T.C.
İSTANBUL AYDIN ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

WEB YAZILIMLARINDA GÜVENLİK PROBLEMLERİ ÜZERİNE ARAŞTIRMA

YÜKSEK LİSANS TEZİ

NARMİN MAMMADOVA
Y1313.010033

Bilgisayar Mühendisliği Ana Bilim Dalı
Bilgisayar Mühendisliği Program

Tez Danışmanı: Prof. Dr. Ahmad BABANLI

Şubat - 2017

T.C.
İSTANBUL AYDIN ÜNİVERSİTESİ
FEN BİLİMLER ENSTİTÜSÜ MÜDÜRLÜĞÜ

Yüksek Lisans Tez Onay Belgesi

Enstitümüz Bilgisayar Mühendisliği Ana Bilim Dalı Bilgisayar Mühendisliği Tezli Yüksek Lisans Programı **Y1313.010033** numaralı öğrencisi **Narmın MAMMADOVA**'nın **"WEB YAZILIMLARINDA GÜVENLİK PROBLEMLERİ ÜZERİNE ARAŞTIRMA"** adlı tez çalışması Enstitümüz Yönetim Kurulunun 10.01.2017 tarih ve 2017/01 sayılı kararıyla oluşturulan jüri tarafından oybirliği ile Tezli Yüksek Lisans tezi olarak kabul edilmiştir.

Öğretim Üyesi Adı Soyadı	İmzası
Tez Savunma Tarihi :03/02/2017	
1)Tez Danışmanı: Prof. Dr. Ahmad BABANLI	.....
2) Jüri Üyesi : Yrd. Doç. Dr. Metin ZONTUL	.....
3) Jüri Üyesi : Yrd. Doç. Dr. M. Ahmed SHAH	.....

Not: Öğrencinin Tez savunmasında **Başarılı** olması halinde bu form **imzalanacaktır**. Aksi halde geçersizdir.

YEMİN METNİ

Yüksek Lisans tezi olarak sunduğum "WEB YAZILIMLARINDA GÜVENLİK PROBLEMLERİ ÜZERİNE ARAŞTIRMA" adlı çalışmanın, tezin proje safhasından sonuçlanmasına kadar bütün süreçlerde bilimsel ahlak ve geleneklere aykırı düşecek bir yardıma başvurulmaksızın yazıldığını ve yararlandığım kaynakların kaynakça'da gösterilenlerden oluştuğunu, bunlara atıf yapılarak yararlanılmış olduğunu belirtir ve onurumla beyan ederim. (03/02/2017)

Narmin MAMMADOVA

ÖNSÖZ

Tez çalışmasında öncelikle web yazılımlarındaki güncel güvenlik zafiyetleri açıklanmıştır. Bu zafiyetlerin nasıl tespit edildiği ve çözüm önerileri ele alınmıştır. Tezin uygulama kısmında güncel zafiyetler gösterilmiştir. Bu zafiyetlere çeşitli öneriler uygulanarak güvenli sonuçlar elde edilmiştir. Çalışmam boyunca değerli fikir ve önerileriyle beni yönlendiren, her konuda destek veren, gösterdiği sabır ve katkılarıyla bilgilerini esirgemeyen danışmanım Prof. Dr. Ahmed BABANLI'ya, eğitimim süresince emeği geçen tüm hocalarıma teşekkürü bir borç bilirim.
Ayrıca tez aşamasında bana gereken manevi desteği ve sabrı gösteren arkadaşım Alican AKKUŞ'a teşekkür ederim.

Şubat 2017 YL Öğrencisi Narmin MAMMADOVA

İÇİNDEKİLER

Sayfa

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

WEB YAZILIMLARINDA GÜVENLİK PROBLEMLERİ ÜZERİNE ARAŞTIRMA

ÖZET

Web uygulamaları genel olarak sonuç odaklı ve hızlı geliştirilme sürecine sahiptir. Geliştirme anında güvenlik kriterleri coğu zaman bilinçli yada bilinçsiz olarak gözardı edilir. Bu nedenle, geliştirilen uygulamaların davranması gerektiğinden farklı şekillerde davranmasına neden olacak güvenlik açıklarına olanak verilmektedir. Geliştirilen web uygulamalarında ki bu zafiyetler ise kötü niyetli kullanıcılar için bir fırsat oluşturur. Web uygulamalarının geliştirilmesinde yapılan bir diğer yanlış ise uygulamanın güvenliği ile network katmanındaki güvenliğin birbirine karıştırılmasıdır. Firewall'ın (güvenlik duvarı) web uygulamalarını korumadığı bilinmelidir. Bu nedenle, geliştirilen uygulamalar için mutlaka sızma ve güvenlik testlerinin yapılması gereklidir.

Yukarıda belirtilen zafiyet ile birlikte kötü niyetli kullanıcıların bulunması sistem üzerinde bir takım risklerin oluşmasına neden olur. Bu riskler ise sistem üzerinde ki diğer kullanıcılara da sıçrayabilmektedir. Risk'in tanımını aşağıdaki gibi verebiliriz:

Risk = Tehdit + Zafiyet

Tez yazımındaki amaç, web uygulamalarında bulunan güncel zafiyetleri ele almak, bu zafiyetlerin oluşmasına neden olan düşünce ve yaklaşımların neler olduğunu öğrenmek, bu yaklaşımlara alternatif olarak neler yapılabileceği ve zafiyetlerin giderilmesi için hangi önlemlerin alınabileceğini ifade etmek amaçlanmaktadır.

Anahtar Kelimeler: *Web uygulama güvenliği, SQL Enjeksiyonu, Siteler arası İstek Sahteciliği, Siteler Arası Betik Çalıştırma, Güvenli Soket Katmanı, Aynı Kaynak Politikası.*

RESEARCH ON PROTECTION SECURITY PROBLEMS IN WEB APPLICATIONS

ABSTRACT

Web applications generally have a result-oriented and rapid development process. At the development time, security criterion ignored consciously or unconsciously. Therefore, it allows security vulnerabilities that will cause differently behave the developed applications than they should behave. These vulnerabilities in developed web applications provide an opportunity for malicious users. Another mistake made in the development web applications is that the security application's security is confused with the network layer's security. It should be known that Firewall (security gateway) does not protect web applications. Therefore, it is absolutely necessary to conduct infiltration and safety tests for the developed applications. The above mentioned weakness and the presence of malicious users causes some risks on the system. These risks can also spread to other users on the system. We can give the definition of risk as follows:
Risk = Threat + Vulnerability
The goal of writing thesis is that to handle the current weakness in the web applications, to learn what are the thoughts and approaches that cause these weaknesses to occur, what alternatives can be made to these approaches and what measures can be taken to eliminate weaknesses.

Keywords*: Web Application Security, SQL Injection, Cross Site Request Forgery, Cross Site Scripting, Secure Socket Layer, Same Origin Policy.*

1. GİRİŞ

"Dünyayı 80'lerde kablolarla, 90'larda bilgisayar ağlarıyla bağladık. Bugünse dünyayı yazılımlarla bağlıyoruz. Bu yazılımların güvenli bir şekilde tasarımı, geliştirilmesi ve uygulamaya sokulması günümüz evrimleşen dünyası için kritiktir" (*Mark Curphey, Microsoft ürün geliştirme başkanı ve OWASP kurucusu*) [1].

Yazılım Güvenliği, statik olarak düşünülen bir kavramdan ziyade bir süreçten ibarettir. Kullanıcıların ve sistemlerin yapmış olduğu en büyük hatalardan biri de bir ürün veya destek alarak güvenliğin sağlandığına inanmalarıdır. Alınan bir ürün ile güvenlik seviyesi tam olarak çözümlenmez. Daha doğru olan yaklaşım ise güvenlik politikaları oluşturmaktır. Kullanıcıların en son isteyecekleri durum, bilgilerinin çalınması, kişisel verilerinin kaybolmasıdır. Güvenlik kavramı gibi yazılım geliştirme kavramı da bir süreçtir. Bu süreç: analiz, tasarım, planlama, kodlama, test, kurulum, destek gibi bölümlere ayrılabilir. Hiçbir hata içermeyen yazılım %100 "güvenli yazılım" demek değildir. Ulaşılmaz ve çalıştırılamayan bir yazılım dışında %100 güvenli yazılım yoktur. Güvenli yazılım, güvenliği göz ardı etmeden tasarlanmış, güvenlik kontrolleri ile geliştirilmiş ve güvenli bir durumda kullanıma sunulmuş yazılım demektir. Güvenli yazılım ile saldırı riski hala bulunsa da, bu durumdan ötürü ortaya çıkacak problemlerin büyük oranda önüne geçilmiş olunur [1]. Yazılım geliştirme süreçlerinde yapılan hatalardan biri de süreçler bittikten sonra güvenlik adımının gerçekleştirilmesidir. Süreçler sonunda yapılacak güvenlik iyileştirmeleri yetersiz kalacaktır. Bu nedenle geliştirme süreçlerinin tamamına yayılacak şekilde bir güvenlik politikası belirlenmeli ve her süreç içerisinde politikanın gerçekleştirildiğinden emin olunmalıdır. Güvenlik sürecinin yazılım geliştirme sürecine entegre edilmesi işlemi sonucunda Güvenli Yazılım Geliştirme (GYG, secure SDLC – secure software development life cycle) süreçleri ortaya çıkmaktadır. Yapılan araştırmalar ve edinilen tecrübelerden yola çıkarak GYG süreçlerinin maliyet ve zaman açısından bir gereksinim olduğu ortaya çıkmaktadır.

Güvenlik politikasının geliştirme süreçlerine dahil edilmemesinden kaynaklı ilerde birçok sorun, hata, zafiyet benzeri geri dönüşler alınabilmektedir. Bu geri dönüşler sonucunda ise sistem üzerinde yama geliştirmek, test ve bakım gibi ekstra maliyet ve

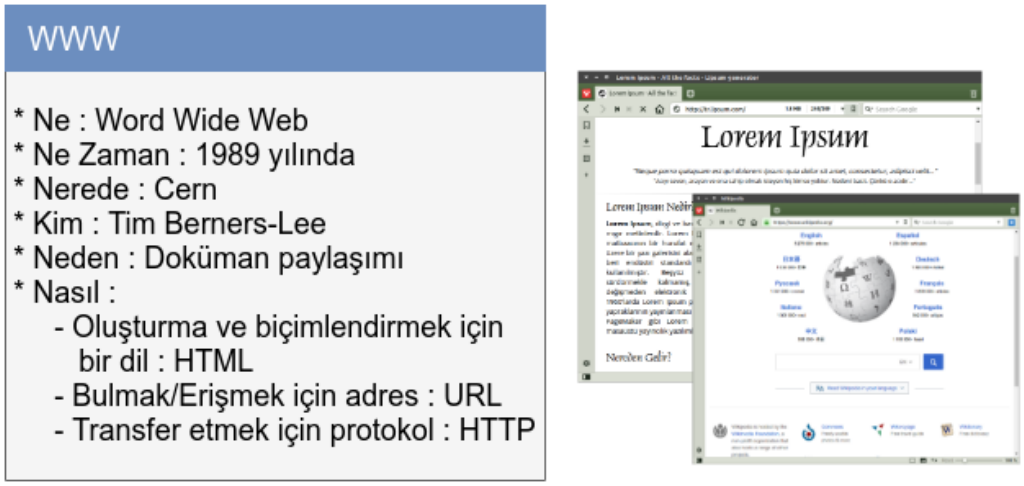
zaman kaybına dönüşmektedir. Sonradan yapılan yama benzeri işlemler geliştirme anında fark edilip düzeltilebilen durumlardan çok daha fazla efor'a ihtiyaç duymaktadır. Ayrıca yukarı da ifade edilen sorun, hata, zafiyet benzeri geri dönüşler kurumun ve/veya sistemin imajını zedelemekle kalmayıp hukuki açıdan birtakım sorunlara da yol açmaktadır. Yazılım geliştirme sürecine başlamadan önce uygulama yapısına ve eldeki kaynaklara uygun sekilde GYG stratejisi belirlenmeli ve stratejinin gerektirdiği güvenlik aktiviteleri hayata geçirilmelidir.

Günümüzde bu amaç için bazı kurum ve kuruluşlar bir takım araçlar geliştirerek farklı güvenlik ihtiyaçlarına göre çalışabilen esnek birtakım ürünler ortaya çıkmıştır. Bunlardan birkaçı SAMM (*Software Assurance Maturity Model*), BSIMM (*Build Security in Maturity Model*), Microsoft SDL (*Security Development Life Cycle*) gibi çalışmalar ön plana çıkmıştır. Örnek olarak vermiş olduğumuz tüm GYG'ler değişik metotlar kullansalar da ortak amaçları web yazılımlarının güvenliğini baştan sona sağlamaktır [2].

Özellikle OWASP (*Open Web Application Security Project*) projesi olan SAMM açık kaynak olmasının yanında organizasyonların yapısına ve güvenlik ihtiyaçlarına göre uyarlanabilen bir model ortaya koymaktadır [1].

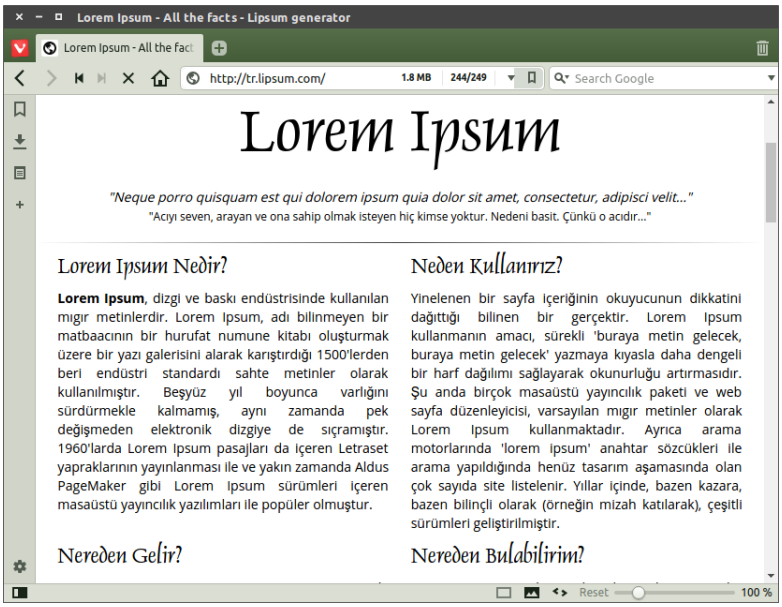
2. WEB SİSTEMLERİNİN AMAÇLARI VE YAPISI

2.1 Web Sistemlerinin Gelişimi



Şekil 2.1 : World Wide Web'in gelişimi

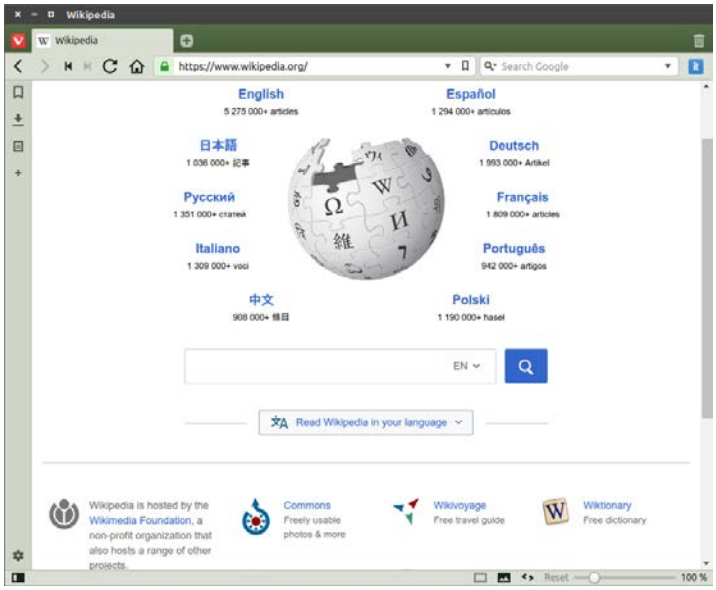
Internetin ilk günlerinde www(word-wide-web) sadece statik belgeleri içeren web sitelerinden oluşmaktaydı. Tarayıcılar, bu belgeleri görüntülemek için icat edildi.



