sonuçları birleştirilerek üzerinde çalışılacak veri seti elde edilmiştir. Veri setinin oluşturulmasından sonra, yapay zekâ ve makine öğrenmesi işlemlerine geçilmiştir.

4.1.7. Oluşturulan Veri Seti Kullanılarak Sınıflandırma İşlemlerinin Gerçekleştirilmesi

Geliştirilen web uygulama zafiyet tarayıcısı kullanılarak, yine bu tez kapsamında oluşturulmuş web test laboratuvarı üzerinde gerçekleştirilen testler sonrasında, 7781 örnek içeren bir veri seti oluşturulmuştur. Oluşturulan veri seti içerisinde, öznitelik çıkarma modülü tarafından çıkartılan, her örneğe ait 70 adet öznitelik bulunmaktadır. Söz konusu özniteliklerin listesi, "4.1.4 Web Zafiyet Tarayıcısı Öznitelik Çıkarma Modülü" başlığı altında verilmiştir. Sınıflandırma çalışmalarında kullanılmak üzere oluşturulan veri setine ait diğer bilgiler Çizelge 4.21' de gösterilmiştir. İlgili çizelgede zafiyetlerin gösterilmesinde, daha yaygın kullanımı nedeni ile İngilizce isimlerine yer verilmiştir.

Çizelge 4.21. Oluşturulan veri setinde yer alan zafiyet sayıları

Zafiyet Türü	Adedi
Blind SQL Injection	1747
Buffer Overflow	46
Command Execution	1119
Header SQL Injection	58
Carriage Line Return Feed	3
Directory Traversal	850
HTML Injection	352
Local File Injection	750
Open Redirect	360
PHP-Code Injection	952
Header Based Cross Site Scripting	2
Remote File Inclusion	9
Shellshock	83
Error Based SQL Injection	596
Server side Includes	166
Server Side Template Injection	224
XML External Entity	2
Cross Site Scripting (XSS)	462

Veri setinin oluşturulmasından sonra, söz konusu veriler kullanılarak sayfa sınıflandırma işlemine geçilmiştir. Geliştirilen modelde, ilk olarak sayfa türü belirlenecek, daha sonra da belirlenen sayfa türüne göre karşılaşılması muhtemel zafiyet türleri listelenecektir. Böyle bir yaklaşım kullanılarak gerçekleştirilen taramalar, test edilen sayfada en yüksek görülme olasılığına sahip web zafiyetlerinin test edilmesini sağlayacaktır. Görülme olasılığı az olan zafiyetler ise kullanıcı isteği doğrultusunda test edilmeyecektir.

4.1.7.1. Sayfa Sınıflandırma İşlemleri

Sayfa türü sınıflandırma işleminde, veri setinde bulunan her örneğe ait URL adresleri tek tek ziyaret edilerek, hangi türden sayfa olduğu etiketlenmiştir. Sayfa etiket türlerine karar verirken, web sayfalarında yaygın olarak karşılaşılan ve diğer sayfa türlerinden farklı karakteristik özelliklere sahip olan türlerin belirlenmesine dikkat edilmiştir. Bu tez çalışmasında, bahsedilen kriterler doğrultusunda sekiz sayfa türü belirlenmiştir. Bunlar;

- Login Page,
- Register Page,
- Search Page,
- List Page,
- Info Page,
- Forgot Password Page
- Contact Us Page
- Submit Page' dir.

Belirlenen sayfa türlerinden ilki; Login Page' dir. Günümüzde çoğu web uygulamasında kullanıcı giriş sayfaları yer almaktadır. Karakteristik bir kullanıcı giriş sayfasında bir kullanıcı adı ya da email girme alanı, bir parola girme alanı, bir de buton yer almaktadır. Ayrıca bazı durumlarda kullanıcıların giriş bilgilerinin kaydedilmesini sağlayan "beni hatırla" checkbox alanı da bulunmaktadır. Diğer sayfa türü Register Page' de Login Page gibi kullanıcılara özel işlemler içeren web uygulamalarında, kullanıcıların kaydolması için geliştirilmiş sayfalardır. Search Page türü, birçok web sayfasında karşılaşılan, belirli kriterlere göre web uygulaması içinde arama

yapılmasını sağlayan türde sayfaları kapsamaktadır. List Page türü, alışveriş sayfaları, öğrenci otomasyonları gibi belirli kategori ya da sınıf altındaki kayıtların listelendiği sayfalar için kullanılmaktadır. Info Page türü, sayfa üzerinde herhangi bir input bileşeni içermeyen, ya da çok az içeren, genellikle bir konu hakkında salt metin üzerinden bilgi vermek için kullanılan sayfaları kapsamaktadır. Bu sayfalara örnek olarak, bir blog sayfasında, blog yazarı kişinin kendine ait özgeçmiş bilgilerini paylaştığı, "hakkımda" temalı sayfalar gösterilebilmektedir. Forgot Password ve Contact Us Page türleri, isimlerindende anlaşılacağı gibi web sayfalarında yaygın olarak bulunan, unutulan parolanın yenilenmesini sağlayan sayfalar ile sayfa sahibi kişi ya da kurum ile iletişime geçilmesini sağlayan sayfaları kapsamaktadır. Son olarak Submit Page türü, daha önce bahsedilen herhangi bir tür içerisine dahil edilemeyen, kullanıcı tarafından girilen verileri submit etmeye yarayan sayfaları kapsamaktadır. Örneğin bir stok takip uygulamasında yer alan, yeni ürünlerin girilmesi için kullanılan sayfalar buna örnek olarak gösterilebilmektedir.

Oluşturulan veri seti üzerinde gerçekleştirilen etiketleme işleminden sonra, elde edilen sayfa türlerine ait örnek sayıları Çizelge 4.22' de gösterilmektedir.

Çizelge 4.22. Etiketleme sonrasında veri setinde yer alan sayfa türü sayıları

Sayfa Türü	**Adedi**
Contact Us Page	249
Forgot Password Page	102
Info Page	799
List Page	353
Login Page	932
Register Page	1209
Search Page	440
Submit Page	3707

Etiketlenen örnekler içerisinde Submit page türünde sayfaların çok çıkmasının nedeni, web uygulamalarında diğer kategorilere girmeyen ama kullanıcı verilerinin alınarak işlendiği tüm sayfaların bu gruba dâhil edilmesidir. Farklı uygulamalarda farklı amaçlarla oluşturululmuş birçok form sayfaları bulunmaktadır. Söz konusu her sayfa

için farklı bir tür oluşturulamayacağından dolayı, söz konusu sayfalar submit page içerisine dâhil edilmiştir.

Etiketleme sonrası elde edilen veri seti üzerinde ilk olarak tekrarlı örnekler çıkartılmıştır. Daha sonra sayfa sınıflandırması için gerekli öznitelikler belirlenmiştir. Bu çalışmada sayfa türünün belirlenmesinde kullanılan öznitelikler aşağıda listelenmiştir. Listelenen her bir öznitelik, sınıflandırılacak web sayfası üzerinde, ismi ile belirtilen input bileşenlerinin sayısını göstermektedir.

- Number of Forms,
- Number of Inputs,
- Number of Password Areas,
- Number Of Search Areas,
- Number of Select,
- Number of Radio Buttons,
- Number of Option,
- Number of Checkbox,
- Number of Email Area,
- Number of Textarea,
- Number of Text,
- Number of File,
- Number of Buttons.

Veri seti kullanarak sayfa sınıflandırma işlemleri Spyder ortamında Python programlama dili kullanılarak gerçekleştirilmiştir. Veri seti üzerinde her bir sınıflandırma algoritmasının uygulanmasında; python' ın pandas, numpy, sklearn ve pickle kütüphaneleri kullanılmıştır. Gerekli kütüphanelerin tanımlanmasından sonra veri seti sınıflandırma işlemleri için hazır hale getirilerek sınıflandırma işlemleri başlatılmıştır. Veri setinin işlenmesi, eğitim ve test setlerinin oluşturulması, sınıflandırma ve tahmin adımlarının gerçekleştirilmesi adımları, sınıflandırma algoritmalarının kullanımı için yazılmış python sınıflarından bir tanesi üzerinde anlatılacaktır. Şekil 4.24' de sayfa sınıflandırma işleminde Random Forest algoritmasının kullanımı için yazılmış "predictrf.py" sınıfına ait kodlar verilmiştir.

predictrf.py | predictdectree.py | neuralnetworks.py | predictsvm.py | predictlogistic.py

```
1 import pandas as pd
2 import numpy as np
3 from sklearn.model_selection import train_test_split
4 from sklearn.ensemble import RandomForestClassifier
5 from sklearn import metrics
6 from sklearn.metrics import confusion_matrix
7 import pickle
8
9 df = pd.read_csv('predict.csv')
10 df.dropna()
11
12 y = df.iloc[:, 0].values
13 X = df.iloc[:,1:14].values
14
15 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,random_state = 0)
16 model = RandomForestClassifier(n_estimators=100)
17 model.fit(X_train,y_train)
18
19 y_trainpred = model.predict(X_train)
20 print("Train Accuracy:",metrics.accuracy_score(y_train, y_trainpred))
21
22 #test Accuracy
23 y_pred=model.predict(X_test)
24
25 print("Test Accuracy:",metrics.accuracy_score(y_test, y_pred))
26 result = confusion_matrix(y_test, y_pred, labels=["Submit Page", "Login Page",
27                                                   "Search Page", "Register Page",
28                                                   "List Page", "Info Page",
29                                                   "Forgot Password Page",
30                                                   "Contact Us Page"])
31
32 pickle.dump(model, open('rfmodel.sav', 'wb'))
33
```

Şekil 4.24. Random Forest algoritması kullanılarak sınıflandırma modeli oluşturulması

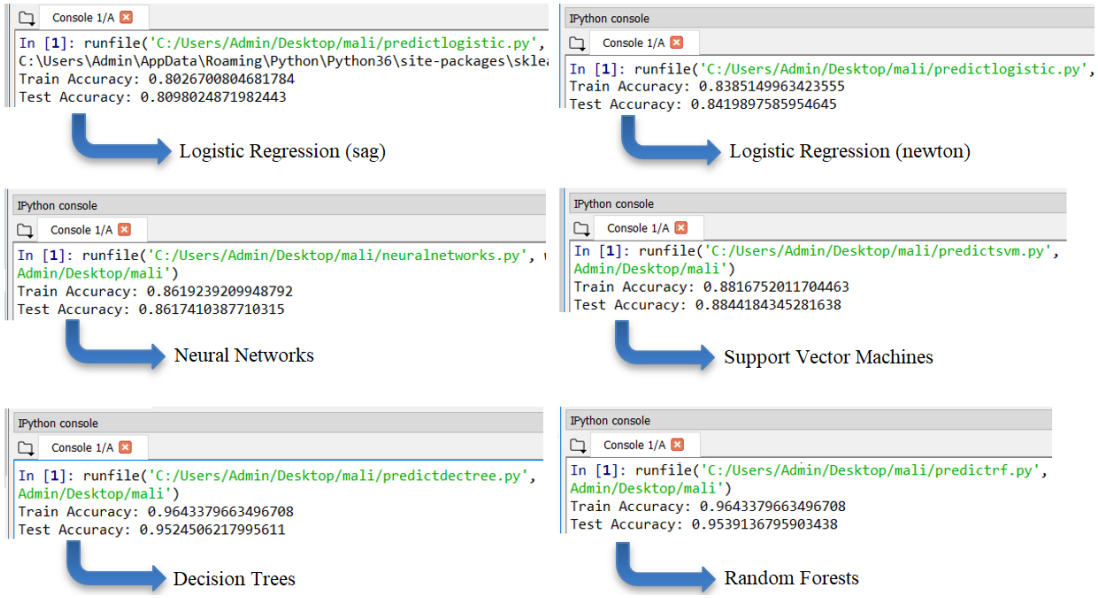
Şekil 4.24' de gösterilmekte olan sınıfa ait kodlar incelendiğinde ilk olarak gerekli kütüphaneler import edilmektedir. Daha sonra 9. Satırda pandas kütüphanesi yardımı ile, veri seti okunarak bir dataframe nesnesine aktarılmıştır. Dataframe nesnesinde yer alan boş veri içeren satırlar dropna metodu kullanılarak kaldırılmıştır. Veri setinin üzerinde gerçekleştirilen bir diğer işlem ise bağımlı ve bağımsız değişkenlerin belirlenmesidir. Veri setinin ilk sütununda yer alan sayfa türü etiketleri bağımlı değişken, geri kalan öznitelikler ise bağımsız değişken olarak belirlenmiştir. Değişkenlerin belirlenmesinden sonra veri setinden eğitim ve test sınıflarının belirlenmesine geçilmiştir. Şekil 4.24' te 15. satırda görüldüğü üzere, kullanılan veri setinin %80 lik kısmı eğitim, %20 lik kısmı ise test verisi olarak belirlenmiştir. Eğitim ve test setlerinin belirlenmesinden sonra 16. satırda kullanılacak sınıflandırma algoritmasına ait sınıf çağırılmış ve model nesnesi oluşturulmuştur. Algoritmanın belirlenmesinden sonra oluşturulan model, eğitim seti kullanılarak eğitilmiştir. Eğitme işlemi sonrasında oluşturulan modelin doğruluğu, test seti kullanılarak kontrol edilmiştir.

Bu tez çalışması kapsamında oluşturulan veri seti üzerinde sırası ile lojistik regresyon, yapay sinir ağları, destek vektör makineleri, karar ağaçları ve rastgele orman algoritmaları kullanılarak sınıflandırma işlemleri gerçekleştirilmiştir. Gerçekleştirilen sınıflandırma işlemlerinden elde edilen başarı oranları; Çizelge 4.23' te gösterilmiştir.

Çizelge 4.23. Sınıflandırma başarı oranları

Sınıflandırıcı	**Sınıflandırma Başarı Oranı**
Random Forest	%95.39
Decision Tree	%95.24
Support Vector Machines	%88.44
Neural Networks	%86.17
Logistic Regression (Newton)	%84.19
Logistic Regression (Sag)	%80.40

Çizelge 4.23' te gösterilmekte olan sınıflandırma başarı oranları incelendiğinde, en yüksek başarı oranı rastgele orman algoritması kullanılarak elde edilmiştir. Rastgele Orman algoritmasına en yakın başarı oranı karar ağaçları algoritmasından elde edilmiştir. Sınıflandırma işleminde en düşük başarı oranı lojistik regresyon algoritmasından elde edilmiştir. Lojistik regresyon algoritması kullanılarak gerçekleştirilen sınıflandırma işlemlerinde sag, saga, liblinear, lbfgs ve newton-ng çözümleyicileri kullanılmıştır. Kullanılan diğer çözümleyicilerden elde edilen doğruluk oranları yakın olduğundan dolayı çizelge içerisinde sag ve newton-ng çözümleyicilerine yer verilmiştir. Şekil 4.25' de sınıflandırma işlemleri için oluşturulmuş sınıfların çalıştırılması ile elde edilen ekran görüntüleri gösterilmiştir.



Şekil 4.25. Çeşitli algoritmalar için yazılan sınıflardan elde edilen doğruluk oranları

4.1.7.2. Sınıflandırma Modelinin Web Zafiyet Tarayıcısına Eklenmesi

Bir önceki başlık altında gerçekleştirilen sınıflandırma işlemlerinde en yüksek doğruluk oranı rastgele orman algoritmasından elde edilmiştir. Bu işlem sonrasında en yüksek doğruluk oranının elde edildiği model kaydedilerek web zafiyet tarayıcısı içerisine eklenmiştir. Kaydedilen modelin proje klasörü içerisine taşınmasından sonra, söz konusu modelin kullanılması için gerekli sınıfların yazılması işlemine geçilmiştir. Oluşturulan model kullanılarak sayfa sınıflandırma işleminin gerçekleştirilmesi için iki adet yeni sınıfa ihtiyaç duyulmaktadır. Bu sınıflardan ilki sınıflandırılacak sayfayı ziyaret edecek, sayfanın kaynak kodları içerisinden sınıflandırmada kullanılacak öznitelik değerlerini çıkaracak ve bunu ikinci sınıfa gönderecektir. İkinci sınıf ise, öznitelik çıkarıcı sınıf tarafından gönderilen verileri, daha önce kaydedilen sınıflandırma modeline sokacak ve sayfanın sınıfına karar verecektir. Şekil 4.26' da web zafiyet tarayıcısına eklenen randomForestClassifier.py sınıfında yer alan classifyPage metoduna ait kodlar verilmiştir.

```
def classifyPage(self):
    #load model
    self.features= pageFeatureExtractor(self.urlAdress, self.session).extractFeatures()
    for f in self.features:
        self.featuresArray.append(self.features[f])
    X_try = np.asarray([self.featuresArray])
    loaded_model = pickle.load(open('classification/rfmodel.sav', 'rb'))
    return loaded_model.predict(X_try)
```

Şekil 4.26. randomForestClassifier.py sınıfı classifyPage metodu

Şekil 4.26’ da gösterimekte olan metod incelendiğinde ilk olarak pageFeatureExtractor.py sınıfının kurucu metoduna parametre olarak öznitelik verileri çekilecek URL adresi gönderilmekte, daha sonra extractFeatures metodunun çalıştırılması söylenmektedir. pageFeaturesExtractor sınıfının kurucu metodu kendisine parametre olarak gönderilen URL adresini ziyaret ederek sayfanın kaynak kodlarını çekmektedir. Sayfanın kaynak kodları çekildikten sonra, söz konusu veriler, input arama ve sayı belirleme işlemlerinin daha kolay yapılabilmesi için bir BeautifulSoup nesnesine atanmıştır. Şekil 4.27’ de pageFeaturesExtractor sınıfı kurucu metoduna ait kodlar verilmiştir.

```
def __init__(self, url, session):
    self.urlAdress = url
    self.session=session
    self.general_headers = {'User-Agent': 'Mozilla/5.0 (Windows; U; Windows NT 6.1; rv:2.2) Gecko/20110201',
               'Accept-Language': 'en-US;',
               'Accept-Encoding': 'gzip, deflate',
               'Accept': 'text/html,application/xhtml+xml,application/xml;',
               'Connection': 'close'}
    self.features = {}
    self.contentOfUrl = self.session.get(self.urlAdress, headers=self.general_headers).content
    self.soup = BeautifulSoup(self.contentOfUrl.decode('utf-8', errors='ignore'), "html5lib")
```

Şekil 4.27. pageFeaturesExtractor.py sınıfı kurucu metodu

Kurucu metod tarafından oluşturulan BeautifulSoup nesnesi, extractFeatures sınıfı tarafından kullanılarak sayfaya ait öznitelik verileri çekilmektedir. Şekil 4.28’ de pageFeaturesExtractor sınıfı extractFeatures metodunun bir kısmına ait kodlar verilmiştir.

```
29      def extractFeatures(self):
30          try:
31              print("[*]Extracting features of page for classification. Url Address: " + self.urlAdress)
32
33              # number of forms
34              forms = self.soup.find_all("form")
35              self.features["Num Of Forms"] = len(forms)
36              # number of inputs
37              inputs = self.soup.find_all("input", attrs={"name": True})
38              self.features["Num Of Inputs"] = len(inputs)
39              # number of password areas
40              inputs = self.soup.find_all("input", attrs={"type": "password"})
41              self.features["Num Of Password Areas"] = len(inputs)
42              # number of search areas
43              inputs = self.soup.find_all("input", attrs={"type": "search"})
44              self.features["Num Of Search Areas"] = len(inputs)
45              # number of selects
46              inputs = self.soup.find_all("input", attrs={"type": "select"})
47              self.features["Num Of Select"] = len(inputs)
48              # number of radio
49              inputs = self.soup.find_all("input", attrs={"type": "radio"})
50              self.features["Num Of Radio"] = len(inputs)
51              # number of options
52              inputs = self.soup.find_all("input", attrs={"type": "option"})
53              self.features["Num Of Option"] = len(inputs)
```

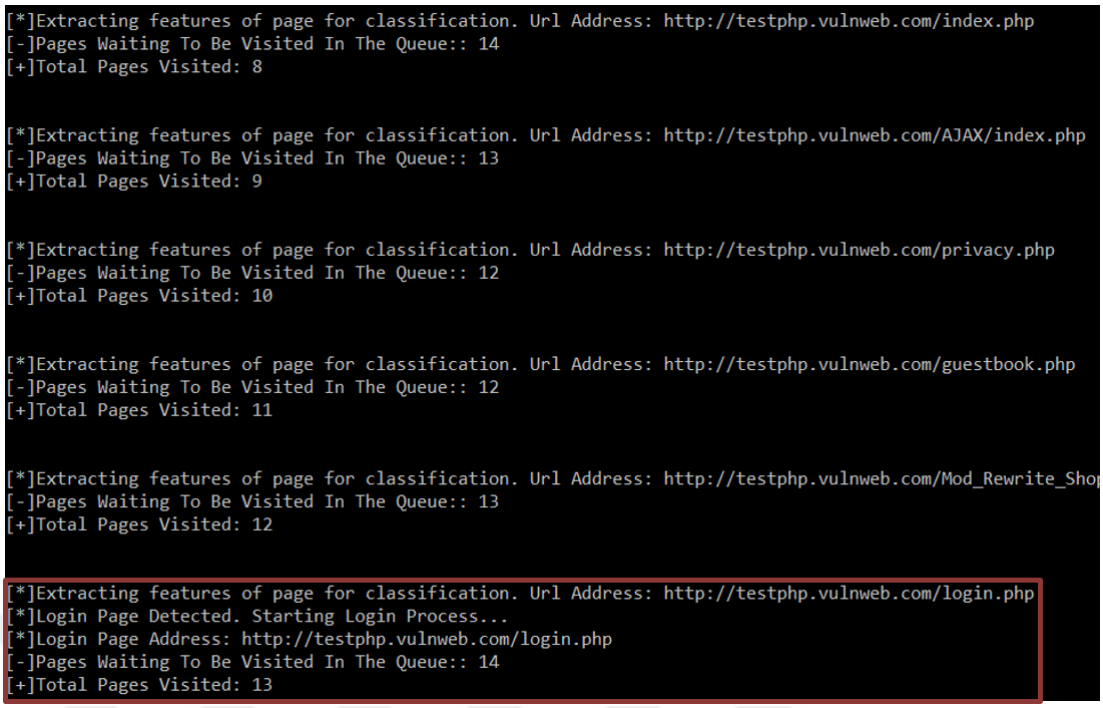
Şekil 4.28. pageFeaturesExtractor.py sınıfı extractFeatures metodu

Sınıflandırma modelinin web zafiyet tarayıcısına eklenmesi ve sonrasında söz konusu model kullanılarak web sayfalarının sınıflandırılmasını sağlayacak python sınıflarının yazılmasından sonra, bu işlevsellik geliştirilen aracın çeşitli noktalarında kullanılmaya başlanmıştır. Sayfa sınıflandırma işleminin kullanıldığı ilk nokta; web zafiyet tarayıcısının kapsam genişletme modülüdür. Web zafiyet tarayıcısı kapsam genişletme modülü; daha önce geliştirilmiş hali ile, web sayfalarında başarılı bir şekilde login olabilmektedir. Fakat bu işlemi gezdiği tüm sayfalardaki form alanlarına kullanıcı adı ve parola girerek yaptığı denemeler ile gerçekleştirmektedir. Daha net ifade etmek gerekirse, ziyaret ettiği sayfanın türü nedir bilmeden sayfa üzerindeki input alanlarına kullanıcı adı ve parola girmekte daha sonra sayfanın tepkisini ölçerek login olup olmadığını anlamaktadır. Bu yöntem mevcut kullanılan birçok web zafiyet tarayıcısının login modüllerine göre daha işlevsel ve başarılı çalışmasına rağmen, sayfa sınıflandırma işlevi eklenmiş bir kapsam belirleme modülü literatürde ilk olma özelliği taşımaktadır. Bu nedenle kapsam genişletme modülü içerisine sayfa sınıflandırma işlevi eklenmiştir. Bu sayede kapsam genişletme modülü çalışırken ziyaret edilen URL adresleri ilk olarak sınıflandırma işlemine tabi tutulmakta, eğer sayfa 'Login Page' olarak sınıflandırılırsa kullanıcı giriş işlemleri başlatılmaktadır. Söz konusu işlemlerin gerçekleştirildiği sınıf içerisinden kısa bir kod bloğu Şekil 4.29' de gösterilmiştir.

```
try:
    if Spider.isLoggendIn==False:
        typeOfPage = randomForestClassifier(partsOfString[1], Spider.session).classifyPage()
        if typeOfPage[0] == 'Login Page':
            print("[*]Login Page Detected. Starting Login Process...")
            print("[*]Login Page Address: "+partsOfString[1])
            Spider.contentOfLoginPage = Spider.session.get(partsOfString[1],headers=Spider.general_headers).content
            Spider.loginURL = partsOfString[1]
            currentUrlAdress = Spider.loginWithSelenium()
```

Şekil 4.29. spider.py sınıfı kullanıcı giriş işlemleri

Şekil 4.29' daki kodlar incelendiğinde, sınıflandırıcı metod kullanılarak sayfa türünün tahmin edildiği, eğer sayfa türü 'Login Page' ise kullanıcı giriş işlemlerinin başlatıldığı görülmektedir. Şekil 4.30' da http://testphp.vulnweb.com adresinin taranması esnasında, login sayfasının otomatik tespit edilmesi ve giriş yapılması görülmektedir.



Şekil 4.30. Login sayfasının otomatik olarak tespit edilmesi ve giriş yapılması

Web zafiyet tarayıcısı içerisinde sınıflandırma işlevinin kullanıldığı bir diğer modül, zafiyet test modülüdür. Bu işlem bir sonraki başlık altında Apriori algoritması ile birliktelik analizi yapılması sonrasında anlatılacaktır.