Şekil 2.2 : Static bilgi içeren web sitesi

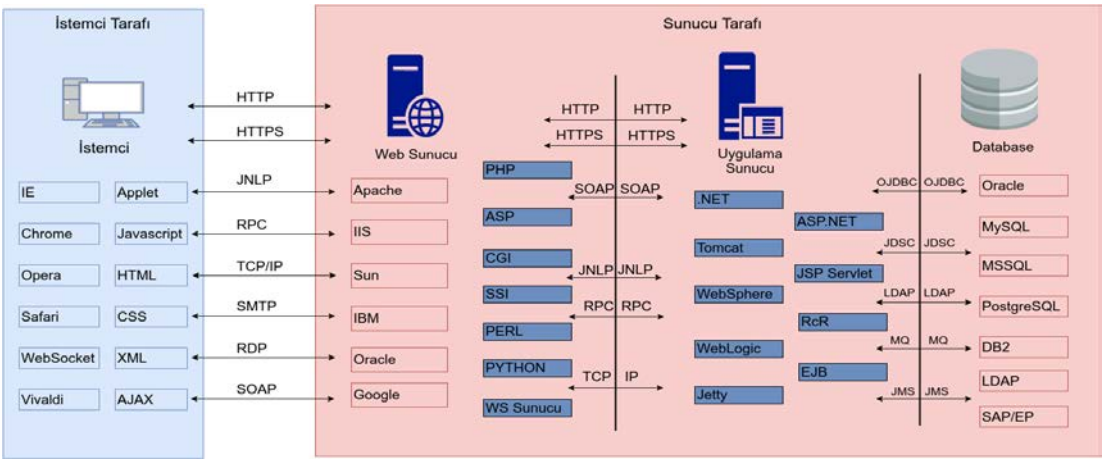
İnternetin ilk zamanlarında bilginin serverdan tarayıcıya doğru yollu bir akımı vardı. Çoğu site kullanıcı kimliğini doğrulamıyordu, çünkü buna gerek de yoktu. Her

kullanıcıya aynı şekilde davranılıyor ve aynı bilgiler sunuluyordu. Bugün, www önceki formundan neredeyse tanınmaz haldedir. Web üzerinde olan sitelerin büyük çoğunluğu, aslında uygulamalardır.



Şekil 2.3 : Tipik bir web uygulaması

Bu uygulamalar işlevsel olup bilginin server ve tarayıcı arasında 2 yolla akımına dayanmaktadır. Bu uygulamalar kayıt ve giriş, finansal işlemler, arama, kullanıcıların kendi içeriklerine yetkileri dahilinde erişim imkanlarını destekler. Bilginin çoğu özel ve yüksek hassasiyetdedir. Güvenlik bu yüzden önemli bir konudur [3]. Hiçkimse şahsi bilgilerini yetkisiz kullanıcılara gösteren web uygulamasını kullanmayı istemez. Her uygulama kendi içerisinde farklıdır ve benzersiz güvenlik açıkları içerebilir. Uygulamaların çoğu, yazılım geliştiricileri tarafından sadece kod içerisinde oluşabilecek kısmi güvenlik sorunlarını anlayarak in house olarak geliştirilmektedir. 15 yıl önce eğer para transfer etmek istersek, bankaya gitmeli ve vezneci tarafından para transferini gerçekleştirmeli oluyorduk. Bugün ise gerekli web uygulamasını kullanarak para transferi işlemini kendimiz gerçekleştire biliyoruz.



Şekil 2.4 : İstemci-sunucu mimarisi

2.1.1 Web uygulamaların ortak fonksiyonları

Web uygulamaları online işlemler yapa bilmemiz için pratik faydalı fonksiyonları yerine getirmek için oluşturulmuşlar.

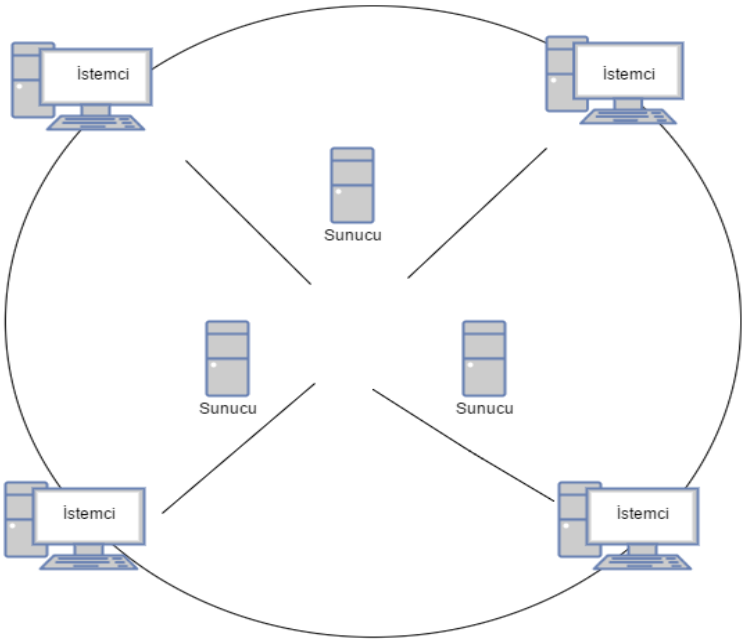
Son yıllardakı bazi web uygulama fonksiyonları aşağıdaki gibidir:

- Alışveriş (Amazon)
- Sosyal Ağ (Facebook)
- Web Arama (Google)
- Web Mail (Gmail)

Bazı teknik faktörler interneti nasıl kullanmamız gerektiği sayesinde meydana gelmiştir. HTTP, ana iletişim protokolu, www'ya erişmek için kullanılmaktadır. Bu protokol iletişim hataları durumunda esneklik sağlar ve her kullanıcı için sunucuya bağlanıp ayrı bir yetki açmak ihtiyacını ortadan kaldırır [3]. HTTP, ayrıca her hangi bir ağ yapılandırılmasında güvenli bağlantıya izin verir. Web uygulamalarını geliştirmek için kullanılan ana teknoloji ve diller nispeten basittir. Platform ve geliştirme toollarının geniş çeşitte olması yeni başlayanlar tarafından güçlü uygulamalar geliştirmek için imkan sağlamaktadır. Bunlara ek olarak geniş miktarda açık kaynak kod ve başka kaynaklar da mevcuttur.

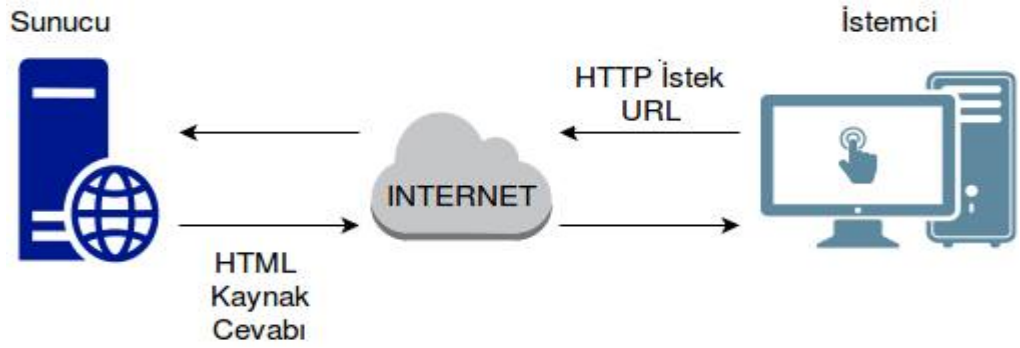
2.2 Web Sistemlerinin Yapısı

Web, bilgisayar, yazıcı vb sistemlerin bir arada çalıştığı bir sistemdir. Milyarlarca istemci (*Mozilla, Safari, IE*) ve sunucu (*WebServer, Apache*) ile birlikte kablolu ve/veya kablosuz ağlar sayesinde birbiri ile iletişimde bulunurlar.



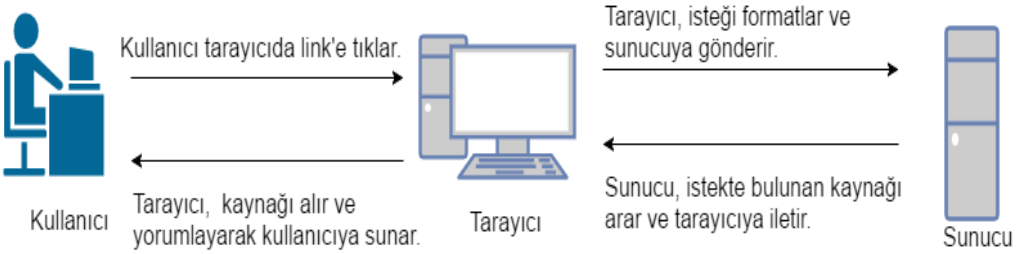
Şekil 2.5 : Web üzerindeki ağ yapısı

Sunucu, istemci'den gelen istekleri karşılar ve istemciye istediği kaynağı geri döndürür. Web tarayıcıları gibi istemciler kullanıcıların talep ettikleri kaynaklar için sunucuya istekte bulunur. Web sunucusu gelen isteği karşılar, kaynağı arar ve istemci'ye cevap olarak ilgili kaynağı iletir. İstek yapılan kaynak tipi, HTML sayfası, resim, ses dosyası veya PDF dokümanı olabilir. Sunucu, istemcinin istekte bulunduğu kaynağı bulamıyor ise istemciye "***404 Not Found***" hatası döndürür. Sunucu, fiziksel makine(donanım) veya web sunucu uygulaması(yazılım) olabilir. Client, tarayıcı uygulamaları ve insanlar olabilir.



Şekil 2.6 : Sunucu ile İstemci iletişimi

Tarayıcı, sunucu ile iletişimin nasıl kurulacağını bilen yazılımlardır. Tarayıcıların diğer bir önemli görevi de alınan HTML cevabını yorumlayarak kullanıcıya web sayfasını sunmaktır.



Şekil 2.7 : Sunucudan talep edilen kaynağın alınması

HTML (*HyperText Markup Language*), alınan kaynakta ki içeriğin kullanıcıya nasıl sunulması gerektiğini ifade eden işaretleme dilidir. Tarayıcılar, HTML kodunu yorumlayarak kullanıcıya sunar. HTML işaretleme dili, etiket(tag) ve etiketlerin sahip olduğu özelliklerden(attribute) oluşmaktadır.

Örnek olarak; ***<form></form>*** etiketi ve form etiketinin ***action*** özelliği verilebilir.

HTTP (*HyperText Transfer Protocol*), istemci ve sunucu arasında ki iletişimin kurulmasını sağlayan protokoldür. İletişim, istemcinin isteği ve sunucunun cevabı ile gerçekleşir. İstemci, HTTP istek ile istediği kaynağı sunucuya gönderir, sunucu ise HTTP cevap ile istenilen kaynağı döndürür. HTTP protokolü, TCP/IP katmanında çalışarak bu görevi gerçekleştirir.

TCP ***(****Transmission Control Protocol****),*** bilginin doğru bir şekilde iletilmesinden, IP ***(****Internet Protocol****)*** ise istemci ve sunucunun adresinin bulunmasından sorumludur.

Internet’e giriş yapmak için kullanılan ana iletişim protokolu olup, web uygulamaları tarafından kullanılmaktadır.

Iki çeşit HTTP mesaj içeriği vardır. Bunlardan biri istek(***request***), diğeri ise yanıt(***response***)'dur. Protokol temelde bağlantısızdır (***connectionless***). HTTP transfer mekanizması için TCP protokolunu kullanmasına rağmen, her değişen istek için yeni bir TCP bağlantısı açılmaktadır. Yani tarayıcı server'a bir bağlantı hazırlayıp server'a gönderiyor, server'dan cevabı alıp bağlantıyı kapatıyor. Başka sözle, bağlantı sadece tek bir request/response için geçerli olmaktadır. Çünkü bağlantı devam etmemekte, sunucu önceki isteği gönderen tarayıcının ikinci isteğini tanımamaktadır. Sunucu her isteğin yeni bir tarayıcıya ait olduğunu varsaymaktadır [3].

HTTP Istek Yapısı ve Çeşitleri

Tüm HTTP mesajları (requests ve responses) genel olarak aşağıdaki bölümlerden oluşmaktadır:

1. Request veya Response için değişen ilk satır.
2. Bir veya daha fazla başlık (header).
3. Boş bir satır.
4. Opsiyonel bir mesaj bölümü (message body).

HTTP metotları server ile client arasında iletilen veriler üzerinde işlem yapılmasını sağlar.

HTTP metotları:

- GET
- POST
- PUT
- DELETE
- TRACE
- OPTIONS

Burada örnek olması açısından sadece GET ve POST HTTP metodlarını ele alacağız.

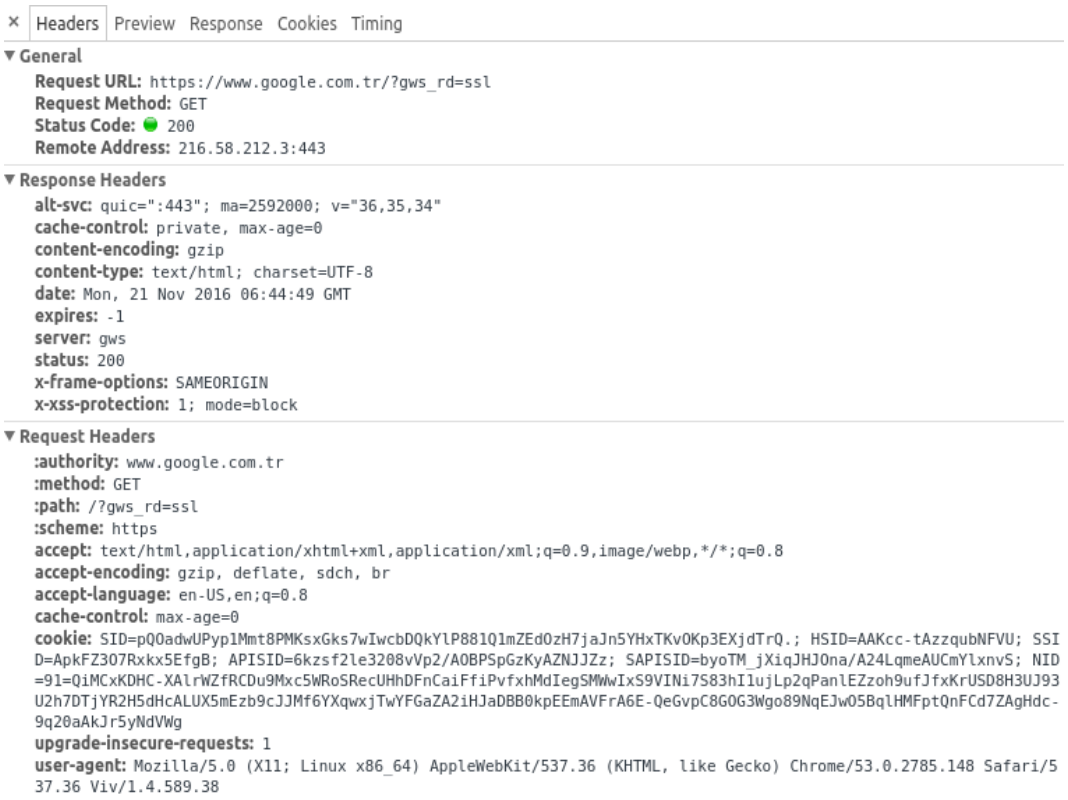
Temelde birinin diğerine göre bariz bir üstünlüğü yoktur. Fakat kullanım yerine göre avantaj ve dezavatajları vardır. Özellikle, GET metodu ile gönderilecek verinin boyutunun POST ile gönderilecek olana göre az ve sınırlı olmasıdır.

GET metodu, URL üzerinden gönderdiğimiz veya göndereceğimiz değerler için kullanılır. POST metodu ile gönderdiklerimiz ise URL üzerinde gözükmez [3]. GET ile gönderilecek karakter sayısı limitlidir, tarayıcıya göre değişir, genelde birkaç bin karakter içerebilir. GET metodu genelde basit istekler, sunucudan bilgi almak, bir kaynağın çağırılması amaçlı kullanılır. En çok kullanılan HTTP talebidir.

Genelde uygulamamızın bir sayfasını çalıştırdığımızda ve o sayfadan başka bir sayfanın çalışmasını isteyeceğimiz anda (link için) GET metodunu kullanırız.

Standart bir HTTP isteği aşağıdaki gibidir:

Şekil 2.8 : Standart bir HTTP isteği



Şekil 2.9 : HTTP istek ve cevap yapısı

Her bir HTTP isteğinin ilk satırı (istek satırı) bir birinden boşluk ile ayrılan 3 maddeden oluşmaktadır:

1. HTTP metodunun tipi. Yaygın olarak kullanılan metod GET'dir. Bu metod sunucudan istenilen kaynağı geri döndürür. Get isteğinin mesaj bölümü yoktur.
2. İstek URL. URL kaynağın isminden ilave, tarayıcıdan istenilen kaynağa gönderilen isteğe bağlı query srting adlanan parametrelerden de oluşabilir. Query String URL'de? İşareti ile gösterilir. Yukarıdaki resimde gösterilmiş URL'de query string parametre olaram name olarak **gws_rd** ve value olarak da **ssl** değerini almıştır.
3. Kullanılan HTTP versiyonudur. Internet üzerinde sadece yaygın olarak kullanılan HTTP versiyonları 1.0 ve 1.1 'dir ve çoğu tarayıcılar default olarak 1.1 versiyonunu kullanıyor. 1.0 ile bağlantı yaptığımızda her istek için ayrı bır bağlantı açmak gerekiyor. 1.1 de ise tek bağlantı üzerinden istediğimiz kadar istek yollayabiliyoruz.

Yukarıda verilmiş olan örnek istek ile ilgili bazı noktalar vardır:

Referer başlığı, istekde bulunan URL'in kökenini gösterir. Örneğin, kullanıcı sayfa üzerinde herhangi bir linke tıklaya bilir.

User-Agent başlığı, istek üreten tarayıcı veya browser hakkında bilgi taşıyor. Çoğu tarayıcılar tarihi nedenlerden dolayı Mozilla prefixsini dahil ediyor. User-Agent stringi orijinal dominant Netscape tarayıcısı tarafından kullanılır ve diğer tarayıcılar bu standarda uygun olduklarını göstermek için onu kullanmaktadır.

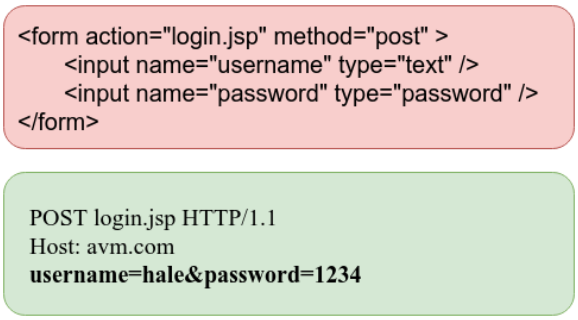
Host başlığı, erişilmek istenilen full URL'deki host ismini belirtiyor. Bu başlık birçok site aynı sunucu üzerinde barındırıldığında önemli olur, çünkü, genelde isteğin ilk satırında gönderilen URL host ismini içermemektedir.

Cookie başlığı, sunucunun tarayıcı için hazırladığı ilave parametreleri göndermek için kullanılıyor.

Post İsteği ise, karmaşık istekler ve özellikle form alanlarına girilmiş olan verileri uygulamaya göndermek için kullandığımız bir metottur. Bu gönderim, form alanlarının ismi ile kullanıcı tarafından bu alanlara girilmiş olan değerlerin

oluşturduğu bir grupu içermektedir. Bu metod ile veri gizli gönderildiği için kullanıcının veri transferine müdahalesi söz konusu değildir.

Bundan dolayı da veri transferinde kullanılan en güvenli metoddur. Parametreler mesaj gövdesine konulur. Aşağıdaki resimde tipik bir post isteği gösterilmiştir [3].



Şekil 2.10 : POST isteği.

Çizelge 2.1 : GET ve POST metodlarının karşılaştırılması

#	GET	POST
Güvenlik	Daha az güvenli	Daha çok güvenli (çünkü parametreler tarayıcının geçmişine kaydedilmiyor veya web server loglarında tutulmuyor)
Gönderilen Istek Parametreleri	URL'in izin verdiği kadar (mak. 2048 karakter)	Sınırsız
Gönderilen Bilgi Tipi	Yalnızca ASCII karakterler	Sınırsız, binary bilgi de gönderilebilir
Geri Butonu Davranışı	Istek baştan oluşturulur	Tarayıcı kullanıcıyı isteğin yeniden oluşturulacağına dair uyarır
Bookmark	Bookmark yapılabilir	Bookmark yapılamaz
History (Geçmiş)	Parametreler tarayıcının geçmişinde kalıyor	Parametreler tarayıcının geçmişinde kaydedilmiyor
Visibility (Görünürlük)	Veri URL'de herkese görünür	Veri URL'de görüntülenmiyor

HTTP Cevap Yapısı ve Çeşitleri ve Durum Kodları (Status Codes)

Cevaplar, durum kodlarıyla (HTTP status code) başlarlar. GET/POST gibi metotlara karşın, sunucudan durum kodları döner [4]. Durum kodları üç haneli sayılardan oluşur. İlk rakam cevabın hangi genel kategoriye düştüğünü belirtir. Bu kategorileri aşağıdaki gibi sıralayabiliriz:

- **1xx** Bilgi vermek amaçlı
- **2xx** Başarılı istek
- **3xx** Client'ın başka bir URL'e yönlendirilmesi
- **4xx** Client taraflı hata
- **5xx** Sunucu taraflı hata

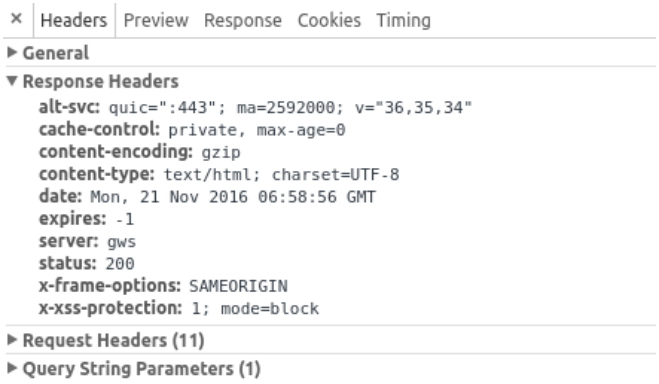
Sadece özel şartlarda kullanılan çok sayıda durum kodları vardır. Karşılaşılması muhtemel olan durum kodları aşağıdaki gibidir:

Çizelge 2.2 : HTTP Durum kodları

Durum Kodu	Durum Açıklaması
100 Continue (Devam)	Sunucu isteğin ilk bölümünü aldığını ve geri kalanını beklediğini belirtmek için bu kodu döndürür. Sunucu istek tamamlandığında ikinci cevabı yolluyor.
200 OK (Başarılı)	Sunucunun istenilen kaynağı başarılı bir şekilde döndürmesi anlamına gelir.
201 Created (Oluşturuldu)	İsteğin başarılı olduğunu ve sunucuda yeni bir kaynak oluştuğunu gösteriyor.
301 Moved Permanently (Kalıcı olarak URL taşındı)	Tarayıcı *Location* başlığında belirtilen farklı bir URL'e yönlendirilir. Tarayıcı gelecekte orijinal URL yerine yeni URL'i kullanmalıdır.
302 Found (Geçici olarak URL taşındı)	Sunucu şu anda isteğe farklı bir konumda bulunan bir sayfayla yanıt veriyor; ancak istekte bulunanın gelecek istekler için özgün konumu kullanmaya devam etmesi gerekiyor.
304 Not Modified (Değiştirilmedi)	Istenen sayfa, son istekten bu yana değiştirilmedi. Sunucu bu yanıtı döndürdüğünde sayfanın içeriğini döndürmez.
400 Bad Request (Yanlış istek)	Sunucu isteğin söz dizimini anlamadı. (URL'de boşluk karakterinin konulması gibi).
401 Unauthorized (Yetkilendirilmemiş)	Bu istek için kimlik doğrulaması gerekiyor. Sunucu giriş yapmadan görüntülenemeyen sayfa için bu yanıtı döndürebilir.
403 Forbidden (Yasak)	Kimseye istekte bulunan kaynağa erişime izin verilmiyor.
404 Not Found (Bulunamadı)	Sunucu istenen sayfayı bulamıyor. Örneğin, istek, sunucuda bulunmayan bir sayfa için yapılmışsa, sunucu genellikle bu kodu döndürür.

Durum Kodu	Durum Açıklaması
413 Request Entity Too Large (Istek varlığı çok büyük)	Eğer istekte suncuuya uzun string datası gönderiyorsak, sunucu bu isteği işleyemeyeceği durumlarda bu durum kodunu döner.

Tipik bir HTTP cevabı (response) aşağıdaki gibidir:



Şekil 2.11 : Tipik bir HTTP cevabı

Her bir HTTP cevabının ilk satırı boşluk ile bir birinden ayrılan 3 maddeden oluşmaktadır:

1. Kullanılan HTTP versiyonu.
2. Yapılan isteğin sonucunu gösteren sayısal durum kodu. 200 en genel durum kodudur. Anlamı, yapılan isteğe karşı, geri dönen cevabın başarılı olduğunu ve istenen kaynağın bulunduğu anlamına gelmektedir.
3. Cevabın durumunu açıklayan metinsel “neden ifade”'si.

Yukarıda verilmiş örnek ile ilgili bazı diğer noktalar da vardır [3]:

Server başlığı, kullanılan web server yazılımını gösteriyor.

Set-Cookie başlığı, sunucudan tarayıcıya bir cookie bilgisi gönderilmesini, ve daha sonra bu bilginin bu sunucuya olan sonraki isteklerde *Cookie* başlığı altında gönderilmesini gösteriyor.

Pragma başlığı, cevabı önbellek'de (cache) tutmaması için tarayıcını bilgilendiriyor.

Neredeyse tüm HTTP cevapları başlıklardan sonra boş çizgi ile ayrılan mesaj gövdesinden oluşuyor. *Content-type* başlığı, bu mesajın gövdesinin HTML

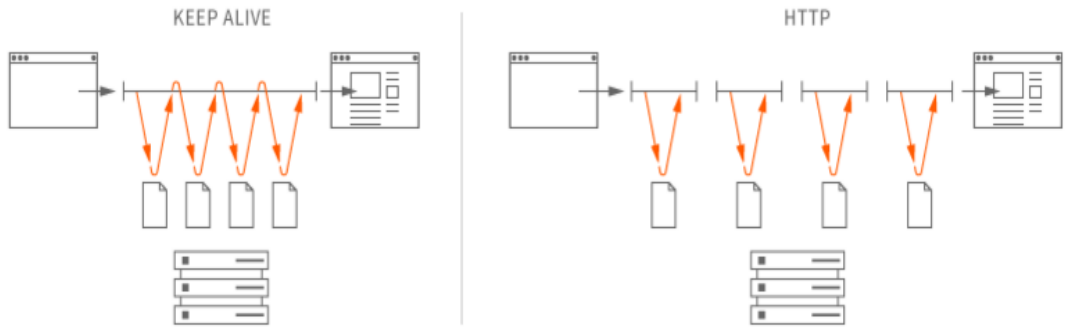
dökümanından oluştuğunu gösteriyor. *Content-Length* başlığı ise, mesajın gövdesinin kaç byte uzunluğunda olduğunu gösteriyor.

HTTP Başlıkları

HTTP çok sayıda başlık (header) destekliyor. Bazı başlıklar hem isteklerde hem de cevaplarda kullanılır ve bazıları ise mesaj tipierine özeldir.

Genel Başlıklar

Connection başlığı, bağlantıyı ne kadar süre açık tutacağını gösteriyor. Keep-Alive istemci ve tarayıcı arasında kalıcı bağlantıyı aralıklı kesintiden koruyor [5]. HTTP Keep-Alive veya kalıcı bağlantı (persistent connection) olarak da biliniyor. Bu her yeni istek için yeni bir bağlantı açılması yerine tüm bağlantıların aynı TCP üzerinden sağlanmasına olanak sağlıyor.



Şekil 2.12 : HTTP Keep Alive Yapısı

Varsayılan HTTP bağlantısı genellikle her istek tamamlandıktan sonra kapanıyor, yani sunucu cevabı teslim ettikten sonra TCP bağlantısını kapatıyor. Çoklu istekler için bağlantıyı açık tutmak için, keep-alive başlığı kullanıla bilir [5].

HTTP Keep-Alive olmadan nasıl çalışıyor?

1. Istemci sunucu ile etkileşim kurmak ve dosyayı almak için yeni bir bağlantı oluşturmalıdır.
2. Istemci yeni bağlantıyı kullanarak HTMP sayfası isteğinde bulunuyor ve bu bağlantı dosyayı aldıktan sonra sona eriyor (terminate).
3. Tarayıcı bu HTML dosyasını yorumluyor ve tüm sayfayı göstermek için başka dosyaların gerekip gerekmediğini kontrol ediyor.

4. Kapsamlı bir analizden sonra, istenen dosyaların her biri için yeni bağlantı oluşturuyor.
5. Tek bir HTML sayfası tüm web sayfasını oluşturan çeşitli kaynaklar içerebilir. Bu kaynaklara htmHTML kodu, CSS dosyaları, JS dosyaları, resimler ve multimedya dosyalarını gösterebiliriz. Örnek olarak, eğer HTML 4 CSS, 2 JS dosyası ve 3 resimden oluşursa, bu dosyaları almak için ilave olarak 9 isteğin olması gerektiğini gösteriyor. Ama bu mekanizma çok verimsizdir, özellikle karmaşık web sayfalarının çok sayıda elemanları varsa [5].

KEEP-ALIVE ihtiyacı

Çok sayıda bağlantı oluşturmak yükleme süresini azalta bilir. Ayrıca sunucu üzerinde birçok kaynak kullanılır. Tüm bu dosyaların aktarımını Keep-Alive 'ı enable (aktif) ederek defalarca yeni bağlantı açıp kapamak yerine tek bir bağlantı üzerinden yapabiliriz [5]. Eğer bu bağlantı aktif değilse, o zaman web sayfasını görüntülemek oldukca uzun sürebilir.

KEEP-ALIVE faydaları

CPU kullanımını azaltıyor: Yeni TCP bağlantıları oluşturmak için CPU ve bellek kullanımı gibi çoklu kaynaklar kullanıla bilir. Bağlantını kalıcı yaparak kaynak kullanımını azalta biliriz.

Web sayfasını hızlandırmak: Çoklu dosyalar aynı bağlantıyı kullanmakla gecikmeyi azalta bilir ve web sayfalarının daha hızlı yüklenmesine izin verir.

HTTPS: Bu bağlantı şekli için Keep-Alive özelliğinin aktif edilmesi tavsiye edilir.

Sonuç olarak, modern tarayıcıların hepsi kalıcı bağlantıyı kullanıyor. Bunu uygulamamızdan ve proxy server ayarlarından kontrol ede biliriz. HTTP/10. Aksine HTTP/1.1 versiyonunda, bu bağlantı şekli varsayılan olarak açık geliyor.

Content-Encoding, mesaj gövdesindeki içerik için kullanılan gzip gibi bazı uygulamalar tarafından hızlı aktarım için cevapları sıkıştırmada kullanılan encoding türünü belirliyor. Hale'ni x ile evez edip her geçen yerde x yazarak çağırmak sıkıştırma algoritmasına örnektir.

GZIP (*GNU Zip*), bir tür sıkıştırma yöntemidir. Web açısından baktığımızda gzip, özellikle HTML, CSS ve JS gibi **metin tabanlı dosyalarda** çok ciddi derecede sıkıştırma sağlayan bir sıkıştırma yöntemidir. Birçok web sunucusu ve güncel tüm tarayıcılar gzip sıkıştırmasını desteklemektedir [5].

Istek başlığında,

Accept: text/html, application/xml, image/jpeg

Accept-Encoding: gzip, deflate

bu başlıklara dikkat edersek;

Accept, istemcinin (yani tarayıcının) kabul ettiği MIME (Multipurpose Internet Mail Extensions) tiplerini, **Accept-Encoding** ise kabul ettiği kodlama biçimlerini gösterir.

Tarayıcı burada gzip ve deflate algoritmaları ile sıkılaştırılmış verileri kabul ettiğini gösteriyor. Isteği gönderdiğimiz sunucu da eğer bu sıkılaştırmayı kabul ediyorsa, verileri sıkılaştırarak gönderiyor.

Cevap başlığında ise;

Content-Encoding: gzip

Content-Type: text/html, charset = UTF-8

bu başlıklara dikkat edersek;

Content-Type ile dönen verinin MIME tipinin text/html olduğu ve bu dosyanın UTF-8 ile kodlandığı belirtilmiş ve ardından ***Content-Encoding*** **i**le de gelen verinin hangi algoritmayla sıkıştırıldığı gösterilmiştir.

Content-Length, mesaj gövdesinin içeriğinin uzunluğunu byte cinsinden belirliyor.

Content-Type, mesaj gövdesinin içeriğinin tipini belirliyor. HTML dökümanları için text/html gibi.

HTTP Metotları

Web uygulamalarına saldırı zamanı, neredeyse genel olarak kullanılan metotlar GET ve POST'dur. Bu yöntemler arasında bazı önemli farkların olduğunun farkına varmalıyız.

GET metodu, kaynağı almak için tasarlanmıştır. Bu metodu istekte bulunduğumuz kaynağa URL query string'de parametre yollamak için kullanabiliriz.

Bu kullanıcılara dinamik olan kaynağı yeniden kullanmaları için URL'i bookmark yapma olanağı sağlıyor.

POST metodu, eylemleri gerçekleştirmek için tasarlanmıştır. Bu metotla istek parametreleri hem URL query string'de hem de mesaj gövdesinde gönderile bilir. URL bookmark olunmasına rağmen, mesaj gövdesinde gönderilen bazı parametreler bookmarkdan hariç olacaktır. Çünkü POST metodu eylemleri gerçekleştirmek için tasarlanmıştır [3]. Eğer kullanıcı bu metodu kullanarak eriştiği sayfaya tarayıcının BACK tuşuna tıklayarak geri dönse, tarayıcı isteği otomatik olarak yeniden yayınlamaz.