4.1.8. Apriori Algoritması ile Birliktelik Analizi

Bu çalışma kapsamında oluşturulan veri seti içerisinde 7781 satır örnek bulunmaktadır. Bu örnekler, sayfa tipi, keşfedilen zafiyet ve keşfedilen metod türü (GET ve POST) yönünden farklılık göstermektedir. Gerçekleştirilen testler sonrasında elde edilen zafiyetlerin 1778 tanesi URL üzerinden yani http GET metodu kullanılarak tespit edilmiştir. Geri kalan 6003 zafiyet ise, web sayfaları üzerinde yer alan formlar üzerinde yani POST metodu kullanılarak tespit edilmiştir. Yapılan çalışmalar sonrasında URL üzerinden yani GET metodu kullanılarak gerçekleştirilen testlerde elde edilen zafiyetlerin, sayfa türleri ile ilişkilendirilmesinin mümkün olmadığı görülmüştür. Çünkü sayfa türlerinin sınıflandırılmasında; sayfa üzerinde yer alan formlar, formlar içerisinde yer alan kullanıcı girdi alanları ve butonlar kullanılmaktadır. URL adresi üzerinden tespit edilen zafiyetlerde ise, sayfa üzerinde yer alan bileşenler bir önem taşımamaktadır. Gerçekleştirilen testlerde URL adresi parçalanmakta, ilgili alanlara payloadlar gömülmekte ve saldırı denemeleri

gerçekleştirilmektedir. Bu nedenler Apriori algoritması kullanılarak gerçekleştirilen birliktelik analizinde sadece sayfa üzerindeki bileşenlerde tespit edilen zafiyetler (POST metodu üzerinden tespit edilen zafiyetler) kullanılmıştır.

Gerçekleştirilen tez kapsamında ilk olarak eldeki veri setinin Apriori algoritmasında kullanılmak üzere düzenlenmesi gerekmektedir. Apriori algoritması kullanılarak birliktelik analizi yapılacak bir veri setinin yapısı satırlardan oluşmaktadır. Her satırda aralarındaki ilişkiye bakılacak verilerin birlikte olduğu örnekler yer alacaktır. Söz konusu işlemi veri setinde yer alan bir örnek üzerinde açıklamak için Şekil 4.31 kullanılmıştır.

7336	http://localhost/cms/admin/category.php	List Page	BLIND SQLI	POST	'/**/or/**	jQuery UI	Font Awes	Twitter Bc	Apache	PHP
7337	http://localhost/cms/admin/category.php	List Page	BLIND SQLI	POST	'/**/or/**	jQuery UI	Font Awes	Twitter Bc	Apache	PHP
7338	http://localhost/cms/admin/category.php	List Page	LFI	POST	/usr/local,	jQuery UI	Font Awes	Twitter Bc	Apache	PHP
7339	http://localhost/cms/admin/category.php	List Page	LFI	POST	/home2\b	jQuery UI	Font Awes	Twitter Bc	Apache	PHP
7340	http://localhost/cms/admin/category.php	List Page	LFI	POST	/home2\b	jQuery UI	Font Awes	Twitter Bc	Apache	PHP
7341	http://localhost/cms/admin/category.php	List Page	COMMAND E	POST	%26echo '	jQuery UI	Font Awes	Twitter Bc	Apache	PHP
7342	http://localhost/cms/admin/category.php	List Page	COMMAND E	POST	%7C"/etc/	jQuery UI	Font Awes	Twitter Bc	Apache	PHP
7343	http://localhost/cms/admin/category.php	List Page	COMMAND E	POST	%26echo '	jQuery UI	Font Awes	Twitter Bc	Apache	PHP
7344	http://localhost/cms/admin/category.php	List Page	SSTI	POST	{{1336%2E	jQuery UI	Font Awes	Twitter Bc	Apache	PHP
7345	http://localhost/cms/admin/category.php	List Page	SSTI	POST	{{1336%2E	jQuery UI	Font Awes	Twitter Bc	Apache	PHP
7346	http://localhost/cms/admin/category.php	List Page	SSTI	POST	{{1336%2E	jQuery UI	Font Awes	Twitter Bc	Apache	PHP

Şekil 4.31. Kullanılan veri setinden bir kesit

Şekil 4.31' de gösterilmekte olan örnek incelendiğinde; "http://localhost/cms/admin/category.php" adresindeki web sayfası üzerinde yer alan farklı input elemanlarında Blind SQLi, Local File Inclusion, Command Execution ve Server Side Template Injection zafiyetleri tespit edilmiştir. Bu çalışmada Apriori algoritması, daha önce de bahsedildiği üzere zafiyetler arasındaki ilişkinin tespit edilmesi için kullanılmıştır. Zafiyetler arası birliktelik analizinin yapılabilmesi için, veri setindeki her URL adresi üzerinde tespit edilen zafiyetlerin araya virgül konularak kaydedilmesi gerekmektedir. Şekil 4.31' de gösterilmekte olan URL adresi üzerinde tespit edilen zafiyetler Apriori algoritmasında kullanılacak veri seti içerisine; "BLIND SQLI, LFI, COMMAND EXEC, SSTI" şeklinde kaydedilmiştir. Eğer söz konusu URL adresi üzerinde tek bir zafiyet tespit edilmiş olsaydı, veri setine sadece o zafiyet eklenecekti. İçerisinde 7000 satırın üzerinde örnek barındıran veri setinden, Apriori algoritması tarafından kullanılacak veri setinin elde edilmesi işlemi manuel yapıldığında çok zaman alacağından, yardımcı bir araç geliştirilmiştir. Şekil 4.32' de söz konusu araca ait kodlar gösterilmiştir.

```
kitap = load_workbook("dataset.xlsx")
sheet=kitap.active
tempURL=""
tempType=""
tempVuln=set()
dict={}
counter=0
listofVulns=[]
for row in sheet.iter_rows(min_row=1, min_col=1, max_row=6300, max_col=2):
    if tempURL=="":
        tempURL=row[0].value
        tempVuln.add(row[1])
    elif tempURL==row[0].value:
        tempVuln.add(row[1].value)
    elif tempURL!=row[0].value and tempURL!="":
        tempString=""
        for t in tempVuln:
            tempString+=str(t)+","
        tempString=tempString[:len(tempString)-1]
        listofVulns.append(tempString)
        tempURL=row[0].value
        tempVuln.clear()
        tempVuln.add(row[1].value)
```

Şekil 4.32. Apriori algoritması tarafından kullanılacak veri setinin oluşturulması için geliştirilen araç

Kullanılacak veri setinin oluşturulmasından sonra, birliktelik analizi işlemi gerçekleştirilmiştir. Birliktelik analizi işlemi, yine Python programlama dili kullanılarak Spyder ortamında gerçekleştirilmiştir. Söz konusu işleme ait kaynak kodlar Şekil 4.33' de verilmiştir.

```
import pandas as pd
import numpy as np

from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules

import mlxtend as ml
#read file contents and add dataset
dataset =[]

file=open("iliski23112019v03.csv", "r")
contents = file.readlines()
for line in contents:
    subLine=line.strip()
    dataset.append(subLine.split(','))

file.close()

te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)

result = apriori(df, min_support=0.02, use_colnames=True)
rules1 = association_rules(result, metric="confidence", min_threshold=0.3)
rules1["antecedent_len"] = rules1["antecedents"].apply(lambda x: len(x))
rules1["consequents_len"] = rules1["consequents"].apply(lambda x: len(x))
rules1 = rules1.sort_values(['confidence'], ascending=False)
```

Şekil 4.33. Apriori algoritması ile birliktelik analizi

Şekil 4.33’ de gösterilmekte olan kodlar incelendiğinde, ilk olarak birliktelik analizinde kullanılacak kütüphaneler proje içerisine dahil edilmiştir. Daha sonra veri seti satır satır okunmuş, okunan her bir satır, gereksiz karakterlerden ayrıldıktan sonra virgüle göre parçalanarak bir liste içerisine atılmıştır. Virgüle göre parçalama işlemi sonunda iki boyutlu bir matris elde edilmiştir. Söz konusu liste daha sonra Apriori algoritması için kullanıma hazır hale getirilerek algoritmaya parametre olarak gönderilmiştir.

Apriori algoritmasında dikkat edilmesi gereken 2 parametre bulunmaktadır. Bunlar; Şekil 4.33’ de 24. Satırda görülen min_support ve 25 satırda görülen min_treshold değerleridir. Min suppor değeri adından da anlaşılacağı üzere Apriori algoritması tarafından alınacak destek (suppot) eşiğidir. Materyal ve Yöntem başlığı altında işlendiği üzere destek değeri, bir elemanın (örneğin SQLi) bulunduğu örnek sayısının, veri setindeki tüm örnek sayısına oranı anlamına gelmektedir. Apriori algoritması için varsayılan destek değeri 0.1’ dir. Bu değer %10 anlamına gelmektedir. Yani içinde bulunduğu örnek sayısının tüm örnek sayısına oranı en az %10 olan zafiyetlere ait birliktelik sonuçları gösterilecektir. Şekil 4.34’ de support=0.1, confidence=0.8 değerlerinde elde edilen birliktelik kuralları gösterilmektedir.

Index	antecedents	consequents	ntecedent suppoı	onsequent suppoı	support	confidence
2	frozenset({'HTMLI'})	frozenset({'XSS'})	0.120924	0.180707	0.111413	0.921348
3	frozenset({'DTRAV', 'COMMAND EXEC'})	frozenset({'LFI'})	0.131793	0.307065	0.11413	0.865979
5	frozenset({'DTRAV', 'PHPI'})	frozenset({'LFI'})	0.141304	0.307065	0.119565	0.846154
0	frozenset({'SQLI'})	frozenset({'BLIND SQLI'})	0.226902	0.557065	0.186141	0.820359
1	frozenset({'PHPI'})	frozenset({'COMMAND EXEC'})	0.383152	0.456522	0.309783	0.808511
4	frozenset({'PHPI', 'LFI'})	frozenset({'COMMAND EXEC'})	0.210598	0.456522	0.169837	0.806452

Şekil 4.34. Support=0.1, confidence=0.8 değerleri birliktelik kuralları

Şekil 4.34’ de gösterilmekte olan kurallar incelendiğinde; veri setinde en çok karşılaşılan zafiyetlere ait kurallar elde edilmiştir. Ayrıca zafiyetler arası 1’e 1 kural sayısı da oldukça az elde edilmiştir. Söz konusu sonucun bu denli yetersiz çıkmasının ilk nedeni destek değerinin 0.1 olarak belirlenmesidir. Çizelge 4.24’ te birliktelik analizine tabi tutulan veri seti içerisinde yer alan zafiyetlerin sayısı gösterilmektedir.

Çizelge 4.24. Veri setinde yer alan zafiyet sayıları

Zafiyet Tipi	**Adet**
Blind SQLi	1565
Command Exec	852
DTRAV	774
PHPI	704
LFI	642
SQLi	538
HTMLI	262
XSS	320
SSTI	216
SSI	143
Buffer Overflow	13
CRFL	2
XEE	2

Çizelge 4.24' de zafiyet sayıları incelendiğinde her zafiyetin farklı sayıda olduğu görülmektedir. Web zafiyetleri doğaları gereği her türde web sayfası içinde görülebilmektedir. Bir web sayfası türünde, bir tür zafiyetin kesinlikle görülemeyeceğini söylemek yanlış bir ifade olacaktır. Web zafiyetleri ile ilgili kesin kanılara varılamamasının nedeni, her web programcının farklı tarzda programlama alışkanlığının olması ve güvenli programlamaya farklı seviyede önem vermesidir.

Veri seti incelendiğinde en çok kör SQL enjeksiyonu ile karşılaşılmıştır. XSS zafiyeti ise, kör SQL enjeksiyonuna oranla daha az tespit edilmiştir. Dolayısı ile veri setinde XSS zafiyetinin olduğu örnek sayısı daha az olacaktır. Bu sonucu, XSS zafiyetinin kör SQL enjeksiyonundan daha önemsiz bir zafiyet olduğu şeklinde yorumlamak yanlıştır. Bu nedenle üzerinde çalışılmakta olan veri seti üzerinden mümkün olduğunca her zafiyet için bir birliktelik kuralı oluşturmak gerekmektedir. Bunu sağlamanın yolu ise, %10 olarak belirlenen destek değerini daha aşağıya çekmek olacaktır. Çizelge 4.20' de verilen rakamlar incelendiğinde, en az karşılaşılan CRLF ve bellek taşması zafiyetlerinin analiz dışı bırakılmasına karar verilmiştir. Bunun nedeni, söz konusu zafiyetler ile ilgili mantıklı birliktelik kararlarının çıkarılamayacak olmasıdır. Bu tez çalışmasında; CRLF ve bellek taşması zafiyetinden sonra en az sayıda tespit edilen,

fakat mantıklı birliktelik kuralları oluşturmaya yetecek kadar örnek sayısına sahip SSI zafiyetini kapsayacak bir destek eşiği belirlenmiştir. Söz konusu işlemde destek eşiği 0.02 olarak belirlenmiştir.

Gerçekleştirilen ilk birliktelik analizinde elde edilen kuralların az sayıda elde edilmesinin bir diğer nedeni de confidence (güven) eşiğinin daha önce belirtildiği üzere %80 olarak belirlenmesidir. Yüksek seviyede belirlenen güven eşiği, veri setinde çok sayıda örnekte karşılaşılan Blind SQLi gibi zafiyetler için birliktelik kuralı elde edilmesini zorlaştırmaktadır. Günümüzde birçok web geliştirici hata tabanlı SQL enjeksiyonuna karşı yüksek farkındalık ve engelleme içgüdüsüne sahiptir. Fakat aynı durumu zaman tabanlı kör SQL enjeksiyonu için söylemek doğru olmayacaktır. Bu nedenle test edilen sayfalarda bulunan zaman tabanlı kör SQL enjeksiyonunun sayısı, hata tabanlı SQL enjeksiyonu sayısına göre fazla tespit edilmiştir. İki zafiyet arasında birliktelik analizi gerçekleştirildiğinde zaman tabanlı kör SQL enjeksiyonunun görüldüğü sayfalarda, hata tabanlı SQL enjeksiyonu görülme olasılığı %33 olarak tespit edilmiştir. Bu oran başlangıçta düşük olarak görülse dahi, temelde aynı mantıkla test edilen, sadece test için kullanılan payloadları farklı olan bu zafiyetler arasında ilişkinin az olduğundan bahsetmek yanlış olacaktır. Bu nedenler gerçekleştirilen birliktelik analizi işleminde destek eşik değeri 0.02, güven eşik değeri de 0.3 olarak belirlenmiştir. Şekil 4.35’ de gerçekleştirilen birliktelik analizine ait kural tablosu verilmiştir.

Index	antecedents	consequents	ntecedent suppor	onsequent suppor	support	confidence
15	frozenset({'SSI'})	frozenset({'COMMAND EXEC'})	0.0421196	0.456522	0.0326087	0.774194
21	frozenset({'SSI'})	frozenset({'DTRAV'})	0.0421196	0.241848	0.03125	0.741935
29	frozenset({'SSI'})	frozenset({'LFI'})	0.0421196	0.307065	0.03125	0.741935
31	frozenset({'SSI'})	frozenset({'PHPI'})	0.0421196	0.383152	0.0326087	0.774194
36	frozenset({'SSI'})	frozenset({'SSTI'})	0.0421196	0.0597826	0.0217391	0.516129
8	frozenset({'SSTI'})	frozenset({'BLIND SQLI'})	0.0597826	0.557065	0.0285326	0.477273
16	frozenset({'SSTI'})	frozenset({'COMMAND EXEC'})	0.0597826	0.456522	0.048913	0.818182
22	frozenset({'SSTI'})	frozenset({'DTRAV'})	0.0597826	0.241848	0.0380435	0.636364
30	frozenset({'SSTI'})	frozenset({'LFI'})	0.0597826	0.307065	0.0434783	0.727273
32	frozenset({'SSTI'})	frozenset({'PHPI'})	0.0597826	0.383152	0.0394022	0.659091
35	frozenset({'SSTI'})	frozenset({'SSI'})	0.0597826	0.0421196	0.0217391	0.363636
3	frozenset({'HTMLI'})	frozenset({'BLIND SQLI'})	0.120924	0.557065	0.0747283	0.617978
24	frozenset({'HTMLI'})	frozenset({'SQLI'})	0.120924	0.226902	0.0720109	0.595506
25	frozenset({'HTMLI'})	frozenset({'XSS'})	0.120924	0.180707	0.111413	0.921348
9	frozenset({'XSS'})	frozenset({'BLIND SQLI'})	0.180707	0.557065	0.116848	0.646617
26	frozenset({'XSS'})	frozenset({'HTMLI'})	0.180707	0.120924	0.111413	0.616541
34	frozenset({'XSS'})	frozenset({'SQLI'})	0.180707	0.226902	0.13587	0.75188
7	frozenset({'SQLI'})	frozenset({'BLIND SQLI'})	0.226902	0.557065	0.186141	0.820359
23	frozenset({'SQLI'})	frozenset({'HTMLI'})	0.226902	0.120924	0.0720109	0.317365
33	frozenset({'SQLI'})	frozenset({'XSS'})	0.226902	0.180707	0.13587	0.598802
2	frozenset({'DTRAV'})	frozenset({'BLIND SQLI'})	0.241848	0.557065	0.108696	0.449438
10	frozenset({'DTRAV'})	frozenset({'COMMAND EXEC'})	0.241848	0.456522	0.131793	0.544944
17	frozenset({'DTRAV'})	frozenset({'LFI'})	0.241848	0.307065	0.163043	0.674157
19	frozenset({'DTRAV'})	frozenset({'PHPI'})	0.241848	0.383152	0.141304	0.58427
4	frozenset({'LFI'})	frozenset({'BLIND SQLI'})	0.307065	0.557065	0.127717	0.415929
12	frozenset({'LFI'})	frozenset({'COMMAND EXEC'})	0.307065	0.456522	0.233696	0.761062
18	frozenset({'LFI'})	frozenset({'DTRAV'})	0.307065	0.241848	0.163043	0.530973
27	frozenset({'LFI'})	frozenset({'PHPI'})	0.307065	0.383152	0.210598	0.685841
5	frozenset({'PHPI'})	frozenset({'BLIND SQLI'})	0.383152	0.557065	0.127717	0.333333
14	frozenset({'PHPI'})	frozenset({'COMMAND EXEC'})	0.383152	0.456522	0.309783	0.808511
20	frozenset({'PHPI'})	frozenset({'DTRAV'})	0.383152	0.241848	0.141304	0.368794
28	frozenset({'PHPI'})	frozenset({'LFI'})	0.383152	0.307065	0.210598	0.549645
1	frozenset({'COMMAND EXEC'})	frozenset({'BLIND SQLI'})	0.456522	0.557065	0.168478	0.369048
11	frozenset({'COMMAND EXEC'})	frozenset({'LFI'})	0.456522	0.307065	0.233696	0.511905
13	frozenset({'COMMAND EXEC'})	frozenset({'PHPI'})	0.456522	0.383152	0.309783	0.678571
0	frozenset({'BLIND SQLI'})	frozenset({'COMMAND EXEC'})	0.557065	0.456522	0.168478	0.302439
6	frozenset({'BLIND SQLI'})	frozenset({'SQLI'})	0.557065	0.226902	0.186141	0.334146

Şekil 4.35. Support=0.02, confidence=0.3 değerleri birliktelik kuralları

Veri setine göre gerçekleştirilen birliktelik analizi sonrası kuralların belirlenmesinden sonra, söz konusu kuralların web zafiyet tarayıcısına entegre edilmesi işlemi gerçekleştirilmiştir. Hali hazırda veri seti kullanılarak gerçekleştirilen sayfa sınıflandırma işlemi ile zafiyetler arası birliktelik analizinden elde edilen birliktelik kurallarının birleştirilmesinde, sayfa türleri üzerinde bulunan zafiyetlerin sayısı kullanılmıştır. Çizelge 4.25' te, sınıflandırma ve birliktelik analizinde kullanılan veri setinde, her bir sayfa türünde tespit edilen zafiyetlerin sayısı gösterilmektedir.

Çizelge 4.25. Sayfa türüne göre tespit edilen zafiyet sayıları

Sayfa / Zafiyet Türü	Submit Page	Register Page	Login Page	List Page	Info Page	Search Page	Contact Us Page	Forgot Password Page
Blind SQLi	723	271	366	28	50	41	60	26
Command Exec	573	50	31	40	90	18	38	12
DTRAV	532	55	37	24	8	18	38	32
PHPI	468	40	34	31	68	16	34	13
SSTI	123	34	6	49	1	3	0	0
LFI	457	53	36	32	16	14	23	11
SQLi	227	134	93	2	5	51	24	2
HTMLI	131	83	6	1	2	35	4	0
Open Redirect	49	120	127	0	56	4	3	1
SSI	80	37	4	14	9	5	3	0
XSS	49	88	66	3	2	103	8	1
Buffer Overflow	0	0	4	0	9	1	0	0
CRFL	0	0	2	0	1	0	0	0
XXE	0	0	0	2	0	0	0	0

Sayfa sınıflandırma sonrasında birliktelik analizinin kullanılması işleminde daha önce de bahsedildiği üzere, bir sayfa tipinde en fazla görülen zafiyetler kullanılacaktır. Söz konusu işlemin algoritması şu şekilde işleyecektir; ilk olarak test edilen URL adresinin sayfa tipi belirlenecektir. Sayfa tipinin belirlenmesinden sonra, söz konusu sayfa tipinde en çok görülen zafiyetlere ait ilk 5 test sınıfı belirlenecektir. Bu işlemden sonra ilk test sınıfı çağırılacaktır. Eğer test edilen zafiyet sayfa üzerinde tespit edilirse, bu kez birliktelik analizinden elde edilen kural tablosuna başvurulacak ve tespit edilen zafiyetin görüldüğü sayfada görülme olasılığı en yüksek olan zafiyetler test edilecektir. Sayfa üzerinde tespit edilen her zafiyet için önce birliktelik kuralına bakılacak ve ilişkili olduğu zafiyetler test edilecektir. Eğer birliktelik kuralında test edilecek zafiyet kalmadıysa, sistem tekrar geriye dönüp sayfa üzerinde en sık görülmüş sıradaki test edilmemiş zafiyet ile işleme devam edecektir. En sık görülen zafiyetler listesinde ve

birliktelik kurallarında test edilecek zafiyet kalmadıysa tarama sonlanacaktır. Elde edilen verilerin raporlanmasından önce, eğer tarama esnasında hakkında birliktelik kuralı bulunmamış ve taranmamış sınıf kalmışsa, raporlama öncesinde bu sınıfların da taranmasının istenip istenmediği, kullanıcıya sorulacaktır. Kurulan bu algoritma sayesinde standart web zafiyet tarayıcıların yaptığı gibi kaba kuvvet yöntemi ile tüm test sınıflarının kullanılması yerine, zafiyet test işlemi belli bir mantığa oturtulmuş olacaktır. Bu sayede birbiri ile alakalı zafiyetler ard arda taranacaktır. Bu sistemin bir diğer avantajı birbirini tetikleme ihtimali olan, birbiri ile ilişkili zafiyetlerin ard arda taranması ile test işlemlerinin daha başarılı sonuçlar üretmesidir. Geliştirilen algoritmanın bir diğer avantajı da bir sayfanın test edilmesinde daha az test sınıfının kullanılacak olmasıdır. Bu sayede, web sayfasının ve veri tabanının yapısını bozma ihtimali olan, sayfa üzerinde görülme olasılığı çok düşük zafiyetlere ait test sınıfları çalıştırılmayacaktır.

4.1.8.1. Apriori Algoritması ile Elde Edilen Birliktelik Kurallarının Web Zafiyet Tarayıcısına Eklenmesi

Apriori algoritması kullanılarak gerçekleştirilen birliktelik analizinin, geliştirilen web zafiyet tarayıcısına eklenmesinde yapılan ilk işlem; elde edilen kuralların uygulama içerisinde tanımlanmasıdır. Her bir zafiyet için elde edilen kuralların tanımlanmasında Python programlama dilinin sözlük yapısı kullanılmıştır. Şekil 4.36' da generalAttackClass.py sınıfı içerisinde tanımlanan birliktelik kuralları gösterilmektedir.

```
attackAttack_map = {'sqli.py': ['blindsqli.py', 'xss.py', 'htmli.py'],
                    'blindsqli.py':['sqli.py','oscommandexec.py'],
                    'oscommandexec.py':['phpi.py','lfi.py','blindsqli.py'],
                    'phpi.py':['oscommandexec.py','lfi.py','lindtraversal.py','windtraversal.py','blindsqli.py'],
                    'ssi.py':['oscommandexec.py','phpi.py','lfi.py','lindtraversal.py','windtraversal.py','ssti.py'],
                    'ssti.py':['oscommandexec.py','lfi.py','phpi.py','lindtraversal.py','windtraversal.py','blindsqli.py','ssi.py'],
                    'lfi.py':['oscommandexec.py','phpi.py','lindtraversal.py','windtraversal.py','blindsqli.py'],
                    'lindtraversal.py':['lfi.py','phpi.py','oscommandexec.py','blindsqli.py'],
                    'windtraversal.py': ['lfi.py', 'phpi.py', 'oscommandexec.py', 'blindsqli.py'],
                    'xss.py':['sqli.py','blindsqli.py','htmli.py'],
                    'htmli.py':['xss.py','blindsqli.py','sqli.py']}
```

Şekil 4.36. Birliktelik analizinden elde edilen kuralların tanımlanması

Şekil 4.36' da gösterilmekte olan kodlar incelendiğinde, her bir zafiyet test sınıfı için elde edilen birliktelik kuralları, yüksek güven değerine sahip olandan düşük güven değerli olana doğru tanımlanmıştır. Tanımlanma işleminde test sınıfları anahtar, ilişkili

olduğu sınıflar ise değer olarak atanmıştır. Böylelikle bir zafiyet tespit edildiğinde anahtar değere ait değer listesi çağırılacak ve elemanları test edilecektir.