Yerine, aşağıdaki resimde gösterildiği gibi kullanıcını yapmak istediği iş ile ilgili uyarır ve kullanıcıların farkında olmadan bir eylemi birden çok kez gerçekleştirmelerini önler. Bu nedenle, POST isteği herhangi bir eylem yapıldığında kullanılmalıdır.

Şekil 2.13 : POST isteğinin tekrarı

GET ve POST metotlarına ilave olarak, HTTP protokolu belirli amaçlar için oluşturulmuş çok sayıda diğer metotları da desteklemektedir. Bu metotlar aşağıdaki gibidir:

HEAD metodu GET metodu ile işlem olarak aynı işi yapar. HEAD metodunda çağrılan adresden cevap başlıkları ile bilgi alınır. Fakat sunucu cevap dönerken sadece header kısmı döner, mesajın body kısmı dönmez. Tüm bilgi header'de tutulur. Sunucu GET isteğine karşılık gelen aynı başlıkları dönmelidir. Bu nedenle, bu metodu kaynağın GET isteği ile yapılmadan önce sunucuda var olup olmadığını kontrol etmek için kullanabiliriz [3].

TRACE metodu diagnostic amaçlar için kullanılıyor. Sunucu cevabın body kısmında aldığı isteğin tam olarak içeriğini döndürmelidir. Bu metot isteği işleyen istemci ve sunucu arasındakı herhangi bir proxy sunucuların etkisini tespit etmek için de kullanılır. Hem cevabı yolluyor hem de o ıstegı nasıl gonderdıgımı gosterıyor [3].

OPTIONS metodu sunucunun desteklediği HTTP metotlarının bir listesini döndürür. Bu liste cevap ile *ALLOW* başlığı altında geliyor [3].

PUT metodu belirli bir kaynağı sunucuya yüklemek içindir. Bu metot enable ise, uygulamaya saldırı zamanı sunucuya gelişigüzel girdi yükleyip sunucu üzerinde çalıştırmak mümkün olabilir [3].

2.3 Web Sistemlerinin Amaçları

Web uygulamaları, internet üzerinde bulunan ve tarayıcılar vasıtasıyla erişilebilen uygulamalardır. Bu programlar, kişisel ve/veya iş konusunda farklı türlerde yardımcı olacak, kolaylaştıracak birtakım fonksiyonlar sunar.

Web uygulamalarına örnek olarak, ürün katalogları, arama motorları, proje yönetim araçları, e-mail, sosyal medya programları gibi örnekler verilebilir. Web uygulamaları statik veya dinamik olabilir. Statik web uygulamaları tüm kullanıcılar için aynı sonuçları üretirken, dinamik web uygulamaları ise kullanıcı etkileşimli uygulamalardır. Örneğin, web üzerinde çalışan stok takip programında her kullanıcının kendine ait kişisel verileri ve bu verileri saklayabileceği bir alanı vardır. Kullanıcı stok bilgisini görüntüleyebilir, değiştirebilir ve kayıt edebilir. Web uygulaması geliştirmenin avantajları olarak özel bir bilgisayar ve/veya işletim sisteminden bağımsız olarak geliştirilebilir ve erişilebilir olmasını gösterebiliriz. Bir web uygulamasına, farklı işletim sistemlerinden, farklı bilgisayar ve cihazlardan erişilebilir. Web uygulamaları, genelde sunucu (*PHP, ASP*) taraflı ve istemci (*HTML, JS*) taraflı teknolojilerle geliştirilirler [4]. Web uygulamalarının yapısında istemci-sunucu mimarisi yer almaktadır. Web uygulamalarının geleceğinde ise farklı işletim sistemerine özel yapıların web üzerinden kullanıma açılması düşünülebilir. Örneğin, Windows işletim sistemi üzerinde çalışan kelime işlemci programı web üzerinden erişilebilir hale getirilerek işletim bağımsız olarak ilgili uygulamanın herkez tarafından kullanılması sağlanabilir. Bu uygulamalara örnek olarak ise Google Apps, Microsoft Office Live ve WebEx WebOffice gösterilebilir.

3. WEB SİSTEMLERİNDE GÜVENLİK

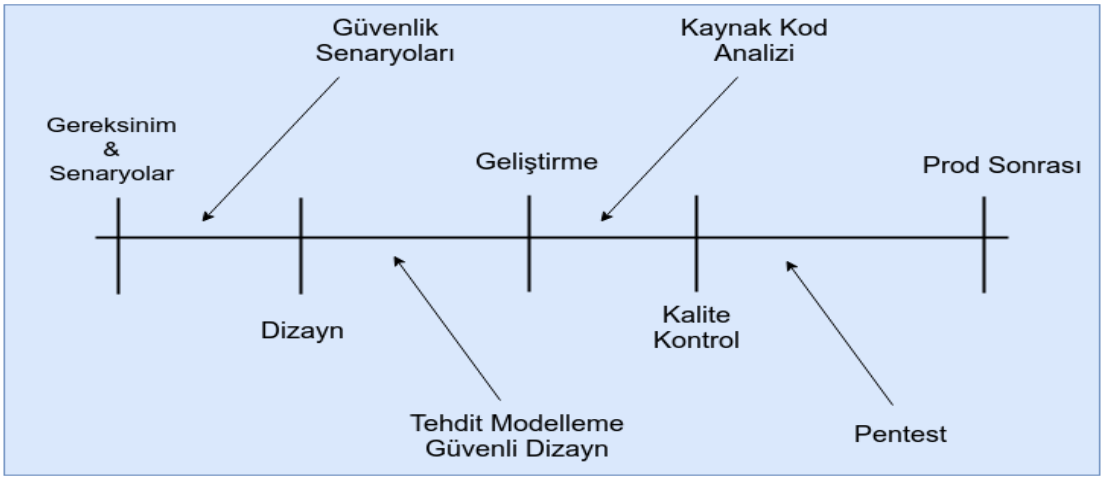
Güvenliğin yazılımlar için yeni bir kavram gibi gözükmesi yanlış bir algının sonucudur. Yazılımların başlangıç zamanlarından beri önem arz eden bu konu, internetin gelişmesi ve web uygulamalarının ve dolayısıyla yazılımların üzerindeki kötü amaçlı kullanımların artması ile kendini iyiden iyiye belirgin hale getirmiştir. Bilgi güvenliğinin 3 temel özelliği vardır (*CIA*) [3]:

- Confidentiality (*Gizlilik*)
- Integrity (*Bütünlük*)
- Availability (*Kullanılabilirlik*)

Ayrıca,

- Authentication (*Kimlik Denetimi*)
- Non-Repudation (*Inkar Edememe*)

Güvenli bir yazılımın prensipieri de bu özelliklerden farklı değildir. Güvenli bir yazılım, işlediği ve ulaşabildiği verinin bütünlüğünü, gizliliğini korumalıdır. Aynı zamanda da doğru çalışmaya devam etmelidir. Güvenli bir ayzılım geliştirmek ve güvenli bir şekilde çalışmasını sağlamak açıkcası kolay bir iş değildir. Bunun için birçok yaklaşım mevcuttur. Uygulama güvenliğinde en etkin ve temel prensip, güvenlik önlemlerinin veya denetimlerin uygulama geliştirme safhasının en erken dönemlerinde gerçekleştirilmeleridir [6].



Şekil 3.1 : Uygulama geliştirme safhasındakı güvenlik aktiviteleri.

Bu resimde uygulama geliştirme safhasında gerçekleştirilebilecek güvenlik aktiviteleri gösterilmektedir.

Örneğin, henüz uygulama gereksinimlerinin belirlendiği ve kullanım senaryolarının oluşturulduğu evrelerde, uygulamanın güvenlik gereksinimleri (kimlik doğrulama vb.) ve kötüye kullanım senaryoları belirlenip, dökümante edilmelidir.

Yapılacak bir başka aktivite ise, güvenlik kod denetimidir. Güvenlik kod denetimi, geliştirme evresinde ve sonrasında kaynak kod üzerinde açıklık bulma amaçlı uygulanan otomatik veya el yordamıyla (manuel) yapılan bir yöntemdir [6]. En yaygın güvenlik denetim yöntemlerinden biri de, yazılım geliştirmede sona taklaşıldığında ve hatta bittikten sonra gerçekleştirilen güvenlik denetimleridir. Bu denetimler çalışan yazılım üzerinde çok kaba anlamı ile **fuzzing** denilen tekniklerin uygulanması ile gerçekleştirilir. Sızma testi olarak da adlandırılan bu yöntemde genellikle bilinen güvenlik problemlerini bulmaya yönelik imzalar, uygulamaya otomatik olarak gönderilir ve cevaplar analiz edilir. Bulunan bulgular doğrulandıktan sonra geliştiriciler tarafından düzeltilirler [6]. Bir yazılım hatasını, geliştirmenin geç safhalarında düzeltmenin maliyeti, erken safhalarında düzeltmeye oranla çok daha yüksektir. Dolayısıyla güvenlik bir süreç olarak algılanmalı ve fikrin ortaya atıldığı andan itibaren uygulama geliştirme yaşam döngüsü içerisinde prensiplerine dikkat edilmelidir [3].

HTTPS

HTTP protokolu şifresiz taşıma mekanizması olarak düz TCP kullanır ve bu nedenle ağ üzerinde bulunan bir saldırgan tarafından ele geçirilebilir.

HTTPS (*HyperText Transfer Protocol Secure*), WEB trafiğinde şifreleme sağlama amaçlı geliştirilmiş protokoldür. Bu ağ üzerinden aktarılan verinin güvenliğini ve bütünlüğünü korur. SSL kullanılmasına bakılmayarak HTTP istek ve cevap işlevi aynı yolla gerçekleşir [3].

HTTPS, SSL (*Secure Socket Layer*) over HTTP olarak adlandırılır.

SSL – kullanıcının tarayıcısı ve web server arasında geçiş yapan verinin bütünlüğünü ve gizliliğini koruyan mükemmel bir teknolojidir. SSL kullanan çoğu uygulamalar güvenlidir.

Çünkü, bu site 128 bit SSL kullanarak yetkisiz kullanıcıların bazı bilgilerimizi görüntülemesini önlemek için tasarlanmıştır ve bu siteyi kullanarak bilgilerimizi koruma altına alabiliriz [3]. Gerçekte ise, SSL teknolojisinin yaygınlaşmasına rağmen web uygulamalarının büyük kısmı güvensizdir (insecure).

Uygulamanın SSL sertifikasını kullanması tamamen sitenin güvenli olması anlamına gelmez. Çünkü, direkt olarak uygulamanın serverına veya istemci (client) bileşenlerine yapılan saldırılar mevcuttur ve SSL sertifikasını kullanmak bunları önlemez [6].

3.1 Web Uygulama Güvenliğinin Geleceği

İnternet üzerinde olan web uygulamaları bugün hala güvenlik açıkları ile dolu. Web uygulamalarının karşılaşmış olduğu bu güvenlik tehditlerini anlamak ve bunları adreslemenin etkili yolları hala sanayi içinde gelişmemiştir. Web uygulama güvenliğinin ayrıntıları statik değildir. SQL0 Injection gibi eski ve iyi anlaşılmış güvenlik açıkları görünmeye devam etsede yaygınlığı giderek azalmaktadır. Web uygulamaları içinde meydana gelen tüm değişikliklere rağmen "klasik" güvenlik açıklarının bazı kategorilerinde azalmaya ait bir belirti görünmüyor [3]. Web'in ilk günlerinde olduğu gibi hemen hemen aynı formda ortaya çıkmaya devam etmektedir. Bunlar iş mantığı kusurlarını ve diğer tasarım konularını içeriyor. Bu zamansız sorunların yaygın kalması muhtemeldir.

3.2 Sızma Testleri – Pentest

Sızma Testleri, başka sözle penetrasyon testi, çalışan uygulamalar üzerinde dinamik olarak yapılan güvenlik zafiyet bulma testleridir. Başka sözle, sistemin güvenliğini sınama işlemidir. Pentest sayesinde mümkün olabilecek her yolun denenerek sisteme izinsiz girilip girilemeyeceği test edilir. Tespit edilen açıklıkların ve zafiyetlerin kullanılmasıyla sistemlere sızma girişimi yapılır [3]. Burdaki amaç firma içindeki güvenlik durumunun tespit edilmesi, tespit edilen açıkların raporlanıp yetkili kişilere bildirilmesi ve bu sayede dışardan gelebilecek herhangi bir tehdite karşı gerekli önlemlerin alınmasıdır. Kısaca bu test, saldırgan bakış açısı ile gerçekleştirilirken bir çok farklı yöntem kullanılabilir [6].

Genel olarak bu yöntemler üçe ayrılır:

- Black-Box (*Zero knowledge*)
- Gray-Box (*Partial-knowledge*)
- White-Box (*Full-knowledge*)

Black Box- Bu yöntemde sızma testini gerçekleştiren firmayla herhangi bir bilgi paylaşımında bulunulmaz. Zarar vermek ya da firmadan bilgi sızdırmak amacıyla sızmaya çalışan saldırgan gibi davranılarak verilebilecek zararları görmek ve önlem almak amacıyla sistemlere girmeye çalışılır [2].

Gray Box- *White Box + Black Box* olarak tanımlayabiliriz. Sistemlerin, sistemler hakkında detaylı bilgi alınmadan test edilmesi yöntemidir. Bu test ile firma içinde kısıtlı yetkilere sahip kullanıcıların sisteme verebilecekleri zararlar tespit edilip önlemler geliştirilir [2].

White Box- Güvenlik uzmanı, firma içinde yetkili kişilerce bilgilendirilir ve firma hakkında ki tüm sistemler hakkında bilgi sahibi olur. Bu yöntemde daha önce firmada yer almış ya da hala çalışmakta olan kişilerin sistemlere verebilecekleri zararlar gözetlenir ve raporlanır [2]. Özellikle uygulanan yöntem blackbox tarafına kaydıkça elde edilen sonuçlar bir sistemin ne kadar güvenli olduğunu değil, ancak ve ancak ne kadar güvensiz olduğunu gösterebilir.

Blackbox -> Whitebox

Genellikle sızma testleri aşağıdaki gibi alt bileşenlere ayrılmaktadır:

- Web uygulama sızma testleri
- Son kullanıcı ve sosyal mühendislik testleri
- DDoS ve performans testleri
- Ağ altyapısı sızma testleri
- Yerel ağ sızma testleri
- Mobil uygulama güvenlik testleri

Sızma testlerinin önemli metodolojilerine vardır. Bunlara örnek olarak, *OWASP* (*Open Web and Application Security Project*), *NIST* (*National Institute of Standards and Technology)*, OSSTMM (*Open Source Security Testing Methodology Manual*), *ISAF (International Security Assistance Force)* gösterebiliriz [2].

NIST'ın analizine göre istismar açıklarının %92'si yazılımdan kaynaklanmaktadır [7].

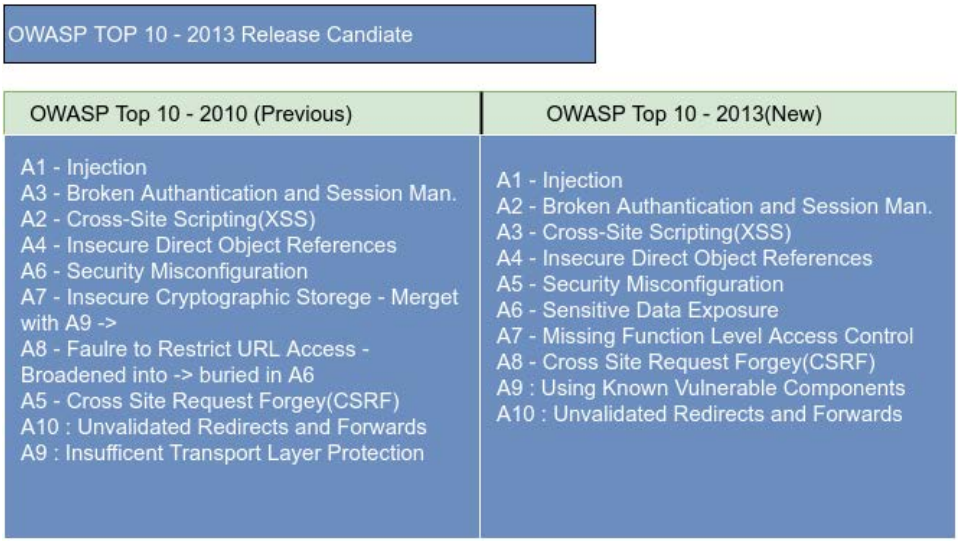
OWASP'ın analizine göre ise güvenlik açıklarının %70 den fazlası uygulama katmanında mevcuttur, ağ katmanında değil [2].