Bir önceki başlıkta anlatıldığı üzere, test edilecek sayfanın türü belirlendikten sonra zafiyet test işlemi başlayacaktır. Zafiyet test işlemi, tespit edilen sayfa türü üzerinde veri setine göre en çok görülen zafiyet test sınıfının çalıştırılması ile başlayacaktır. Bu sebeple sayfa türlerine göre en fazla görülen zafiyet listeleri, yine sözlük formatında uygulama içerisinde tanımlanmıştır. Sayfa sınıflandırma, sayfa türüne göre test başlatma, tespit edilen zafiyete göre ilişkili test sınıflarını çalıştırma işlemlerinde kullanmak üzere geliştirilmiş attackOverPost metodu Şekil 4.37' de gösterilmektedir.

```
280 def attackOverPost(page_url,session,loginUrl,username,password,contentofloginpage):
281
282     attack_result_map = {}  # başlangıçta boş, test sonuçlarına göre true ya da false döndürüyor
283
284     # çağırılmadan çalışmayacak
285     def attack_method(attack_type, attack_url):
286         if attack_type in attack_result_map:  # daha önce test edildiyse, true ya da false o sonucu al attack_result içine at
287             attack_result = attack_result_map[attack_type]
288         else:  # eğer daha önce test edilmediyse önce test et, oradan dönen true false yanıtı result_mape kaydet
289             listOfUsedTestClasses.append(attack_type)
290             attack_result = test_attack(attack_type,attack_url,"POST",session,loginUrl,username,password,contentofloginpage)
291             attack_result_map[attack_type] = attack_result
292
293         if attack_result:  # eğer test edilen zafiyet başarılı ise, ya da daha önce test edilip başarılı bulunduysa
294             for s in attackAttack_map[attack_type]:  # onunla ilişkili sayfaları test etmeye başla
295                 if s not in attack_result_map:  # eğer ilişkili zafiyet daha önce test edilmediyse tet işlemini bunun için ba
296                     attack_method(s,page_url)
297
298     page_type = detect_type(page_url,session,loginUrl,username,password,contentofloginpage)
299     print("[*]Type of Page: "+page_type)
300     for a in pageAttack_map[page_type]:
301         attack_method(a,page_url)
```

Şekil 4.37. attackOverPost metodu

Şekil 4.37' de gösterilmekte olan metod incelendiğinde ilk olarak attack_result_map isminde, sözlük tipinde bir değişkenin tanımlandığı görülmektedir. Bu değişken test edilen URL adresi üzerinde; hem hangi test sınıflarının çalıştığını, hem de çalışan test sınıflarından hangilerinin zafiyet tespit ettiğini kontrol etmek için kullanılmaktadır. Böylelikle bir kez çalışan bir test metodu tekrar çalıştırılmayacak, tespit edilen bir zafiyet ile ilgili test metodları da çağırılabilecektir. Şekil 4.37' de gösterilen metod ayrıca içerisinde recursive bir yapı içermektedir. Bu sayede, bir sayfa türünde en çok karşılaşılan zafiyetlerden, tespit edilen zafiyet ile ilgili sınıflar çalıştırıldıktan sonra, araç kaldığı yerden diğer en çok görülen zafiyetleri test etmeye devam edebilecektir. Kullanılan yöntemin daha iyi anlaşılması adına örnek bir URL adresi üzerinde gerçekleştirilen test adımları aşağıda numaralandırılmıştır.

1. http://teztest.com/addinventory.php URL adresini alınır ve sınıflandırıcıya sokulur.
2. Sınıflandırıcıdan geriye "Submit Page değeri döner. Test işlemi Search Page türünde en çok görülen zafiyetlerden ilki olan blind SQLi ile başlar. (**1. Blind SQLi**, 2. Command Execution, 3. Directory Traversal, 4. Code Injection)
3. Sayfa üzerinde blind SQLi zafiyeti tespit edilir. Tarama işlemine blind SQLi ile birlikte en çok görülme olasılığı olan zafiyet olan SQLi ve da sonra command execution ile devam edilir.
4. Test edilen her iki zafiyet tespit edilemez. Kural tablosuna göre blind SQLi ile ilgili başka zafiyet kalmadığından tarama işlemi, Submit page üzerinde en çok görülen zafiyetlerden sıradaki eleman ile devam eder.
5. Submit page' de en çok görülenler listesinde 2. Eleman olan command execution zafiyeti, blind SQLi ile olan ilişkisinden dolayı daha önce test edilmiştir. Dolayısıyla test işlemi, submit page' de en çok görülen 3. Zafiyet olan directory traversal testi ile devam edecektir.
6. Test işlemi, sıralı listede ve birliktelik listelerinde çalıştırılacak test sınıfı kalmayana kadar devam edecektir.
7. Sıralı listede ve birliktelik listesinde test için çalıştırılacak sınıf kalmadığında, testler sonuçlanacaktır. Fakat raporlama öncesinde son adım olarak, yapılan tarama işleminde, kendisi hakkında birliktelik kuralı bulunmamış, dolayısıyla çalıştırılmamış test sınıflarını yine de çalıştırmak isteyen kullanıcılara, kalan test sınıflarını çalıştırmak isteyip istemedikleri sorulmaktadır.
8. Kullanıcı tercihine göre gerekli işlem varsa yürütüldükten sonra, raporlama adımına geçilmekte ve tarama işlemi tamamlanmaktadır.

İlişki analizinin web uygulama zafiyet tarayıcısı içerisine eklenmesi ile geliştirilen araç,

- Hem GET hem de POST üzerinden test yapabilme,
- Bilgisayar çekirdek sayısına göre dinamik olarak üretilen threadler ile multi threading çalışabilme,
- Sadece kullanıcı adı ve parola bilgisi vermek suretiyle, kullanıcı giriş sayfalarını otomatik tarayarak giriş yapabilme ve kullanıcı panelindeki sayfaları tarayabilme,

- Literatürdeki diğer zafiyet tarayıcılarda olduğu gibi tüm test sınıflarını doğrudan çalıştırmak yerine, birbiri ile ilişkili test sınıflarını ardı sıra çalıştırarak daha başarılı ve mantıklı test sonuçları üretebilme yeteneklerine sahiptir.

Oluşturulan web zafiyet tarayıcısı ve gerçekleştirilen işlemlerin, yaygın olarak kullanılan diğer zafiyet tarayıcılar ve literatürdeki diğer çalışmalar ile kıyaslanması tartışma ve sonuçlar bölümünde ele alınmıştır.

5. TARTIŞMA VE SONUÇLAR

Bu tez çalışmasında, yapay zekâ ve dinamik analiz tabanlı bir web uygulama zafiyet tarayıcısı geliştirilmiştir. Bu tez kapsamında yapılan çalışmalar ve geliştirilen araç, çeşitli yönlerden literatürdeki geçmiş çalışmalara göre daha iyi sonuçlar vermektedir. Yapılan çalışmanın literatürde yer alan diğer çalışmalar ile kıyaslanması, her bir modül için ayrı olarak işlenecektir.

Geliştirilen web zafiyet tarayıcısı, güvenlik uzmanları tarafından web sızma testlerinde yaygın olarak kullanılan açık kaynak kodlu iki tarayıcı olan WASCAN ve WAPITI ile karşılaştırıldığında test ettiği zafiyet türü sayısına göre üstünlük sağlamaktadır. Kali işletim sistemi içerisinde varsayılan olarak yüklü gelen WASCAN 16 farklı zafiyeti test etmektedir. Kara kutu web sızma testlerinde yaygın olarak kullanılan WAPITI ise, 12 farklı zafiyeti test edebilmektedir. Bu tez çalışması kapsamında geliştirilen web zafiyet tarayıcısı, bünyesinde 21 farklı zafiyet test sınıfı barındırmaktadır. Ayrıca zafiyet test sınıfları, yeni bir zafiyet ekleneceği zaman, uyarlanması çok kolay olacak şekilde planlanmış ve kodlanmıştır. Geliştirilen araca yeni bir zafiyet eklenmek istendiğinde, zafiyeti tetikleyecek payload listesi hazırlanıp, zafiyetin tespit edilmesi durumunda meydana gelecek olayları (sayfa üzerinde hata, gecikme, ulaşılamama vb.) tanımlamak ve kontrol etmek yeterli olacaktır.

Bu çalışma kapsamında sahip olduğu web uygulama sayısı yönünden, literatürde görülen en büyük web sızma testi laboratuvarı oluşturulmuştur. Söz konusu laboratuvar içerisinde; blog, haber, alışveriş, bankacılık türünden sitelerin yanı sıra, okul, restoran, otel, hostel, kütüphane, uçak, otobüs gibi otomasyon uygulamaları da bulunmaktadır. Hazırlanan web sızma testi laboratuvarı içerisinde 207 PHP, 55 ASP tabanlı web uygulaması bulunmaktadır. Github, Gitlab ve Sourceforge üzerinden açık kaynak kodlu olarak paylaşılmış web projeleri kullanılarak oluşturulan laboratuvar sayesinde, gerçek hayatta karşılaşılması en muhtemel web uygulamalarının test edilmesi imkânı elde edilmiştir.

Bu tez kapsamında geliştirilen web zafiyet tarayıcısı kullanılarak, oluşturulan web test laboratuvarı içerisindeki her bir uygulama test edilmiştir. Gerçekleştirilen testler sonrasında elde edilen zafiyet verileri kullanılarak bir veri seti elde edilmiştir.

Literatürde HTML tagları aracılığı ile sayfa sınıflandırma işlemlerinde kullanılan veri setleri, söz konusu taglar içerisindeki text verilerinin parçalanmasıyla oluşan öznitelikler içermektedir. Fakat bu tez kapsamında oluşan veri seti içerisinde, HTML tagları sayısal olarak ele alınmıştır. Söz konusu veri seti içerisinde; çeşitli HTML taglarının sayfada ve zafiyetin tespit edildiği form üzerinde kaç adet bulunduğu bilgileri yer almaktadır. İlgili veri seti içerisinde, zafiyetin bulunduğu sayfa, form ve inputa ait 70 farklı öznitelik bulunmaktadır. Bu çalışma kapsamında oluşturulan veri seti, içerdiği öznitelikler bakımından literatürdeki tek veri seti olma özelliği taşımaktadır. Bu sayede, söz konusu veri seti kullanılarak gerçekleştirilen sınıflandırma ve birliktelik analizi işlemlerinin özgün olma özelliği taşımaktadır.

Bu tez kapsamında elde edilen veri seti kullanılarak yapılan ilk işlem sayfa sınıflandırma işlemidir. Bu işlem için ilk olarak web sayfaları gruplara ayrılmıştır. Veri seti için toplanan örneklerdeki sayfalar, login page, register page, submit page, search page, info page, list page, forgot password page ve contact us page olmak üzere sekiz farklı türde etiketlenmiştir. Veri seti içerisindeki örneklerin etiketlenmesinden sonra, sayfa sınıflandırma işlemi gerçekleştirilmiştir. Sayfa sınıflandırma işleminde beş farklı sınıflandırma algoritması kullanılmıştır. Gerçekleştirilen sınıflandırma işlemleri içerisinde en yüksek doğruluk oranı; %95.39 başarı oranı ile Rastgele Orman algoritması ile elde edilmiştir.

Litetatürde yer alan sayfa sınıflandırma işlemleri incelendiğinde, Woogue, 2017 yılında gerçekleştirdiği çalışmasında, web sayfalarını sanat, teknoloji, mühendislik, sağlık gibi sekiz farklı kategoride sınıflandırmıştır. Söz konusu çalışmada, 7 farklı sınıflandırma algoritması test edilmiş, en yüksek %92 doğruluk oranı yakalanmıştır. Sayfa sınıflandırma konusunda gerçekleştirilen bir diğer çalışma Deri ve arkadaşları tarafından 2015 yılında gerçekleştirilmiştir. Söz konusu çalışmada HTML tagları içerisindeki kelime ve kelime kümeleri kullanılarak web sayfaları sınfınlandırılmış ve ortalama %80 doğruluk oranı yakalanmıştır. Klassen tarafından 2012 yılında yapılan çalışmada, HTML tagları içerisindeki text verileri kullanılarak, arama formları sınıflandırılmıştır. Yapay sinir ağları kullanılarak gerçekleştirilen sınıflandırma işleminde %91 başarı oranı elde edilmiştir. Awad, 2012 yılında gerçekleştirdiği çalışmada, bir haber sitesinden topladığı sayfalardan oluşturduğu veri seti üzerinde SVM, KNN ve GİS algoritmalarını kullanarak sınıflandırma işlemi gerçekleştirmiştir.

Söz konusu çalışmada en yüksek başarı oranı %84 olarak elde edilmiştir. Literatürdeki HTML tagları üzerinden sayfa sınıflandırma çalışmalarında, web sayfaları; indeksleme, reklam gibi amaçlar doğrultusunda temasal olarak yani; spor, haber, teknoloji gibi kategoriler altında sınıflandırılmıştır. Ayrıca söz konusu çalışmalarda, belirlenen HTML taglarının içeriği yani text verisi kullanılmıştır. HTML input alanlarının sayfa üzerinde bulunma miktarları kullanılarak gerçekleştirilen tek sınıflandırma işlemi bu tez çalışmasında gerçekleştirilmiştir. Elde edilen başarı oranı da literatüre özgün ve başarılı bir çalışma katılmasına olanak sağlamıştır.

Bu tez çalışmasında geliştirilen web zafiyet tarayıcısının bir diğer özelliği, test öncesinde web uygulamasının gezilerek, zafiyet testi yapılacak URL adreslerinin kaydedilmesi esnasında, sınıflandırıcı kullanmasıdır. Geliştirilen araç bu sayede, web uygulamasının kullanıcı giriş sayfalarını kendisi tanıyarak, başarılı bir şekilde giriş yapabilmektedir. Günümüzde web testlerinde yayın olarak kullanılan, ticari ve açık kaynak kodlu web zafiyet tarayıcıların hiçbiri bu özelliğe sahip değildir. Açık kaynak kodlu WASCAN ve WAPITI araçları kullanıcı girişi yapabilmek için geçerli bir cookie verisine ihtiyaç duymaktadırlar. Taranacak web sayfasına daha önce kullanıcı girişi yapılmalı, giriş sonrasında oluşan cookie verisi bir şekilde elde edilerek söz konusu araçlara parametre olarak verilmelidir. Bu, kullanıcılar için gerçekleştirmesi zor, zaman alıcı ve sürdürülebilir bir işlem değildir. Siber güvenlik dünyasında ticari web zafiyet tarayıcılar içerisinde başı çeken iki web zafiyet tarayıcısı Netsparker ve Acunetix' dir. Netsparker aracı, bir web uygulamasına kullanıcı girişi yaparak, doğrulama arkasındaki sayfaları tarayabilmek için; kullanıcıdan login sayfasının URL adresini, kullanıcı adını ve parola bilgisini istemektedir. Bu işlem programın çalıştırılması esnasında girilmesi gereken ek bir yük oluşturmaktadır. Acunetix aracı ise, tarama öncesinde, kendi bünyesinde açtığı bir tarayıcı üzerinden kullanıcının taranacak web uygulamasında giriş işlemi yapmasını istemektedir. Böylelikle kullanıcı tarafından hangi adrese girildi, hangi inputlara kullanıcı adı ve parola yazıldı bilgisini kaydederek, daha sonra tekrarlamaktadır. Bu çalışma kapsamında geliştirilen web zafiyet tarayıcısı ise, kullanıcı girişi yapabilmek için sadece geçerli bir kullanıcı adı ve parola bilgisine ihtiyaç duymaktadır. Geliştirilen araç çalışma anında gezdiği sayfaları tek tek sınıflandırarak, login sayfası olup olmadığını kontrol etmektedir. Sınıflandırma işleminde login sayfası tespit edildiğinde, sayfa üzerinde yer alan inputlar otomatik olarak kullanıcı adı ve parola bilgileri ile doldurulmakta ve başarılı bir şekilde giriş

işlemi gerçekleştirilmektedir. Bu özellik web uygulama zafiyet tarayıcılar alanında literatüre kazandırılmış yeni bir işlev özelliği taşımaktadır.

Bu tez çalışmasında ayrıca; web uygulama güvenlik testlerinde kullanılmak üzere, sayfa sınıflandırma ve birliktelik analizi işlemlerinin birleştirilerek üretilen, yeni bir test modeli geliştirilmiştir. Geliştirilen modelde yapılan birliktelik analizi işleminde; "bir zafiyetin tespit edildiği bir sayfada, hangi türden zafiyetin görülme olasılığı daha yüksektir?" sorusuna yanıt aranmıştır. Söz konusu birliktelik analizi, veri setindeki sadece POST metodu üzerinden bulunan zafiyet örneklerinde incelenmiştir. Bunun nedeni GET metodu üzerinden gerçekleştirilen testlerde bulunan zafiyetlerin sayfa türü ile bir ilişkisinin bulunamamasıdır. Zafiyet türleri ile sayfa türleri arasında bir ilişki bulabilmek için POST üzerinden tespit edilen zafiyet örneklerinin incelenmesi gerekmektedir. Gerçekleştirilen birliktelik analizi sonrasında web zafiyet tarayıcısı, bir zafiyet tespit ettiğinde, o zafiyet ile ilişkili zafiyete ait test sınıfını çalıştıracak şekilde güncellenmiştir.

Birliktelik analizi sonunda elde edilen sonuçlar, güvenlik uzmanlarından alınan görüşler (EK-1) ile karşılaştırılıp, geçerliliği doğrulanmıştır. Örneğin server side includes (SSTI) zafiyeti için elde edilen sonuçlarda, SSTI görülen yerlerde en çok görülme olasılığına sahip zafiyet Command Execution (komut çalıştırma) olarak tespit edilmiştir. Manuel olarak yapılan testlerde sunucu taraflı SSTI zafiyeti tespit edildiğinde, söz konusu zafiyet üzerinden yaygın olarak komut çalıştırma saldırıları gerçekleştirilmektedir. HTML enjeksiyonu zafiyeti için elde edilen birliktelik kuralında; HTML enjeksiyonu görülen yerlerde en çok görülme olasılığına sahip zafiyet XSS olarak belirlenmiştir. Web uygulama testlerinde uzmanlar bir HTML enjeksiyonu zafiyet ile karşılaştıklarında ilk test edeceği diğer zafiyet türü XSS olmaktadır. Bunun nedeni, HTML enjeksiyonu ve XSS zafiyetlerinin istismar edilmesinde kullanılan paylodların HTML taglarından oluşmasıdır. Son olarak directory traversal zafiyeti görülen bir sayfada, görülme olasığı en yüksek zafiyet olarak local file injection tespit edilmiştir. Örnek olarak PHP include, include once fonksiyonlarının her iki zafiyeti de tetiklediği düşünülürse, elde edilen birliktelik kararı doğrudur.

Geliştirilen modele göre test edilecek web sayfası ilk olarak sınıfınlandırılmakta; sınıflandırma sonrasında tespit edilen sayfa türüne göre, veri setine dayanarak, en çok görülmüş zafiyetin test sınıfı çalıştırılmaktadır. Test işlemleri esnasında, eğer bir zafiyet tespit edilirse; birliktelik analizi ile elde edilen birliktelik kurallarına göre, o zafiyetin görüldüğü sayfalarda görülme olasılığı en yüksek olan diğer zafiyet test sınıfları çalıştırılmaktadır. Bu sayede gerçekleştirilen testler gerçek bir sızma testi uzmanı tecrübesi ve mantığına dayanır şekilde devam etmektedir. Geliştirilen model sayesinde tespit edilen zafiyetler ile ilişkisi bulunmayan zafiyetlere ait test sınıfları çalıştırılmayacak, böylelikle gereksiz testler sonunda sayfa yapısının veya veri tabanı tablolarının bozulmasının önüne geçilecektir. Geliştirilen model, tüm test sınıflarının ilişki gözetmeksizin sırayla çalıştırıldığı klasik modele göre daha hızlı sonuç vermektedir. Şekil 5.1' de geliştirilen zafiyet tarayıcısının standart tarama yapan versiyonu ile sayfa sınıflandırma- birliktelik analizi modeli kullanarak tarama yapan versiyonu kullanılarak test işlemi gerçekleştirilmiştir. Test işlemi; içerisinde çeşitli güvenlik açıklarını barındıran ve web test işlemleri için oluşturulmuş online laboratuvar; "http://testphp.vulnweb.com üzerinde gerçekleştirilmiştir. Aynı hedef üzerinde, aynı test sınıfları kullanılarak, aynı internet bağlantısı ile gerçekleştirilen bu test işlemleri, geliştirilen sayfa sınıflandırma- birliktelik analizi tabanlı modelin, klasik tarama yöntemi ile karşılaştırılmasına olanak sağlamıştır. Şekil 5.1 (a)' da görüldüğü üzere klasik yöntemle yapılan tarama işlemi 2428 saniyede yani yaklaşık 40 dakikada tamamlanmıştır. Şekil 5.2 (b) de görüldüğü üzere, geliştirilen model kullanılarak yapılan tarama işlemi 1912 saniye yani yaklaşık 31 dakikada tamamlanmıştır. Gerçekleştirilen her iki tarama işlemi sonrasında, 7 kör SQL enjeksiyonu, 2 hata tabanlı SQL enjeksiyonu, 4 komut çalıştırma, 4 HTML enjeksiyonu, 4 yerel dosya dahil etme (LFI), 4 PHP kod enjeksiyonu ve 4 siteler arası betik çalıştırma (XSS) olmak üzere toplam 29 zafiyet tespit edilmiştir. Elde edilen sonuçlar incelendiğinde; geliştirilen modelin standart tarama modeline ile aynı etkide ama daha hızlı çalıştığı tespit edilmiştir. Elde edilen sonuçlarda, geliştirilen yeni modelin, standart tarama modeline göre %21 hız kazancı sağladığı görülmektedir.

(a)

(b)

Şekil 5.1. Geliştirilen model ile klasik yöntemin karşılaştırılması

Bu çalışmada oluşturulan sayfa sınıflandırma- birliktelik analizi tabanlı test modeli, dinamik analiz tabanlı web zafiyet testlerinde, test sınıflarının belirli kurallar çerçevesinde çalıştırılmasını amaçlayan tek çalışmadır. Günümüzde kullanılan zafiyet tarayıcılar, test edilecek input alanlarını tespit ettikten sonra sırası ile test sınıflarını çalıştırmaktadır. Birkaç zafiyet tarayıcısı (örn: WAPITI) bazı test sınıfları çalıştırılmadan önce, başka bir sınıfın çalışması koşulunu koymuşlardır (örn: BlindSQLi testi öncesinde SQLi testi yapılması). Fakat tüm test işlemlerinin modellenmesi amacı ile gerçekleştirilen bir çalışmaya literatürde rastlanmamıştır.

Geliştirilen araç şu ana kadar WAPITI, WASCAN, Netsparket ve Acunetix araçları ile karşılaştırılmıştır. Bunun nedeni söz konusu araçların, hem günümüzde web uygulama testlerinde en yaygın kullanılan araçlar olması, hem de yaygın kullanımları nedeniyle sürekli güncellenerek geliştirilmeye devam etmeleridir. Bu çalışmada geliştirilen araç, literatürde web zafiyet tarayıcılar alanında gerçekleştirilen diğer çalışmalar ile karşılaştırıldığında da çeşitli yönlerden üstünlük göstermektedir. Örneğin Secubat aracı (Kals vd., 2006), SQLi testlerinde, testin başarılı olup olmadığını kontrol etmek amacıyla, test sonrası dönen sayfa yanıtını inceleyerek, çeşitli yanıt kalıpları ve anahtar kelimeler kullanarak ve bir güven değeri oluşturmaktadır. Bu çalışma kapsamında geliştirilen SQL enjeksiyonu test sınıfı ise; Oracle, mySQL, msSQL gibi 14 farklı

tabanı türde veri tabanı sunucusu için tanımlanmış geniş bir hata kontrol listesi içermektedir. Literatürdeki, geliştirilmiş bir diğer web zafiyet tarayıcısı Sania' dır (Kosuka vd., 2007). Söz konusu araç, web uygulaması ile veri tabanı sunucusu arasında bir Proxy sunucu gibi çalışarak, uygulamadan sunucuya gönderilen SQL sorgularını yakalamaktadır. Söz konusu araç, yakaldığı sorgular içerisine SQL enjeksiyonu test payloadlarını ekleyerek zafiyet testini içermektedir. Bu tez çalışması kapsamında geliştirilen zafiyet tarayıcısı ise, araya girerek sorguları yakalamak yerine, HTML kaynak kodlarından, sayfadaki form alanlarını inceleyerek, içerisinde test payloadları içeren zararlı sorguları dinamik olarak oluşturmaktadır. Literatürde yer alan diğer bir web zafiyet tarayıcısı SDAPT' tır (Halfond vd. 2009). Söz konusu zafiyet tarayıcısı web sayfası üzerindeki, potansiyel SQL enjeksiyonu saldırı noktalarını tespit etmek için statik analiz yöntemini kullanmaktadır. Statik analiz yöntemi ile elde edilen saldırı noktaları, daha sonra dinamik analiz yöntemi ile teste tabi tutulmaktadır. Söz konusu araç hem tek zafiyet testi yapması hem de test yapabilmek için test edilecek uygulamanın kaynak kodlarına ihtiyaç duyması nedeni ile, bu çalışmada geliştirilen aracın gerisinde kalmaktadır. Yukarıda değinilen çalışmalarda da olduğu gibi; literatürde web uygulama zafiyet tarayıcısı başlığı altında incelenen çalışmalar, çoğunlukla spesifik birkaç zafiyetin test edilmesini amaçlamışlardır. Artunes ve Viera (2009) çalışmalarında, web servisleri üzerinde SQL enjeskiyonu testi gerçekleştiren bir araç geliştirmişlerdir. Chen ve Wu, 2010 yılında gerçekleştirmiş oldukları çalışmalarında, SQL enjeksiyonu ve XSS zafiyetlerinin tespiti için bir web uygulama zafiyet tarayıcısı geliştirmişlerdir. Galan vd. (2010) çalışmalarında, web uygulamaları üzerinde yer alan XSS zafiyetlerinin tespit edilebilmesi için bir araç geliştirmişlerdir. Ali vd. (2011) çalışmalarında, MySQLlInjector adında, PHP dili ile geliştirilmiş web uygulamaları üzerinde SQL enjeksiyon zafiyeti testi yapan bir araç geliştirmiştir. Singh ve Roy (2012) çalışmalarında; NVS isminde, web uygulamaları üzerinde yer alan SQL enjeksiyonu zafiyetlerini testpit eden bir araç geliştirmişlerdir. Djuric tarafından 2013 yılında gerçekleştirilen çalışmada; SQL enjeksiyonu zafiyetinin tespiti için SQLIVDT adında bir web uygulama zafiyet tarayıcısı geliştirilmiştir. Qianqian ve Xiangjun (2014) çalışmalarında, komut ekranından çalışmakta olan mevcut açık kaynak kodlu bir web zafiyet tarayıcısına bir arayüz eklemiş ve servis olarak yayınlamışlardır. İncelenen çalışmalar ile bu tez kapsamında geliştirilen araç karşılaştırıldığında, sansar zafiyet

tarayıcısının, söz konusu çalışamalara göre daha fazla zafiyeti test edebildiği ve test işlemlerinde yapay zekâ tabanlı model kullanan tek araç olduğu görülmektedir.