3.2.1 Neden pentest?

Uygulamamızda ki/Sistemimizde ki güvenlik açıklarını gerçek saldırganlar bulmadan ve bunları kötüye kullanmadan önce bulmalı ve bunları kapatmalıyız. Günümüzde nasıl geliştirilen uygulamalar için fonksiyonel testler muhakkak gerçekleştiriliyorsa gerekli güvenlik testlerinin de yapılması artık kaçınılmazdır. Pentest sürecinde gerçek saldırganların kullandığı araçlar ve yöntemler kullanılır/kullanılmalıdır. Pentest sonucu bulunan açıkların kapatılması takip edilmeli ve bu her organizasyonda bir süreç olarak tanımlanmalıdır. [7] Yanlış algılama: "Önce uygulamayı geliştirelim, en sonunda güvenli hale getiririz." Bu yaklaşım sisteme maliyet, zaman ve imaj kaybı olarak geri dönmektedir [6]. Maliyet ve zaman bakımından, herhangi bir açığı sistem geliştirildikten sonra gidermek 2-3 kat daha fazla maliyete ve zamana neden olmaktadır. İmaj bakımından ise kullanıcılara güven vermeyen bir sistem izlenimi oluşturur. Güvenlik açıklarının gerçek saldırganlar tarafından kötüye kullanılmasının hukuki yaptırımları / cezaları vardır.

3.3 OWASP TOP 10

OWASP TOP 10 web uygulamaları için en kritik 10 riski ve gerekli güvenlik kontrollerini listeleyen bir projedir [2].



Şekil 3.2 : OWASP TOP 10

3.4 Ana Savunma Mekanizmaları

Web uygulamalarının temel güvenlik sorunları – bütün kullanıcı girdileri güvenilmezdir – birçok güvenlik mekanizmaları uygulamaların kendilerini korumaları için kullanılır. Aslında tüm uygulamalar kavramsal olarak benzer mekanizmaları kullanırlar [3].

Web uygulamaları tarafından kullanılan koruma mekanizmaları aşağıdaki temel unsurları içermektedir:

- Uygulama verisine kullanıcı erişimini ele almak ve yetkisiz erişim sağlanmasını engellemek;
- Istenmeyen davranışa neden olan hatalı girdiyi önlemek için kullanıcı girdisini ele almak;
- Doğrudan hedef zamanı uygulamanın uygun bir şekilde davranması, uygun savunma mekanizması alması ve saldırganın saldırılarını boşa çıkarmak için saldırganları ele almak;

- Yöneticilere izin vererek uygulamanın aktivitelerini ve fonksiyonelliğini yönetmek;

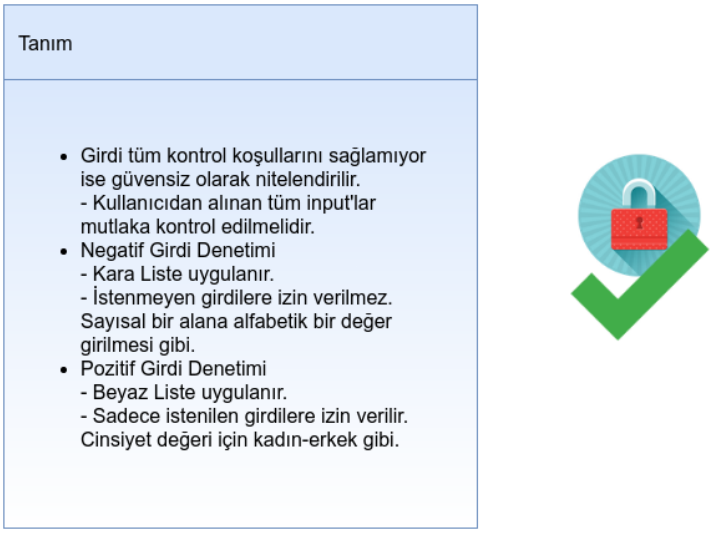
3.4.1 Kullanıcı erişimini ele almak

Web uygulamalarının çoğu, ilişkili güvenlik mekanizması üçlüsünü kullanarak erişimi ele almaktadır:

- Kimlik Doğrulama (*Authentication*)
- Oturum Yönetimi (*Session Management*)
- Erişim Kontrolü (*Access Control*)

Bu mekanizmaların herbiri uygulamanın genel duruşu için temel adımlardan biridir. Onların bağımlılığı yüzünden, genel güvenlik bu mekanizmalar tarafından sağlanır [3]. Uygulamanın herhangi tek bir bileşeninde olan kusur saldırganın uygulamanın işlevselliği ve verilerine sınırsız erişim sağlamasına olanak tanıyabilir.

3.4.1.1 Kullanıcı girdisini ele almak (Girdi / Çıktı Denetimi)



Şekil 3.3 : Girdi / Çıktı Denetimi

Web uygulamalarında genelde karşılaşılan problemlerden biri de veri ile zararlı kodun bir birine karışmasıdır. Tüm kullanıcı girdileri güvenilmezdir [8]. Web uygulamalarına karşı saldırıların büyük çoğunluğu beklenmedik girdinin gönderilmesini kapsamaktadır. Buna bağlı olarak, uygulamanın güvenliğini savunmak için önemli gereksinimlerden birisi de kullanıcı girdisini ele almaktır.

Girdi tabanlı güvenlik açıkları uygulamanın işlevselliğinin herhangi bir yerinde ortaya çıkabilir [9]. "Girdi Doğrulama" (Input Validation) bu saldırılara karşı gerekli savunma olarak anılmaktadır. Girdi denetimi sunucu tarafında yapılmalıdır, istemci tarafında yapılan denetimler saldırganlar tarafından atlanabilmektedir.

3.4.1.2 Girdi çeşitleri

Tipik bir web uygulaması kullanıcı tarafından sağlanan verileri farklı şekillerde işler. Girdi Doğrulamasının bazı türleri girdinin tüm bu şekilleri için uygun ya da uygun olmaya blir. Aşağıdaki resimde kullanıcı kayıt fonksiyonu tarafından sık sık uygulanan girdi doğrulama türü gösterilmistir [3].

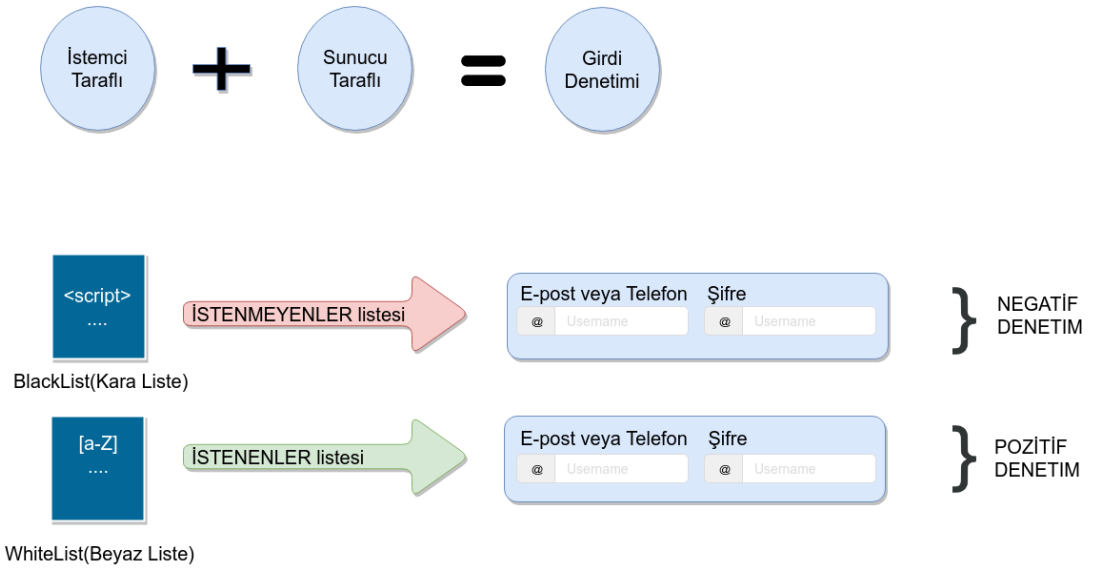
Şekil 3.4 : Tipik bir login işlemi

Bir çok durumda, uygulama özel madde veya girdiler üzerinde sıkı doğrulama kontrolleri uygulayabilir. Örneğin, login fonksiyonuna gönderilen kullanıcı adının maksimum 8 karakter uzunluğunda olması ve yalnızca alfabetik karakterler içermesi talep olunabilir. Diğer durumlarda uygulama, girdinin geniş aralığına tahammül gösterebilir [8].

Örneğin; Adres alanı için kişisel ayrıntı sayfasına gönderilen girdi harfler, rakamlar, boşluklar, tireler, apostroflar, ve farklı karakterler içerebilir. Ancak, bu madde için, kısıtlamalar hala zorunlu tutulabilir [3]. Veri, makul olan uzunluk sınırını aşmamalı (50 karakter gibi) ve bazı HTML tagları içermemelidir. Bazı durumlarda, uygulama kullanıcı tarafından keyfi girişleri kabul etmeye ihtiyac duya bilir.

Örneğin, blogging uygulamasının kullanıcısı konusu “web application hacking” olan bir blog oluştura bilir. Bloga yapılan postlar ve yorumlar meşru saldırı stringlerinden ibaret olabilir. Bu uygulama bu girdini veritabanında tutmaya, diske yazmaya ve güvenli bir şekilde kullanıcılara geri göstermeye ihtiyaç duyabilir [6]. Kullanıcıların tarayıcı arayüzünü kullanarak girmiş olduğu farklı girdi çeşitlerine ilave olarak, tipik bir uygulama sunucuda başlayan ve istemciye gönderilen, daha sonra sonraki isteklerde istemciden sunucuya iletilen çok sayıda veri kabul ediyor. Bu maddeleri içeren cookie gibi öğeler, uygulamanın sıradan kullanıcıları tarafından görünmez ancak bir saldırgan tarafından görüle bilir ve değiştirilebilir [9].

3.4.1.3 Girdi işleme yaklaşımları



Şekil 3.5 : Girdi işleme yaklaşımı

Farklı durumlar, farklı girdi türleri için farklı yaklaşımlar tercih edilebilir.

Negatif girdi yaklaşımı

Negatif liste yaklaşımında bir girdinin hangi karakterler ya da sabit değerler içeremeyeceği tanımlanır. Bu doğrulama mekanizması negatif listeye uygun olan ve başka şeylere izin veren bazı verileri bloke eder [9]. Ayrıca “olumsuz” veya “karaliste” doğrulaması olarak bilinen bu strateji, olumlu doğrulamaya zayıf bir alternatifdir. Negatif liste tanımlarını es geçmek mümkün olabildiği için pozitif liste yaklaşımı tercih edilmelidir [11].

Aslında, %3f veya Javascript veya benzeri gibi karakterleri görmeyi beklemek istemiyorsak, bu karakterler içeren stringleri reddetmeliyiz. Bu tehlikeli bir

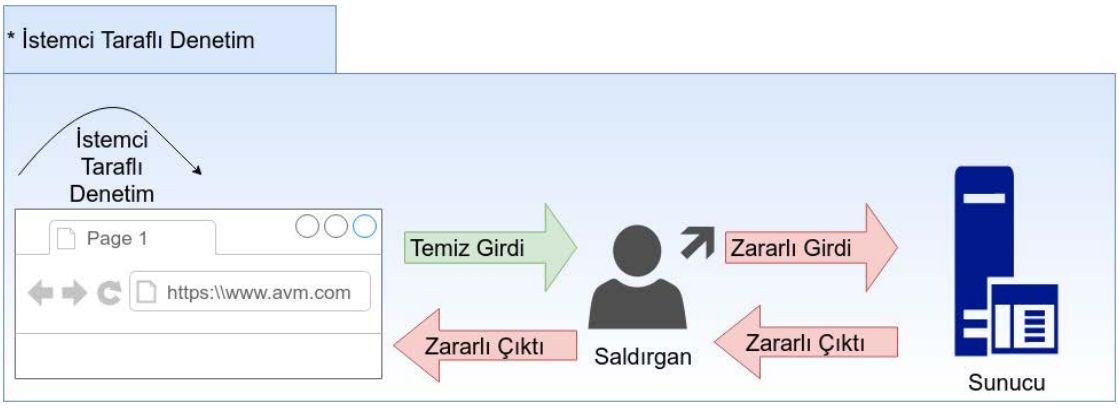
stratejidir, çünkü mümkün olan kötü veri seti potansiyel olarak sonsuzdur [3]. Bu stratejiyi kabul etmek, sonsuza dek “known bad” karakterler ve patternler listesini sağlamıyor olacağız. Birçok kara tabanlı filtreler girdiye önemsiz ayarlamalar yaparak bloke edilip kolaylıkla atlatılabilir (bypass) [3].

Örneğin,

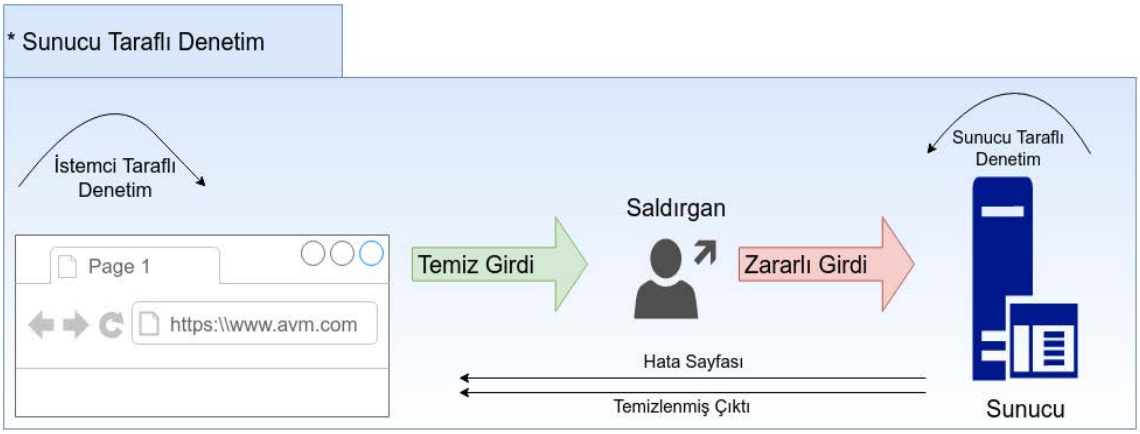
- SELECT bloke edilmiş ise, SeLeCt çalıştırılır.
- or 1=1 bloke edilmiş ise, or 2=2 çalıştırılır.
- alert ('xss') bloke edilmiş ise, prompt ('xss') çalıştırılır.
- ‘→ %27
- <script> → <sc<script>ript> ya da <ScRiPt>

Pozitif girdi yaklaşımı

Pozitif liste yaklaşımında bir girdinin hangi karakterler ya da sabit değerler içerebileceği tanımlanır. Bu doğrulama mekanizması veriyi pozitif liste ile eşleşmeye ve diğer heryerde bloke etmeye izin verir [3]. Örneğin, veritabanında talep edilen ürün kodunu aramadan önce, uygulama onun sadece alfanumerik karakterlerden oluşmasını ve tam olarak 6 karakter uzunluğunda olduğunu doğrulamalıdır [8]. Ürün kodu üzerinde yapılacak olan sonrakı işlem göz önüne alındığında, developerler bu testi geçen girdinin muhtemelen herhangi bir soruna sebep olmayacağını biliyordur [9].



Şekil 3.6 : İstemci taraflı denetim



Şekil 3.7 : Sunucu taraflı denetim

Bu yaklaşımın uygulanabildiği durumlarda, en etkili yol potansiyel kötü niyetli girdiyi ele almaktır. Bazı durumlarda uygulama "iyi" olarak bilinen bazı makul kriterlere uymayan veriyi işlemek için kabul etmelidir. Örneğin, bazı insanların ismi apostrof veya tireden ibaretdir. Bunlar veritabanına karşı saldırılarda kullanılabilir. Whitelist tabanlı yaklaşım kullanıcı girdisinin ele alınmasındaki soruna çok amaçlı bir çözüm getirmemektedir [3]. Girdi denetiminde negatif liste (*black-list*) yerine pozitif liste (*white-list*) yöntemi tercih edilmelidir. Girdi denetiminde regular expression'lardan yararlanılmalıdır [12].

Girdi ile alakalı şu özellikler denetlenmelidir:

- Tip: İnteger, String, boolean vb. → Casting
- Min-Max değerler
- Uzunluk
- İçerebileceği karakterler: sadece rakam vs. → Regular expression (^[a-zA-Z0-9.-] {1-100} $) Min uzunluk- 1, max uzunluk-100
- Sabit format: email, tarih, plaka, posta kodu vs. → Regular expression
- Sabit değerler: ülkeler, şehirler, vs. → Pozitif liste (^ (Adana|Adıyaman|...|Zonguldak) $)

SQL Injection ve XSS (*Cross Site Scripting*)'e karşı da girdi denetimi tavsiye edilir, ancak ana güvenlik kontrolü çıktı kodlama (*output escaping)* işleminden geçirmektir.

3.4.2.1 Canonicalization (Doğallaştırma/Normalleştirme)

Saldırganlar, girdi üzerinde farklı kodlama teknikleri (*ASCII, Hexadecimal, UTF-8, Unicode, URL Encoding, vs.*) uygulayarak girdi denetimi filtrelerini atlamaya çalışırlar.

- Orjinal: <
- Farklı kodlama &lt (html), %3c

Doğallaştırma ile farklı yöntemlerle kodlanmış girdiler standart bir formata dönüştürülür ve bundan sonra girdi denetimi işlemine hazır hale gelir.

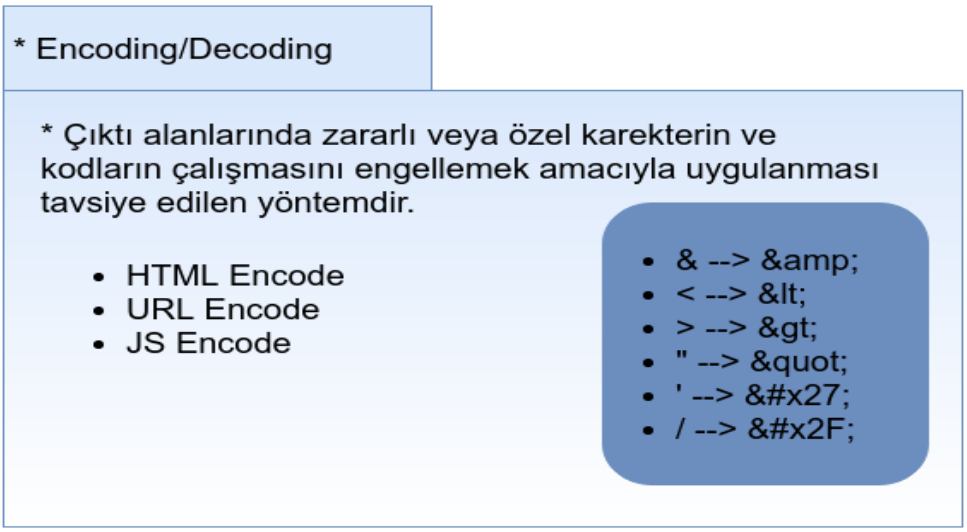
ESAPI Encoder ile doğallaştırma işlemi gerçekleştirilebilir [10]:

- String clean = ESAPI.encoder(). canonicalize (request. getParameter("input"));

ESAPI Validator, denetleme işleminden önce canonicalize() metodunu otomatik çağırır:

- String name=ESAPI.validator(). isValidInput ("test", request. getParameter("input"), "username", 20, false);

3.4.2.2 Çıktı denetimi

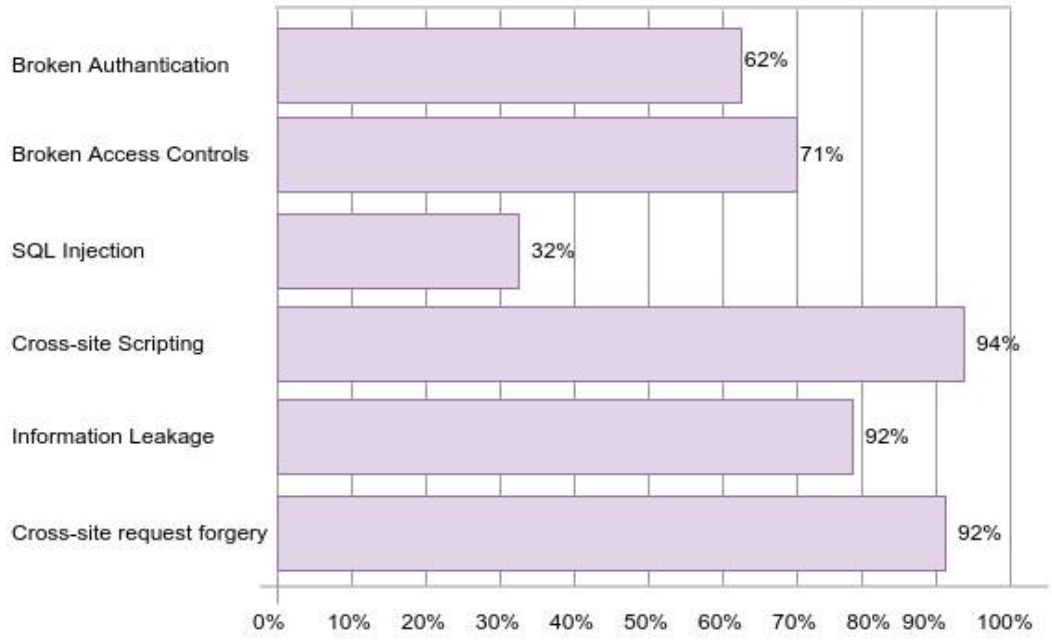


Şekil 3.8 : Çıktı denetimi

Web uygulamalarında, sadece girdi alanları değil çıktı alanları da problem oluşturabilirler. Bazı durumlarda uygulama koduna karıştırılan zararlı kodlar çıktı olarak istemciye gönderilebilir. Böyle bir durumda istemci dolaylı olarak bu zafiyetden etkilenir [6].

3.4.2 Web yazılım güvenliğindeki mevcut problemler

Web Yazılım güvenliği - kötü niyetli saldırı (hacker) risklerine karşı yazılımı korumak için uygulanan bir fikirdir. Eğer yazılımın güvenliği doğru bir şekilde sağlanmış ise potansiyel riskler altında bile doğru çalışmaya devam eder [3]. Güvenli yazılım; işlediği ve ulaşabildiği verinin bütünlüğünü, gizliliğini korumalıdır. Aynı zamanda bilgiye erişimin devamlılığını da sağlamalı, yani doğru çalışmaya devam etmelidir. Web yazılımlarında güvenlik sorunlarının büyük çoğunluğu uygulamadan kaynaklanıyor. Saldırıların %75'i uygulama katmanında gerçekleştiriliyor [7]. Yazılım geliştiricilerin çoğu, geliştirdikleri yazılımın güvenli olduğundan emin değildir(Microsoft). Daha doğrusu geliştirdikleri yazılımları nasıl güvenli şekilde geliştireceği konusunda bilgi sahibi değildirler. Bu bölümde web yazılım güvenliğide mevcut problemlere olan yaklaşımları ve onlara karşı çözüm yollarını öğreniyor olacağız [12].



Şekil 3.9 : Güncel zafiyetlerin oranı

Bu resimde 2007 ve 2011 yılı sırasında 100'den çok web uygulama üzerinde test yapılarak uygulamaların güvenlik açıkları % ile test edilerek gösterilmiştir [3].

- Broken Authentication (Kırılmış Kimlik) (62%)

Bu kategorideki güvenlik açıkları uygulamanın login mekanizmasındakı farklı kusurları yani saldırganın zayıf şifreleri tahmin etmesine olanak verecek, kaba kuvvet saldırısı başlatabilecek veya giriş atlayabilecek kısımlarını kapsamaktadır [3].

- Broken Access Controls (Kırılmış Erişim Kontrolu) (71%)

Bu kategorideki güvenlik açıkları saldırganın sunucu üzerinde tutulan kullanıcıya ait hassas verileri görüntülemesine veya ayrıcalıklı eylemleri gerçekleştirmesi durumlarını kapsamaktadır [3].

- SQL Injection (SQL Enjeksiyonu) (32%)

SQL Injection, veri odaklı uygulamalarda SQL dili özelliklerinden faydalanılarak standart uygulama ekranındaki ilgili alana ek SQL ifadelerini ekleyerek yapılan bir tür atak tekniğidir [9].

- XSS- Cross-Site Scripting (Siteler Arası Betik Çalıştırma) (94%)

Bu kategorideki güvenlik açıkları HTML, CSS, Javascript vb. ile hazırlanmış kod parçaçıklarının hedef kullanıcının tarayıcısında izinsiz olarak çalıştırmasından kaynaklanmaktadır [8].

- Information leakage (Bilgi Kaçağı) (78%)

Bu kategorideki güvenlik açıkları sistem verisinin veya hata ayıklama bilgisinin (debugging) loglama fonksiyonu veya herhangi bir çıkış akımı ile programdan çıkmasından kaynaklanmaktadır [8].

- CSRF- Cross-Site Request Forgery (Siteler Arası Istek Sahteciliği) (92%)

Bu kategorideki güvenlik açıkları uygulamaya gelen her isteğin, gerçekten kendi uygulaması üzerinden geldiğinin farkına varılamamasından veya bu kontrolün yapılmamasından kaynaklanmaktadır [12].

3.4.3 Mevcut problemlere olan yaklaşımlar

3.4.3.1 Kimlik doğrulama (Authentication)

Kimlik doğrulama mekanizması kullanıcı erişiminin ele alınmasındakı en temel başlıklardan birisidir. Yetkisiz erişimlerin sağlanmaması gereken her ortamda *kimlik doğrulama* işlemlerine ihtiyaç duyulur [3]. Dolayısıyla web uygulamalarını

düşünerek, bir kimlik doğrulama tanımı yapacak olursak; uygulamayı kullanacak olan kullanıcıların geçerli/tanımlı bir kullanıcı olup olmadığını kontrol edilmesi işlemine verilen isimdir diyebiliriz.

Kimlik Doğrulama işlemleri her tür uygulamada kritik seviyede önemlidir! [13] Bugünün web uygulamalarının çoğu, geleneksel kimlik doğrulama modelini kullanıyor yani kullanıcı doğruluğu kontrol etmek için uygulamaya kullanıcı adı ve şifresini gönderiyor. Aşağıdaki resim tipik oturum açma (login) fonksiyonudur [7].

Şekil 3.10 : Klasik login işlemi

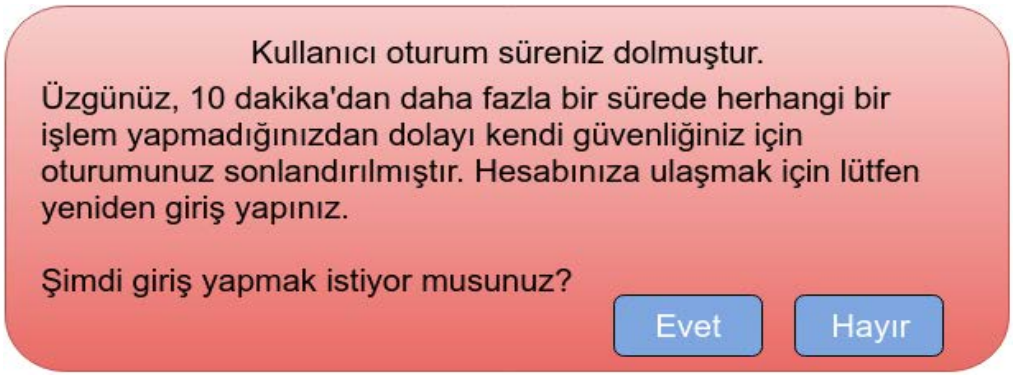
Online olarak bankalar tarafından kullanılan güvenlik açısından kritik olan uygulamalarda, bu basit model ek kimlik bilgileri ve bir çok aşamalı oturum açma işlemi tarafından uygulanır. Güvenlik ihtiyaçları daha yüksek olduğu durumlarda, istemci sertifikaları, smartkartlar (akıllı kartlar) vb. gibi işaretlere dayalı başka kimlik doğrulama modelleri kullanılabilir [12]. Ana kimlik doğrulama işlemi sürecine ek olarak, kimlik doğrulama mekanizmaları genellikle hesap kurtarma (account recovery), şifre değiştirme vb. gibi bir dizi destekleyici fonksiyonellikler kullanmaktadır [3]. Oturum açma mekanizmasında ki ortak olan sorunlar genellikle mantıkda ki hatalardan kaynaklanabilir. Bunlara, saldırganın başka kullanıcının kullanıcı adını tespit etmesi, şifrelerini tahmin etmesi veya oturum açma mekanizmasını atlayarak uygulamaya giriş yapmasını gösterebiliriz [3].

3.4.3.2 Oturum yönetimi (Session Management)

Kullanıcı erişiminin ele alınmasındaki diğer bir madde ise yetkili kullanıcının oturumunu yönetmektir. Web uygulamaları HTTP üzerinden haberleşirler. Fakat HTTP, istemcinin kimlik bilgisini doğruladıktan sonra istemcinin durum bilgisini (oturum bilgisini) tutmaz. Yani bir kullanıcıdan gelen iki isteğin aynı kullanıcıdan geldiğini anlayamaz [13]. Bu nedenle istemcinin durum bilgisinin kontrol edilmesi

için her istemciye özel bir değer sunucu veya geliştirici tarafından üretilir (Session ID). SESSION ID, benzersiz bir stringdir [3]. Kullanıcı bu değeri kabul ettiğinde, tarayıcı otomatik olarak her yeni HTTP isteğinde onu sunucuya geri yollar ve uygulama isteği sağlayan kullanıcıyı tanır [6].

Oturum bilgisi URL'de, form gizli alanlarında veya en yaygın olan çerezlerde (cookiler'de) taşınabilir. Eğer kullanıcı belirli bir zaman içinde istekde bulunmamışsa, aşağıdakı resimde gösterildiği gibi oturumun süresi biter;



Şekil 3.11 : Oturumun zaman aşımı

Oturum yönetimi zafiyetlerine aşağıda belirttiğimiz maddeleri gösterebiliriz:

- Rasgele oluşturulmamış, tahmin edilebilir oturum ID'si (session identifier);
- Logout işleminden sonra oturum ID'sinin geçersizleştirilmemesi ya da uygulamada logout özelliğinin sağlanmaması;
- Kullanıcı, belirli bir süre inaktif olduktan sonra oturum ID'sinin geçersizleştirilmemesi.
- Oturum ID'sinin maksimum yaşam süresinin belirlenmemesi;
- Uygulamanın birden fazla oturum ID'si içermesi ve uygulamanın bunları nasıl kullandığının belirli olmaması;
- Oturum ID'sinin başarılı kimlik denetimi sonrası yenilenmemesi (Session Fixation);
- Oturum ID'sinin URL içine gömülmesi;
- Oturum ID'sinin güvensiz kanallar üzerinden taşınması;
- Oturum ID'si içeren cookie'lerin httpOnly ve secure parametrelerinin aktifleştirilmemesi;

Oturum ID'si günümüz şartlarında kaba-kuvvet saldırılarına karşı en az 128-bitlik güvenlik sağlamalıdır. Oturum ID'sinin iletimi esnasındaki zafiyetleri önlemek için;

- HTTP yerine HTTPS kullanılmalıdır;
- Secure parametresi oturum ID'sini taşıyan çerezler için aktifleştirilmelidir. Bu sayede oturum ID'si güvenli olmayan kanallar üzerinden gönderilmeyecektir;
- Firesheep;

Firesheep, Winpcap kütüphanesi ile birlikte kullanılan ve aynı ağ üzerindeki cookileri yakalayan Firefox üzerindeki en etkili araçlardan biridir. Bu aracın yakaladığı çerez bilgileri ile facebook, twitter vb. Gibi birçok hesap ele geçirilebilir. Bu yüzden de ortak ağlara bağlanılırken dikkat edilmelidir [3].

XSS ***(****Cross-Site Scripting****)*** açığını içeren uygulamalarda js kodları çalıştırarak kullanıcıların oturum ID'lerini ele geçirmek ve bu sayede kullanıcıların hesaplarına erişmek mümkündür. (Session Hijacking) [3]

Şekil 3.12 : Oturum sabitinin gösterimi

Ekstra oturum yönetimi kontrollerine;

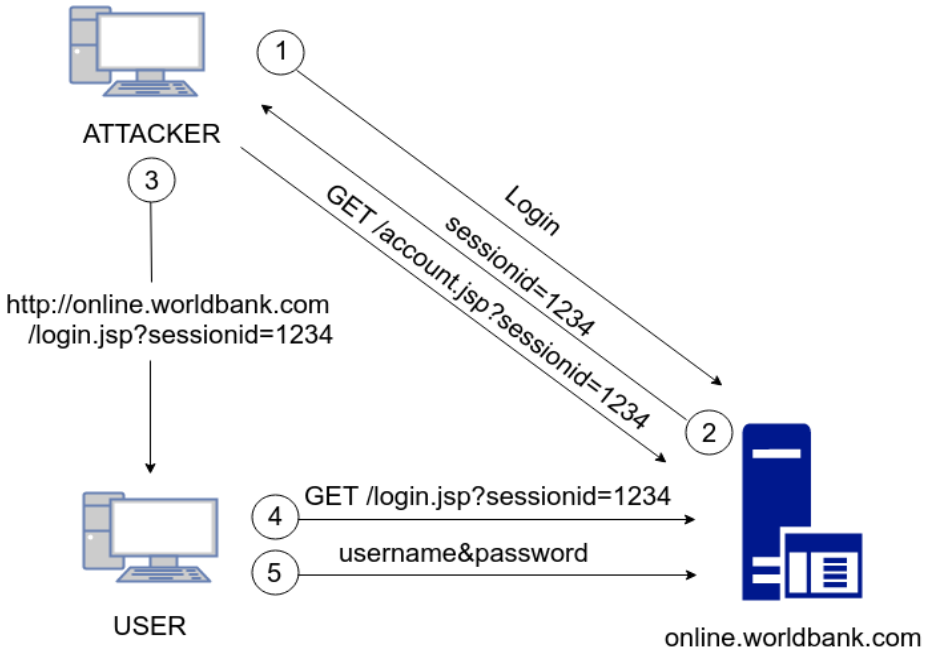
- Oturum ID'si geçersizleştirme (logout, inaktiflik, maksimum süre)
- httpOnly parametresini aktifleştirme (isSession = true ise)

gösterebiliriz.