Yapılan tüm karşılaştırmalar özetlendiğinde, bu tez kapsamında geliştirilen zafiyet tarayıcısının, günümüzde yaygın olarak kullanılan açık kaynak kodlu zafiyet tarayıcılardan daha fazla test sınıfına sahip olduğu görülmektedir. Söz konusu araç, OWASP en kritik 10 zafiyet listesindeki (OWASP, 2017) zafiyetlerde dahil olmak üzere, girdi doğrulama açıklarından kaynaklı birçok zafiyeti test edebilmektedir. Ayrıca web uygulamasını indeksleme işleminde sınıflandırma modeli kullanan tek araç olma özelliği taşımaktadır. Bunun yanında, web uygulama testlerinde kullanılmak üzere sayfa sınıflandırma- birliktelik analizi tabanlı bir model oluşturulmuş, oluşturulan modelin klasik tarama yöntemine göre daha hızlı ve daha güvenilir testler gerçekleştirdiği görülmüştür.

Bu tez çalışması kapsamında geliştirilen web zafiyet tarayıcısının aynı zamanda ulusal siber güvenlik alanında önemli bir yer alacağı düşünülmektedir. Geliştirilen web zafiyet tarayıcısının, güvenlik uzmanları tarafından gerçekleştirilecek web güvenlik testlerinde aktif olarak kullanılması amaçlanmaktadır.

KAYNAKLAR

Acunetix, 2019, Directory Traversal Attacks. Erişim Tarihi: 15.10.2019. https://www.acunetix.com/websitesecurity/directory-traversal/

Agrawal, S., ve Gupta, R. D., 2014. Development and comparison of open source based Web GIS Frameworks on WAMP and Apache Tomcat Web Servers. The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, 40(4), 1.

Akar, Ö., ve Güngör, O., 2012. Rastgele Orman Algoritması Kullanılarak Çok Bantlı Görüntülerin Sınıflandırılması. Jeodezi ve Jeoinformasyon Dergisi, 1(2), 139-146.

Akcay, A., 2019, File Inclusion Nedir?, Erişim Tarihi: 15.10.2019. https://medium.com/@akcayca90/file-inclusion-nedir-da2f2218383f

Akgün, F., Buluş, E., ve Buluş, H. N., 2005. Yazılım Mühendisliği Açısından Uygulamalardaki Ara Bellek Taşması Zafiyetinin İncelenmesi, Elektrik-Elektronik-Bilgisayar Mühendisliği 11. Ulusal Kongresi ve Fuarı, İstanbul, Türkiye.

Ali, A. B. M., Abdullah, M. S., and Alostad, J., 2011. SQL-İnjection Vulnerability Scanning Tool For Automatic Creation Of SQL-İnjection Attacks. Procedia Computer Science, 3, 453-458.

Andreu A., 2006, Professional Pen Testing For Web Applications. John Wiley and Sons, 9-10.

Antunes, N., and Vieira, M. 2009. Detecting SQL İnjection Vulnerabilities in Web Services. In 2009 Fourth Latin-American Symposium on Dependable Computing, 1-4 September,Joao Pessoa, Brazil, 17-24.

Auger, R. 2009, Information Leakage, Erişim Tarihi: 18.10.2019. http://projects.webappsec.org/w/page/13246936/Information%20Leakage

Awad, W.A. 2012. Machine Learning Algorithms in Web Page Classfıcation. International Journal of Computer Science & Information Technology (IJCSIT), 4, 5.

Başeğmez, M., 2016, HTTP – Request Methods, Erişim Tarihi:19.10.2019. http://mehmetbasegmez.com.tr/frontend/http-request-methods/

Bau, J., Bursztein, E., Gupta, D., and Mitchell, J., 2010. State of The Art: Automated Black-Box Web Application Vulnerability Testing. In 2010 IEEE Symposium on Security and Privacy, 22-25 May, Berkeley, 332-345.

Burlu, K., 2010. Bilişimin Karanlık Yüzü, Nirvana Yayınları, 479s, Ankara.

Canbek, G., ve SAĞIROĞLU, Ş., 2006. Bilgi, Bilgi Güvenliği ve Süreçleri Üzerine Bir İnceleme. Politeknik Dergisi, 9(3), 165-174.

Çelik, E., 2011. Derinlemesine XSS (DOM, Stored, Reflected) Saldırısı. Erişim Tarihi: 25.09.2019. http://eyupcelik.com.tr/guvenlik/403-derinlemesine-xss-dom-stored-reflected-saldirisi

Çetin, H., Gundak, İ., ve Çetin, H. H., 2015. E-işletme Güvenliği ve Siber Saldırılar Üzerine Bir Araştırma. Çankırı Karatekin Üniversitesi Sosyal Bilimler Enstitüsü Dergisi, 6(2), 223-240.

Chen, J. M., and Wu, C. L. 2010. An Automated Vulnerability Scanner For İnjection Attack Based On İnjection Point. International Computer Symposium (ICS2010) 16-18 December. Tainan, Taiwan, 113-118.

Çıtak, Ö., 2016, Dom Based XSS' i Anlamak. Erişim Tarihi: 15.09.2019 https://omercitak.com/dom-based-xssi-anlamak/

Deepa, G., & Thilagam, P. S., 2016. Securing Web Applications From İnjection And Logic Vulnerabilities: Approaches And Challenges. Information and Software Technology, 74, 160-180.

De Jimenez, R. E. L., 2016. Pentesting On Web Applications Using Ethical-Hacking. IEEE 36th Central American and Panama Convention, 9-11 November, San Jose, 1-6.

Deri, L., Martinelli, M., Sartiano, D., Sideri, L. 2015. Large Scale Web-Content Classification. Expo 2015, Milano, Italy.

Djuric, Z., 2013. A Black-Box Testing Tool For Detecting SQL İnjection Vulnerabilities. Second International Conference on Informatics & Applications (ICIA), 23-25 September, Lodz, Poland, 216-221.

Fonseca, J., Seixas, N., Vieira, M., and Madeira, H., 2013. Analysis Of Field Data On Web Security Vulnerabilities. IEEE Transactions On Dependable And Secure Computing, 11(2), 89-100.

Galán, E., Alcaide, A., Orfila, A., and Blasco, J., 2010. A Multi-Agent Scanner To Detect Stored-XSS Vulnerabilities. In 2010 International Conference for Internet Technology and Secured Transactions. 8-11 November, London, UK, 1-6.

Gislason P.O., Benediktsson J.A, Sveinsson J.R., 2004, Random Forest Classification Of Multi-Source Remote Sensing And Geographic Data, IEEE International Geoscienceand Remote Sensing Symposium IGARSS '04Proceedings, 2,1049 – 1052.

Golub, K., Ardö, A. 2005. Importance of HTML Structural Elements and Metadata in Automated Subject Classification. A In International Conference on Theory and Practice of Digital Libraries, 18-23 September, Berlin, 368-378.

Halfond, W. G., Choudhary, S. R., ve Orso, A. 2009. Penetration Testing With İmproved İnput Vector İdentification. In 2009 International Conference on Software Testing Verification and Validation, 1-4 April, Denver, USA. 346-355

Hudak,P., 2017. The Art of Subdomain Enumeration. Erişim Tarihi: 20.10.2019. https://blog.sweepatic.com/art-of-subdomain-enumeration/

ITRC, 2018. Identity ITRC 2018 End-of-Year Data Breach Report. Erişim Tarihi: 21.09.2019. https://www.idtheftcenter.org/2018-end-of-year-data-breach-report/

Jimenez, R.E.L. 2016. Pentesting On Web Applications Using Ethical Hacking. IEEE 36th Central American and Panama Convention, 9-11 November, San Jose, 1-6.

Kals, S., Kirda, E., Kruegel, C., ve Jovanovic, N. 2006. Secubat: A Web Vulnerability Scanner. In Proceedings of the 15th İnternational Conference On World Wide Web. 23-26 May, Edinburgh, Scotland, 247-256.

Karro, J., & Wang, J. 1998. Protecting Web Servers From Security Holes İn Server-Side İncludes. In Proceedings 14th Annual Computer Security Applications Conference,7-11 December, Arizona, 103-111.

Kettle, J., 2019. Server Side Template Injection. Erişim Tarihi: 19.10.2019. https://portswigger.net/research/server-side-template-injection

Khalid, S., Abbas, H., Pasha, M., & Raza, A. 2012. Securing Internet Information Services (IIS) configuration files. In 2012 International Conference for Internet Technology and Secured Transactions,10-12 December, London, 726-729.

Kieyzun, A., Guo, P. J., Jayaraman, K., Ernst, M. D., 2009. Automatic Creation Of SQL Injection And Cross-site Scripting Attacks. 31st International Conference On Software Engineering, 16-24 May, Vancouver, Canada, 199-209.

Klassen, M., 2012. A Frame Work For Search Forms Classification. 2012. International Conference on Systems, Man, and Cybernetics, 14-17 October, Seoul, 1029-1034.

Korkem, E. 2013. Mikroarray Gen Ekspresyon Veri Setlerinde Random Forest Ve Naıve Bayes Sınıflama Yöntemleri Yaklaşım. Hacettepe Üniversitesi, Sağlık Bilimleri Enstitüsü, Yüksek Lisans Tezi, 52, Ankara.

Kosuga, Y., Kono, K., Hanaoka, M., Hishiyama, M., ve Takahama, Y. 2007. Sania: Syntactic and semantic analysis for automated testing against sql injection. In Twenty-Third Annual Computer Security Applications Conference. 10-14 December, Miami, USA. 107-117.

Krutil, J., Kudelka, M., Snasel, V. 2012. Web Page Classification Based On Schema.org Collection. Fourth International Conference on Computational Aspects of Social Networks. 21-23 November, Sao Carlos, 356-360.

Lee, I., Jeong, S., Yeo, S., and Moon, J., 2012. A Novel Method For SQL İnjection Attack Detection Based On Removing SQL Query Attribute Values. Mathematical And Computer Modelling, 55(1-2), 58-68.

Lee, J.-H., Yehb,W.-C., Chuanga, M.-C. 2015. Web Page Classification Based On A Simplified Swarm Optimization. Applied Mathematics and Computation, 270, 13–24.

Liaw A.,Wiener M., 2002. Classification And Regression by Random Forest, R News, 2(3). 18-22.

Makino, Y., ve Klyuev, V., 2015. Evaluation Of Web Vulnerability Scanners. 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications,24-26 September, Warsaw, 399-402.

Mary, C., 2015. Shellshock Attack On Linux Systems–Bash. International Research Journal of Engineering and Technology, 2(8), 1322-1325.

Michael, C, C. 2005. Risk-Based and Functional Security Testing. Erişim Tarihi: 16.09.2019. https://buildsecurityin.us-cert.gov/articles/bestpractices/security-testing/risk-base-and-functional-security-testing.

Mirdula, S., Manivannan, D. 2013. Security Vulnerabilities in Web Application-An Attack Perspective. International Journal of Engineering and Technology. 5, 2.

Mohammad, S., Pourdavar, S., 2010. Penetration test: A Case Study On Remote Command Execution Security Hole. Fifth International Conference On Digital Information Management, 5-8 July, Thunder Bay, Canada, 412416.

Morgenroth,S.,2017. CRLF Injection ve HTTP Response Splitting Zafiyeti, Erişim Tarihi: 22.10.2019. https://www.netsparker.com.tr/blog/web-guvenligi/crlf-injection-ve-http-response-splitting-zafiyeti/

OWASP, 2013. Code Injection, Erişim Tarihi: 25.10.2019. https://www.owasp.org/index.php/Code_Injection

OWASP, 2017. “The Ten Most Critical Web Application Security Risks”, Erişim Tarihi: 05.11.2019. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

OWASP, 2018. “Command Injection”, Erişim Tarihi: 26.10.2019. https://www.owasp.org/index.php/Command_Injection

Özel, A., Eken, S. 2014. Javacsript Güvenlik Açıkları ve Güncel Çözüm Önerileri. 7. Uluslar Arası Bilgi Güvenliği ve Kriptoloji Konferansı, 17-18 Ekim, İstanbul, 252-257.

Pietraszek, T., Berghe, C.V. 2005. Defending Against Injection Attacks Through Context-Sensitive String Evaluation, Recent Advances in Intrusion Detection, 7-9 September, Seattle, 124-145.

Qianqian, W., and Xiangjun, L., 2014. Research And Design On Web Application Vulnerability Scanning Service. In 5th International Conference on Software Engineering and Service Science, 27-29 June, Beijing, 671-674.

Resmî Gazete, 2007. Bankalarda Bilgi Sistemleri Yönetiminde Esas Alınacak İlkelere İlişkin Tebliğde Değişiklik Yapılmasına Dair Tebliğ (Tarih: 14.09.2007, Sayı: 26643). Erişim Tarihi: 17.08.2019. http://www.resmigazete.gov.tr/eskiler/2007/09/20070914-1.html

Ru, Y., Horowitz, E. 2007. Automated Classification Of HTML Forms On E-Commerce Web Sites. Online Information Review, 31(4), 451-466.

Sarıman G., 2015. Yazılım Güvenliği Test Ve Değerlendirme Aracı Geliştirilmesi. Süleyman Demirel Üniversitesi, Fen Bilimleri Enstitüsü, Doktora Tezi, 141s, Isparta

Sedek, K. A., Osman, N., Osman, M. N., ve Kamaruzaman, J. H., 2009. Developing a Secure Web Application Using OWASP Guidelines. Computer and Information Science, 2(4), 137-143.

Shar, L. K., and Tan, H. B. K., 2013. Predicting Sql Injection And Cross Site Scripting Vulnerabilities Through Mining İnput Sanitization Patterns. Information and Software Technology, 55(10), 1767-1780.