Saldırı açısından, oturum yönetimi mekanizması simgelerin güvenliğine son derece bağımlıdır. Saldırıların büyük çoğunluğu kullanıcılara verilen simgelerin aranıp bulunarak uzlaşmasına dayanmaktadır. Bunun mümkün olduğu durumlarda, saldırgan, kendisini kurban kullanıcı olarak taklit ederek uygulamaya sızıp sanki o kullanıcıymış gibi uygulamayı kullanabilir.

Zafiyetin başlıca alanları simgelerin nasıl üretilmesinden, saldırganın başka kullanıcılara verilen simgeleri tahmin edebilmesinden, ve simgelerin sonradan nasıl ele alınmasındakı kusurlardan kaynaklanmaktadır. Bu durum Siteler arası istek sahteciliği saldırısına yol açmaktadır [15].

Bunu en iyi şekilde aşağıdaki gibi gösterebiliriz:



Şekil 3.13 : Oturum sabitinin çalınmasına yönelik senaryo

Yukarıdaki senaryonu açıklamak gerekirse;

- Saldırgan, web uygulamasından kendisine gelen (kimlik denetiminden önce) oturum ID'sini ilgili linkle birlikte kurbana yönlendirerek onun bu linki ve oturum ID'sini kullanarak login olmasını sağlar.

- Şayet, web uygulaması, kimlik denetimini gerçekleştirdikten sonra oturum ID'sini yenilemez ise (session fixation) kurbanın oturumu tehlikeye girer. Zira bu oturum ID'si saldırgan tarafından da bilinmektedir.

  Çözümü: Her başarılı kimlik denetimi sonrası yeni bir oturum ID'si oluşturulmalı ve atanmalıdır [15].

  Oturum değişkenlerini denetlerken dikkat edilmesi gereken başlıca konulara aşağıdakileri gösterebiliriz;

- Oturum değişkenlerinin nasıl taşındığı (GET, POST, hidden field)
- Oturum değişkenlerinin HTTPS üzerinden aktarılması
- HTTP'nin yerine HTTPS'e yönlendirilmesi

  Cache-control: Header bilgisi ile gönderilen Cache-control mekanizması tarayıcının dosyaları ön bellekte tutup tutmamasını eğer tutacaksa ne kadar süre ile tutacağını belirtmek içindir. Bu mekanizma firebug, liveHeaders gibi eklentilerle test edilebilir [3].

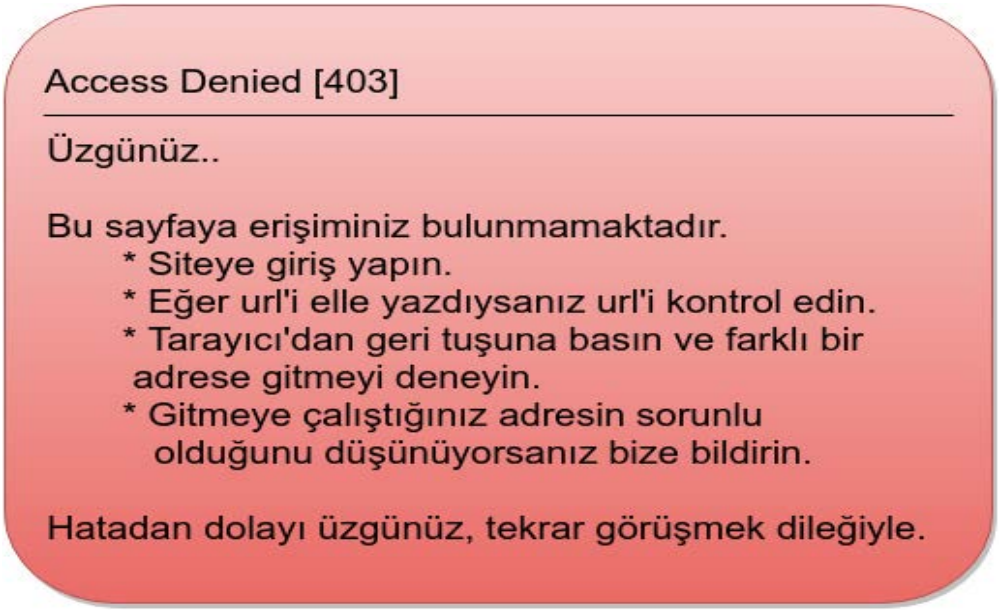
Çizelge 3.1 : Önbellek değişkenleri ve açıklamaları

Önbellek değişkeni	Önbellek değişken açıklaması
Max-age = [seconds]	Kaynağın belirtilen süre kadar önbelleğe alınacağını ve daha sonra tekrar kullanıla bileceğini belirtir. Cookie Expires özelliğine benzemektedir.
no-cache	HTTPS üzerinden kullanıldığında önbelleğe almayı engeller. Fakat HTTP üzerinden kullanıldığında Expires :-1 gibi değerlendirilip kaynağı, süresi hemen dolacak olarak önbelleğe alır.
no-store	Kaynağı hiçbirzaman önbelleğe almaz. Kaynak için her defasında request yapıldığında sunucuya istek gider ve yanıt döner.
public	Kaynağın herhangi bir önbellek tarafından önbelleğe alınacağını gösterir.
private	Kaynağın tek bir kullanıcı için tasarlandığını ve genel önbellek tarafından paylaşılmamasını belirtir. Özel bir önbellek tarafından (örn, proxy) önbelleğe alınabilir.

Şekil 3.14 : Cache kontrolu eklenmesi

3.4.3.3 Erişim kontrolu (Access Control)

Kullanıcı erişiminin ele alınması sürecindeki sonuncu mantıksal adım her isteğin izin verilip veya red edilmesi ile ilgili doğru kararlar alınması ile alakalıdır. Eğer bu mekanizmalar doğru çalışırsa, uygulama her istekte bulunan kullanıcının kimliğini tanır. Istekte bulunan kullanıcının eylemi gerçekleştirmek veya veriye erişmek için yetkisinin olup olmadığı ile ilgili görsel gösterilmiştir.



Şekil 3.15 : Erişimin reddedilmesi

3.4.3.4 Kimlik doğrulamasını korumak (Securing Authentication)

Bazı durumlarda “daha” güvenli aslında tersi olabilir. Örneğin, kullanıcıları çok uzun şifreler belirlemeye ve sık sık onları değiştirmeğe zorlamak kullanıcıların şifrelerini bir yere kaydetmelerine neden olmaktadır.

Güçlü kimlik bilgileri kullanmak;

- Uygun olan minimum parola kalite gereksinimleri uygulanmalıdır. Bu gereksinimler; minimum uzunluk; alfabetik ve sayısal görünüm, büyük ve küçük karakterlerin görünümü, sözlük kelimelerinden, isimlerden ve ortak kullanılan şifrelerden kaçınmak, şifrelerin kullanıcı adı ile başlamasını engellemek, daha önce ayarlanmış parolaların benzerliği veya eşleşmesinin önlenmesi ile ilişkin kuralları içerebilir [14]. Farklı kullanıcı kategorilerine göre gerekli olan minimum şifre kalite gereksinimleri değişiklik göstere bilir.
- Kullanıcı adları tekil olmalıdır.
- Kullanıcılara yeteri kadar güçlü parola ayarlamaya uygulama izin vermelidir. Örneğin; uzun parolalara ve geniş aralıktaki karakterlere izin verilmelidir.

Kimlik bilgilerinin gizliliğini yönetmek;

- Tüm kimlik bilgileri yetkisi olmayan kullanıcılara ifşa edilmeden belirli bir şekilde oluşturulmalı, saklanmalı ve taşınmalıdır.
- Tüm istemci/sunucu iletişimleri SSL gibi kriptolojik teknolojiler ile korunmalıdır.
- Uygulama içerisinde kimlik doğrulamaya gerek olmayan alanlar HTTP ile çalışa bilir, ancak kimlik doğrulama ihtiyacı olan alanların HTTPS olarak yüklenmesi gerekmektedir [12].
- Kimlik bilgilerinin sunucuya gönderilmesinde POST istekleri kullanılmalıdır. Kimlik bilgileri URL'de veya cookie içerisinde tutulmamalıdır [6].
- Sunucu tarafında şifrelenen kullanıcı bilgilerinin saldırgan tarafından ele geçirilmiş olsa bile orijinal değerine yeniden çevrilmesine izin verilmemelidir.
- İstemci taraflı "beni hatırla" özelliği sadece kullanıcı adı gibi gizli olmayan kısımları hatırlamalıdır [6].
- Şifre değiştirme özelliği uygulanmalı ve kullanıcı belirli aralıklarla buna zorlanmalıdır.

Kimlik bilgilerini uygun bir şekilde doğrulamak;

- Şifreler harfe duyarlı bir şekilde, filtrlenmeden veya herhangi bir karakterleri değiştirmeden, ve şifre kırpmadan eksiksiz olarak doğrulanmalıdır.

- Kimlik doğrulamadaki mantık hatalarını belirlemek için hem pseudocode hem de uygulamanın kaynak kodu yakından incelenmelidir.
- Kimlik doğrulama kısmında rasgele bir doğrulama sorusu sorulmalıdır. Örneğin, en sevdiğin yemek nedir? Bu sayede saldırganın bu soruya doğru bir şekilde cevap verilmesi engellenir.
- Çok katmanlı sistemlerde her katmanda farklı bir doğrulama sorusu sorulmalıdır.
- Kullanıcıya sorulan doğrulama sorusuna verilen cevap kullanıcının profilinde saklanmalı ve her kimlik doğrulama işlemlerinde kullanıcı doğru cevap verene kadar kimlik doğrulama yapılmamalıdır [6].
- Kullanıcının doğrulama sorusuna verdiği cevap HTML hidden input olarak tutulmamalı ve her defasında doğruluğu sunucuya sorulmalıdır.

Bilgi sızmasını önlemek;

- Uygulama tarafından kullanılan farklı kimlik doğrulama mekanizmaları kimlik doğrulama ile alakalı bazı bilgileri ifşa etmemelidir. Saldırgan hangi işlemin kimlik doğrulamasında başarısız olduğunu bilmemelidir.
- Kimlik doğrulamasının başarısız olduğu genel bir mesaj ile bildirilmelidir. Ayrıca hata durumları sunucudan dönen farklı HTTP durum kodları ile bildirilmelidir.
- Tekrarlı olarak aynı tarayıcı üzerindeki kimlik doğrulama işlemlerinin başarısız olduğu durumlarda belirli bir süre için kullanıcı askıya alınır, genel bir mesaj ile bilgi verilerek kimlik doğrulama işleminin başarısız olduğuna neden olan kısım sızdırılmadan kullanıcı belirli bir süre askıya alınıp, daha sonra bu işlemi tekrar denemesi istenilir. Eğer farklı tarayıcı ile aynı anda başarısız istekler geliyor ise bu saldırı olduğuna işaret eder [3].

Şifre değiştirme özelliğinin yanlış kullanılmasını önlemek;

- Şifre değiştirme özelliği belirli aralıklarla şifrelerin süresinin dolmasını sağlayarak şifrenin değiştirilmesini ve herhangi bir sebepten dolayı kullanıcın şifresini değiştirmek istediğinde bu özelliği sağlamalıdır [3].

- Bu özellik sadece kimliği doğrulanmış kişiler tarafından yapılması sağlanmalıdır.
- Kullanıcı adı açık yada gizli form alanında veya cokie içerisinde belirtilmemelidir. Nedeni ise farklı kişilerin farklı kullanıcıların şifrelerini değiştirmesini engellemektir.
- Yeni girilmiş şifre kullanıcın emin olması için iki defa girilmelidir.
- Kullanıcılara şifrelerinin değiştiği ile ilgili mail gönderilmeli ama bu mailin içeriği eski veya yeni kimlik bilgilerini içermemelidir.

  Loglamak, İzlemek ve Bildirmek;

- Uygulama, kimlik doğrulama ile ilgili durumları (login, logout, şifre değiştirme, hesabın askıya alınması ve hesap kurtarma) loglamalıdır (kayıt altına almalıdır). Bundan başka başarılı ve başarısız denemeler de loglanmalıdır. Loglar söz konusu bilgileri (kullanıcı adı ve IP adresi gibi) içermelidir, ama parola gibi güvenlik sırlarını içermemelidir. Loglar yetkisiz erişimlerden fazlasıyla korunmalıdır, çünkü loglar bilgi sızması için kritik kaynaklardır [3].
- Kimlik doğrulamadaki anomali durumlar uygulama tarafından işlenmeli ve saldırı için önlemler alınmalıdır. Örneğin; uygulama yöneticileri kaba kuvvet saldırıları gösteren durumların farkına varmalı ve bunlar için uygun savunma tedbirleri almalılar.
- Kullanıcılar herhangi bir kritik güvenlik durumlarında haberdar edilmeliler. Örneğin; şifre değiştiğinde uygulama kullanıcın kayıtlı olan email adresine mesaj göndermelidir [3].
- Kullanıcılar sık sık oluşan güvenlik durumlarında da haberdar edilmelidir. Örneğin; başarılı bir giriş yapıldıktan sonra, uygulama son girişin zamanını ve IP adresini ve o zamana kadar yapılmış geçersiz login işlemlerinin sayısını kullanıcıya bildirmelidir.

3.4.3.5 Güvenli soket katmanı (SSL)

Internet üzerinde yapılan bilgi alışverişinde iki temel güvenlik açığı doğmaktadır:

1. Gerçekten doğru bilgisayar veya sunucuya bağlanıp bağlanmadığımızı bilememek; bankacılık işlemleri için gerçekten bankanın kendi web sitesine mi bağlanılmaktadır?

2. İnternet üzerinde paylaşacağınız bilgilerin muhatabınız dışındaki kişilerce ele geçirilip geçirilmediğini bilememek.

Acaba girdiğiniz kullanıcı numarası ve internet bankacılığı şifresi kendi bankanıza giderken aynı zamanda da kötü niyetli kişilerin eline ulaşıyor mu?

SSL *(Secure Socket Layer)*, ağ üzerinden iletişim kuran client ve server arasındakı trafiği şifreleyerek güvenli bilgi alış verişini sağlayan ve default'ta 443 nolu portu kullanan bir güvenlik protokolüdür [16]. SSL protokü ile veri karşı tarafa gönderilmeden önce belirli bir şifreleme algoritması ile şifrelenir ve sadece doğru alıcı tarafından bu şifre çözülerek gerçek veri elde edilir. Temelleri Netscape firması tarafından 1994 yılında atılan SSL aynı yılda ticari olarak piyasaya sürüldü ve bir sonrakı yıl IETF tarafından standart olarak kabul edildi. 1996 yılında ise v3.0 versiyonun yayınlanmasıyla bütün internet tarayıcılarının desteklediği bir standart hale gelmiştir [17]. Aslında standartın asıl ismi TLS ***(****Transport Layer Security****)*** olmasına rağmen genellikle SSL kullanımı tercih edilmektedir. SSL/TLS, uygulama katmanı ile taşıma katmanı arasında yer alır ve bilginin şifreleme gibi gerekli olan kriptografik işlemlerden geçtikten sonra karşı tarafa iletilmesini sağlar. SSL'in aşağıdaki sürümleri mevcuttur [16]:

- SSL v2.0 (1995), SSL v3.0 (1996)
- SSL v3.1 = TLS 1.0 (1999)
- TLS 1.1 (2006), TLS 1.2 (2011)

HTTPS/SSL/TLS, sadece iletişimde veri gizliliği sağlar, hedef sistemin güvenlik zaafiyetlerine karşı ek bir koruma sağlamaz.

HTTP + TLS veya HTTP + SSL

Her iki tarafta da doğrulama yapılarak işlemin ve bilginin hem gizliliği hem de bütünlüğü korunur [17].