Singh, A. K., and Roy, S. 2012. A Network Based Vulnerability Scanner For Detecting Sqli Attacks İn Web Applications. 1st International Conference on Recent Advances in Information Technology (RAIT), 15-17 March, Dhanbad, India, 585-590.

Şengün, G., 2018. Http Cookie Nedir, Hangi Amaçlarla Ve Nasıl Kullanılır?. Erişim Tarihi: 29.10.2019. https://medium.com/@gokhansengun/http-cookie-nedir-hangi-amaçlarla-ve-nasıl-kullanılır-5e9b9141fb7b

Thomas, S., & Williams, L. 2007., Using Automated Fix Generation To Secure Sql Statements. In Proceedings of the Third International Workshop on Software Engineering for Secure Systems, 20-26 May, Washington, 9.

Trustwave, 2019, Understanding and Discovering Open Redirect Vulnerabilities. Erişim Tarihi: 10.11.2019. https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/understanding-and-discovering-open-redirect-vulnerabilities/

Tsipenyuk, K., Chess, B., & McGraw, G., 2005. Seven Pernicious Kingdoms: A Taxonomy Of Software Security Errors. IEEE Security & Privacy, 3(6), 81-84.

Verizon Enterprise, 2019. Data Breach Investigations Report, Technical Report. Erişim Tarihi: 20.11.2019. https://enterprise.verizon.com/resources/reports/2019-data-breach-investigations-report.pdf

Vural, Y., 2007. Kurumsal Bilgi Güvenliği ve Sızma (Penetrasyon) Testleri. Gazi Üniversitesi, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 269s, Ankara.

Wang, J., Cai, H., Xu, B., Jiang, L.2008. CUCS: A Web Page Classification Algorithm for Large Training Set. IFIP International Conference on Network and Parallel Computing, 23-24 August, Mongolia.

Weinberger, J., Barth, A., & Song, D. 2011, August. Towards Client-side HTML Security Policies. 6th USENIX Workshop on Hot Topics in Security, 9 August, San Francisco.

Woogue, P.D.P., Pineda, G.A.A., Christian V. Maderazo, C.V. 2017. Automatic Web Page Categorization Using Machine Learning and Educational-Based Corpus. International Journal of Computer Theory and Engineering, 9, 6.

Xiong, P., Peyton, L., 2010. A Model-Driven Penetration Test Framework For Web Applications. Eighth Annual International Conference on Privacy, Security and Trust, 17- 19 August, Ottawa, Canada, 173-180.

Yalçınkaya, M.A., 2015. Gelişmiş Hedef Odaklı Siber Saldırılar, Süleyman Demirel Üniversitesi, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 186s, Isparta.

Yavuz, N., 2019, ShellShock (Bash Bug) zafiyeti nedir?, https://medium.com/@nuriyavuz2.71/shellshock-bash-bug-zafiyeti-nedir-8b5d20f71a0d

Yılmaz, H., 2014. Random Forest Yönteminde Kayıp Veri Probleminin İncelenmesi ve Sağlık Alanında Bir Uygulama. Eskişehir Osman Gazi Üniversitesi, Sağlık Bilimleri Enstitüsü, Yüksek Lisans Tezi, 78s, Eskişehir.

EKLER

EK A. Uzman Görüşü Formları

EK A. Uzman Görüşü Formları

Çizelge Ek A.1. Bir numaralı uzman görüşü (Innovera Bilişim Teknolojileri A.Ş., Güvenlik Proje Lideri / Kıdemli Sızma Testi Uzmanı, Mesleki Tecrübe: 3 Yıl)

Soru: Bir web uygulamasında hangi zafiyetlerin bir arada görülmesini beklersiniz?
Yaptığımız testlerde, kullanıcı giriş sayfalarında çoğunlukla SQL enjeksiyonu ve yansıtılmış XSS zafiyeti ile karşılaşıyoruz. Kullanıcı kayıt (register) sayfalarında yine SQL enjeksiyonu ve kalıcı XSS zafiyeti ile karşılaşıyoruz. Kod Enjeksiyonu (code injection), işletim sistemi komut enjeksiyonu (command execution) ve yerel dosya dahil etme (local file injection) zafiyetleri çoğunlukla 'Contact Us Page', 'Forgot Password Page' sayfalarında birlikte görülme olasılığı olan zafiyetler olarak tanımlanabilir. Bunun yanında HTML enjeksiyonu zafiyeti ile karşılaşılan sayfalarda çoğunlukla XSS zafiyeti ile de karşılaşıyoruz.

Çizelge Ek A.2. İki numaralı uzman görüşü (KoçSistem Bilgi ve İletişim Hizmetleri A.Ş.- Sızma Testi Uzmanı, Mesleki Tecrübe: 4 Yıl)

Soru: Bir web uygulamasında hangi zafiyetlerin bir arada görülmesini beklersiniz?
Gerçekleştirdiğimiz sızma testlerinden elde ettiğimiz tecrübelerimize göre; kod enjeksiyonu, işletim sistemi komut enjeksiyonu ve yerel dosya dahil etme zafiyetlerinin benzer noktalarda ortaya çıktığını söyleyebiliriz. Bunun yanında SQL enjeksiyonu görülen noktalarda yüksek ihtimalle XSS zafiyeti ile karşılaşmayı beklediğimizi söyleyebiliriz. Sunucu taraflı template enjeksiyonu (server side template injection) zafiyeti ile karşılaşılan noktalarda ise işletim sistemi komut enjeksiyonu zafiyetinin de çıkmasını bekliyoruz. Yerel dosya dahil etme zafiyeti ile karşılaşılan sayfalarda dizin gezinimi (directory traversal) zafiyetinin de ortaya çıkma ihtimali bulunmaktadır.

Çizelge Ek A.3. Üç numaralı uzman görüşü (Kuveyt Türk Katılım Bankası A.Ş. – Siber Güvenlik Mühendisi, Mesleki Tecrübe: 4 Yıl)

Soru: Bir web uygulamasında hangi zafiyetlerin bir arada görülmesini beklersiniz?
SQL enjeksiyonu ile karşılaştığımız noktalarda çoğunlukla XSS zafiyeti ile de karşılaşıyoruz. HTML enjeksiyonu ve XSS zafiyetinin de benzer noktalarda ortaya çıkmasını bekliyoruz. Kod enjeksiyonu ile karşılaştığımız sayfalarda işletim sistemi komut enjeksiyonu zafiyetinin de görülme ihtimali bulunmaktadır. Yerel dosya dahil etme zafiyeti ile dizin gezinimi zafiyetlerinin aynı noktalarda görüldüğü durumlar ile de karşılaştık.

ÖZGEÇMİŞ

Adı Soyadı : Mehmet Ali YALÇINKAYA

Doğum Yeri ve Yılı : Isparta, 1990

Medeni Hali : Evli

Yabancı Dili : İngilizce

E-posta : mehmetyalcinkaya@ahievran.edu.tr

Eğitim Durumu

Lise : Isparta Anadolu Lisesi,2004- 2008 (94.01/100)

Lisans : Süleyman Demirel Üniversitesi, Teknik Eğitim Fakültesi, Bilgisayar Sistemleri Öğretmenliği, 2008- 2012 (3,51/4)

Yüksek Lisans : Süleyman Demirel Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı, 2013- 2015 (3.71/4)

Mesleki Deneyim

Süleyman Demirel Üniversitesi Mühendislik Mimarlık Fakültesi Bilgisayar Mühendisliği Bölümü – Araştırma Görevlisi (2013- 2017)

Kırşehir Ahi Evran Üniversitesi Mühendislik Mimarlık Fakültesi Bilgisayar Mühendisliği Bölümü – Araştırma Görevlisi (2017- Halen)

Yayınları

Alanında, Yurtiçinde Yayımlanan Kitap

M.A.Yalçınkaya,E.U.Küçüksille, 2015. "Uygulamalı Sızma Testleri Pentest Lab", ISBN:9786059129251,Abaküs Kitap,160s.

SCI, SSCI ve AHCI Dışındaki İndeks Ve Özler Tarafından Taranan Dergilerde Yayımlanan Teknik Not, Editöre Mektup, Tartışma, Vaka Takdimi Ve Özet Türünden Yayınlar Dışındaki Makale

Ecemiş, A., Küçüksille, E., U., Yalçınkaya, M., A., 2018. Yaygın Görülen Dosya Enjeksiyon Zararlılarının Tespiti ve Sistematik Olarak Analizi, Ömer Halisdemir Üniversitesi Mühendislik Bilimleri Dergisi, Cilt 7, Sayı 2, ss. 478-489

Yalçınkaya, M.A., Altınok, B., Gürdal, M., Akdoğan, M., Küçüksille, E.U., 2017. Zararlı Yazılım Yayma Aracı Olarak Mobil Uygulamaların Kullanılması: Pokemon Go Örneği, Mehmet Akif Ersoy Üniversitesi Fen Bilimleri Enstitüsü Dergisi, Cilt 8, Sayı 1, ss. 88-96

Ganal, S., Yalçınkaya, M.A., Küçüksille, E.U., 2017. "Kurumsal Bilgi Güvenliği Üzerinde Yeni Kayıtlı İnternet Sitelerinin Etkisinin Analiz Edilmesi", Uluslararası Bilgi Güvenliği Mühendisliği Dergisi, Cilt 3, Sayı 2, ss. 42- 47.

Yüksel, A.S., Çankaya, İ.A., Yalçınkaya, M.A., Ateş, N., 2015. Mobil Tabanlı Optik Form Değerlendirme Sistemi, Pamukkale Üniversitesi Mühendislik Bilimleri Dergisi, 22(2), 94-99.

Karabulut, Y.E., Boylu, G., Küçüksille E. U., Yalçınkaya, M.A., 2015, "Characteristics of Cyber Incident Response Teams in the World and Recommendations for Turkey", Balkan Journal Of Electrıcal & Computer Engıneerıng, 3(4), pp. 175-178, 2015

Uluslararası Toplantıda Sunularak Tam Metin Olarak Yayımlanan Bildiri

Yalçınkaya, M.,A., Gürel, A., Aydoğan, T., Küçüksille, E., U., 2019. "Oltalama (Phishing) Saldırılarına Karşı Kurumsal Farkındalık Seviyesinin Belirlenmesi İçin Otomatize Araç Geliştirilmesi ", 8th International Conference On Applied Sciences, 11-13 Ekim, Diyarbakır/Türkiye.

Ganal, S., Yalçınkaya, M.A., Küçüksille, E.U., 2018. "Detection of Proxy Misconfigurations via Log Alarms ", 6th International Symposium on Digital Forensic and Security (ISDFS 2018), 22-25 March, Antalya/ Turkey.

Küçüksille, E.U., Yalçınkaya, M.A., Ganal, S., 2015, "Developing a Penetration Test Methodology in Ensuring Router Security and Testing it in a Virtual Laboratory ", The 8th International Conference on Security of Information and Networks, 8-10 September, Sochi/Russia.

Küçüksille, E.U., Yalçınkaya, M.A., Uçar, O., 2014. "Physical Dangers in the Cyber Security and Precautions to be Taken", The 7th International Conference on Security of Information and Networks, Glasgow/UK.

Küçüksille, E.U., Yalçınkaya, M.A., Uçar, O., "Siber Saldırılarda İstismar Kitlerinin Kullanımı Üzerine Bir Analiz ve Savunma Önerileri",7th International Conference on Information Security and Cryptology, 9-11 September, İstanbul/Türkiye.

Gürel, A., Ertaş, F., Küçüksille, E.U., Yalçınkaya, M.A., "The Development Of Penetration Test Lab For The Web Application Security Training, International Conference on Engineering Technologies ICENTE'17, 07-09 December, Konya/Türkiye.

Uluslararası Toplantıda Poster, Sözlü Sunum İle Gösterimleri

Küçüksille, E.U., Katı, B.E., Yalçınkaya, M.A., 2015. "PoS Sistemlerine Yönelik RAM Kazıma Saldırılarının İstatistiksel Analizi ve Savunma Önerileri ", 8th. International Conference on Information Security and Cryptology (ISC Turkey 2015), 30-31 Ekim, Ankara/Türkiye.

Uluslararası Toplantıda Sunularak Özet Metin Olarak Yayımlanan Bildiri

Yalçınkaya, M.,A., Ganal, S., Küçüksille, E., U., 2018. "Kripto Para Üretme Odaklı Siber Saldırılar ve Alınabilecek Önlemler", II.Internatıonal Scıentıfic And Vocatıonal Studıes Congress, (BILMES EN-NAT 2018), 05-08 Temmuz, Nevşehir/Türkiye.

Yalçınkaya, S., Yalçınkaya, M.,A. "Gıda İşleme Tekniklerinde Kullanılan Bilgisayar Uygulamaları", II.Internatıonal Scıentıfic And Vocatıonal Studıes Congress, (BILMES EN-NAT 2018), 05-08 Temmuz, Nevşehir/Türkiye.

Ulusal Toplantıda Sunularak Tam Metin Olarak Yayımlanan Bildiri

Bingöl,O.,Yalçınkaya,M.A., Tosun,O.,2015. "Bilgisayar Arayüzlü DsPIC Kontrollü Fırçasız Doğru Akım Motoru Sürücü Sistemi", Akademik Bilişim Konferansı 2015, 4-6 Şubat, Eskişehir.

Yüksek Lisans Tezi

Yalçınkaya, M.A., 2015. Gelişmiş Hedef Odaklı Siber Saldırılar, Süleyman Demirel Üniversitesi, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 186s, Isparta.