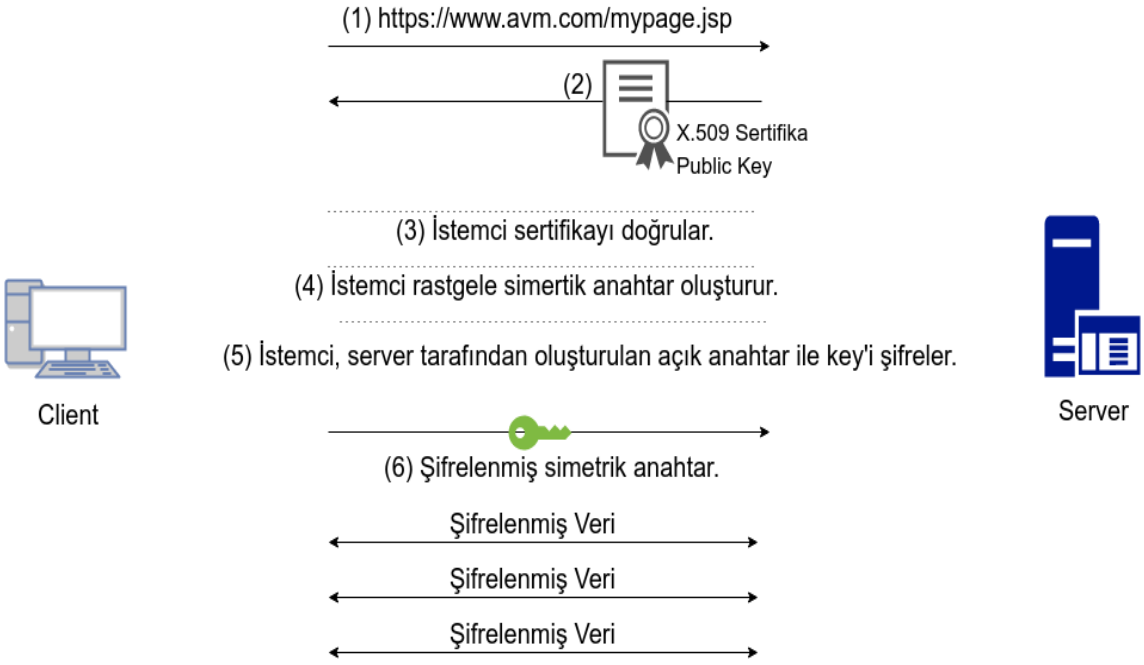
- Gizlilik → simetrik şifreleme (anahtar değişimi)
- Bütünlük → MAC
- Kimlik Denetimi → Dijital sertifika kontrolü

SSL sertifikası, web sitesinin kimliğini doğrulayan ve SSL protokolü kullanılarak sunucuya gönderilen ve alınan bilgilerin şifrelenerek aktarılmasını sağlayan dijital bir

belgedir. SSL sertifikaları aracılığıyla kullanılan SSL protokolü, internet veya bir bilgisayar ağı üzerinde güvenli bilgi akışını sağlayabilen bir teknolojidir. Çok fazla kaynak harcamasından dolayı güvenli veri iletişimi gerektiren web sayfalarında kullanılmalıdır [18]. Çünkü HTTPS kullanımının yoğun trafik akışına neden olması sistemin performansına yansır. SSL protokü ile veri karşı tarafa gönderilmeden önce belirli bir şifreleme algoritması ile şifrelenir ve sadece doğru alıcı tarafından bu şifre çözülerek gerçek veri elde edilir.

SSL sertifikalarında şifreleme algoritmaları simetrik ve asimetrik olmak üzere ikiye ayrılır;

1. *Simetrik şifreleme*- Veriyi şifrelemek ve şifreli veriyi çözmek için her iki tarafında bildiği ortak-tek bir anahtar kullanılır [18]. Bu teknikte her iki tarafta da aynı anahtar kullanıldığından güvenlik düşüktür.
2. *Asimetrik şifreleme*- Veriyi şifrelemek ve şifreyi veriyi çözmek için 2 farklı anahtar kullanılmaktadır. Bu anahtarlar public key ve private key'dir. Bu şifreleme algoritmasında veriyi alan taraf kendisinin bildiği bir private key ve diğer kişiler tarafından bilinen public key oluşturur [18]. Gönderen veriyi, alıcının public key'i ile şifreler. Alıcıya ulaşan şifreli veri, alıcının private key'i ile çözülür. Veriyi şifreli yapan private key'in gizli olmasıdır.
3. Çünkü belirli bir kişinin public key'i ile şifrelenen veri o kişinin private key'i ileçözülebilir. Dolayısıyla private key bilinmediği için araya girilse bile veri okunulmaz halde olur.



Şekil 3.16 : İletişimin şifrelenmesi

Veri + Private Key = Şifreli Veri

Şifreli Veri + Public Key = Veri

Şifreleme yönteminin gücü kullanılan anahtar uzunluğuna bağlıdır. Anahtar uzunluğu bilginin korunması için çok önemlidir. SSL protokolünde 40 bit ve 128 bit şifreleme kullanılmaktadır. Bit (*Binary Digit*), ikili sistemde bir rakamı ifade eder. Bir bit, 0 veya 1 olmak üzere iki farklı değer alabilir. 128 bitlik bir şifrenin çözülmesi için bir milyon dolarlık yatırım ve 67 yıl gerekmektedir [17].

Sayısal – Dijital imza kavramı

Göndericiye ait olan damgadır. Göndericinin doğruluğu sayısal imza ile kontrol edilir. Asimetrik şifreleme algoritması kullanılmaktadır. Sayısal imza, kişinin veya kurumun public key'inin hash değerinin alınması ve bu değerin bir sertifika otoritesinin private key'i ile imzalanmasıdır. Böylece kurumun kimliği geçerli bir sertifika otoritesi tarafından doğrulanmış olur.

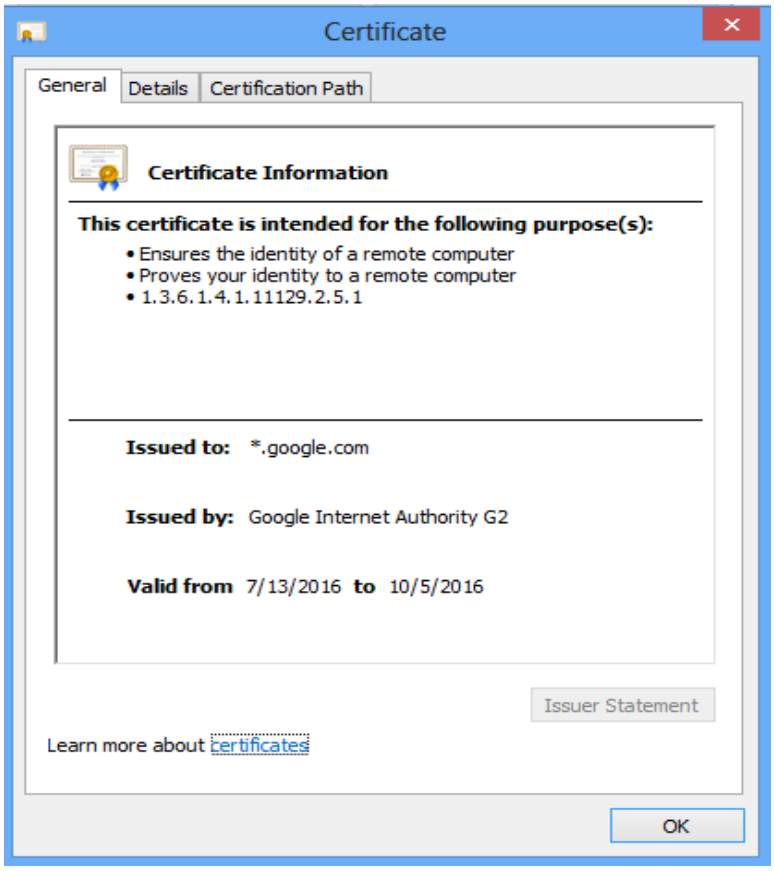
Dijital sertifikalar

Dijital sertifika ile bir web sitesi kendisini, kendisine erişmeye çalışan kullanıcıya tanıtır. Sertifika, kurumun public key'i ve bu public key'in hash değerinin sertifika otoritesinin private key'i ile imzalanmasından oluşur.

Sertifika = (Private KeyCA (Hash (Public KeyKurum))) + Public Key = Sayısal Imza + Public Key

SSL'in temel güvenliği Sertifika otoritesi olarak adlandırılan aracı kurumlar ve bu kurumlar bünyesinde tutulan gizli anahtarla sağlanır.

CA (*Certification Authority*)- sertifika otoritesi, sertifika vermeye yetkisi olan kurum. Noter mührü gibi. Bu kurumlara örnek olarak Verisign, Globalsign, COMODO, TRUST gösterebiliriz [6]. Sertifika otoritesinin ele geçirilmesi o otorite tarafından onaylanan tüm sertifikaları güvensiz hale getirir. SSL sunucularının dijital sertifikalarının olması zorunludur. Bu sayede tarayıcılar, sunucuların kimlik doğrulamasını gerçekleştirirler. Mutual Authentication aktif edildiyse, yani SSL sunucusunda tarayıcıların kimlik kontrolu gerçekleştirilmesi gerekiyorsa o zaman her bir tarayıcının da geçerli bir dijital imzaya sahip olması gerekir [17]. Bir dijital sertifikada aşağıdaki resimde gösterilen başlıca bilgiler yer almaktadır:



Şekil 3.17 : Sertifika detayı

Sertifikanın hangi kurum/web sayfası için oluşturulduğu, hangi CA tarafından oluşturulduğu, sertifikanın geçerli olduğu tarih aralığı ve sertifikanın bütünlüğünü kontrol etmeyi sağlayan hash değerleri.

2 türlü SSL sertifikası mevcuttur:

1. CA imzalı SSL sertifikaları
2. Self-Signed SSL sertifikaları

Self-Signed sertifikaları genellikle test amaçlı veya lokal kullanımda tercih edilir. Ancak bu sertifikayı oluşturan CA, tarayıcı tarafından tanınmadığı için kimlik doğrulama esnasında kullanıcılar uyarı ile karşılaşmaktadır. Self-Signed sertifikalar OpenSSL yardımı ile oluşturuluyorlar.

Dijital sertifika kontrolü

1. Sertifika geçerli bir CA tarafından oluşturulmalı ve de CA hiyerarşisindeki her CA ve sub CA sertifikası uygulama ortamında güvenilir ve geçerli olmalıdır. Self-Signed CA'lar sadece test ortamında kullanılmalıdır.

2. Her dijital imza belirli bir zaman aralığında geçerlidir. İlgili sertifikanın süresinin dolmadığından emin olunmalıdır. Süresi bitmek üzere olan sertifika yenilenmelidir.
3. Dijital sertifika'lar, ilgili kurumu temsil eden CN (*Common Name*) tanımı ile oluşturulur. CN (genel ad), kullanılan sunucunun hostname'ine karşılık gelmelidir. Farklı olması durumunda istemciye sertifika da bulunan kurum ismi ile erişilmek istenen site arasında çelişki olduğuna dair uyarı mesajı dönecektir. Bu uyarı, kullanıcıları erişilmek istenen adresin güvenli olup olmadığı konusunda şüpheye düşürebilir. Dolayısıyla sertifika ismi ile sunucu ismi aynı olmalıdır.

3.5 Web Uygulamlarına Yönelik Saldırı Türleri

3.5.1 Deneme yanılma saldırıları

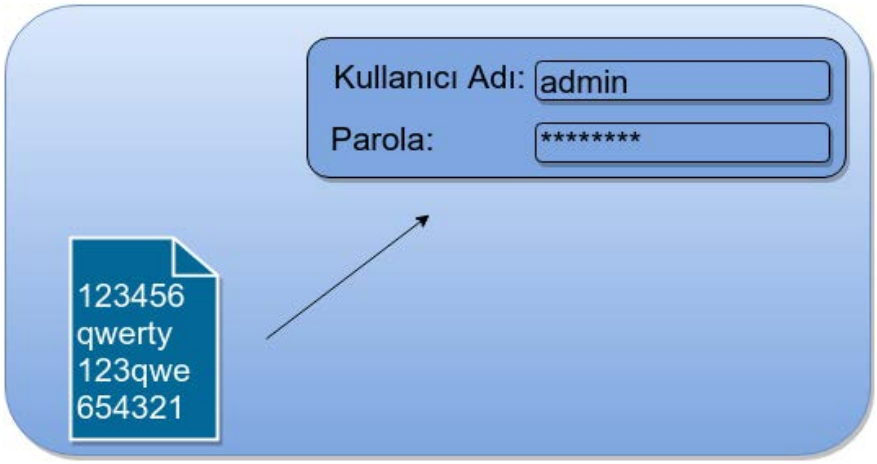
Hedef olarak genellikle uygulamanın giriş formlarını ele alan ve tahmini isteklerde bulunarak, bu isteklerin cevabının analizlerinden oluşan bir saldırı tipidir. Manuel olarak insanlar tarafından el ile yapılmaz (aslında yapılabilir ama uzun ve zahmetli bir iştir), bunun için yazılmış programlar aracılığı ile yapılır. Bu saldırıdaki asıl amaç kullanıcın kullanıcı adına ait şifreyi bulmak ve yönetimi ele geçirmektir. Yönetim ele geçirilince web sitesini kolayca hackleyebilirler. Bu saldırı çeşidinin uygulanabilir olmasının en büyük kaynağı, kullanıcıların kolay tahmin edilebilen ve karmaşık olmayan parolalar seçmelerinden kaynaklanmaktadır [3]. Parolalar ne kadar karmaşık olsa otomatize araçlar tarafından tahmin edilmesi bir o kadar minimuma indirilmiş olur. Başka bir etken de giriş formlarında bulunan detaylı hata mesajlarıdır. Bu saldırı tipinde saldırganlar otomatize araçlar aracılığı ile hızlı bir şekilde ellerinde olan kelime listeleri ile sürekli login sayfasına atak yaparlar. Bu kelime listeleri yazılı olanları program sırasıyla dener ve şifreyi bulmaya çalışır.

Deneme Yanılma Saldırıları genel itibari ile iki şekilde yapılmaktadır:

- Sözlük tabanlı saldırılar (Dictionary Attack)
- Kaba kuvvet saldırıları (Brute Force)

3.5.1.1 Dictionary Attack (Sözlük Saldırısı)

Daha önce oluşturulmuş bir listedeki tüm sözcüklerin denenmesi ile yapılır.

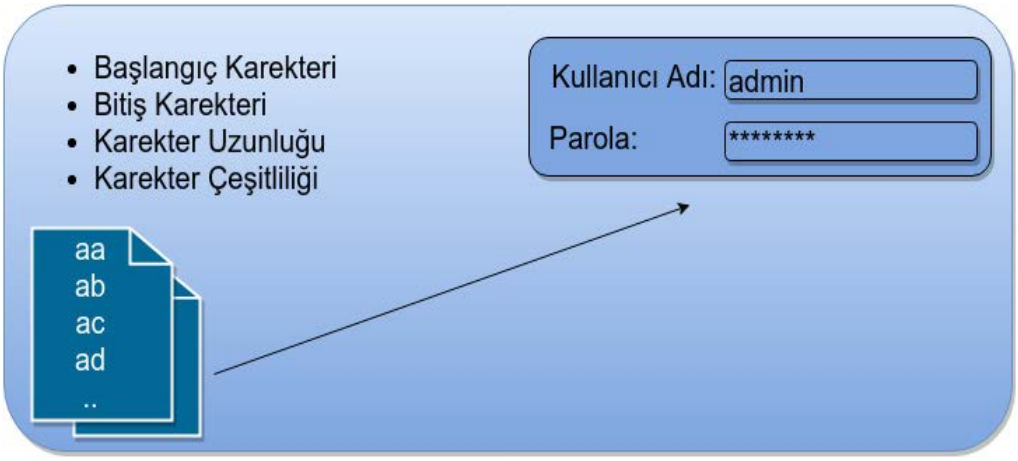


Şekil 3.18 : Sözlük saldırısı

Bu listeler genellikle GB'lar boyutunda olup daha önceden çalınmış veritabanlarından çıkartılmış en çok kullanılan kullanıcı adı ve parola bilgilerini içerir. Burp Suite, pentest işlemlerinde en çok kullanılan proxy programıdır. Web işlemleri gerçekleştirirken istemci-sunucu mimarisini kullanırız. Bir web sitesini incelemeye başladığımız zaman giden ve gelen verilere, isteklere, isteklerin gidiş ve dönüş şekillerine göre işlemler gerçekleştiririz. Pentest işlemlerinin sağlıklı işlenmesi için istemci ile sunucu arasında yapılan her işlemin gerek istemciden çıkıp sunucuya varmadan kontrolü, gerekse de sunucudan istemciye dönen cevabın istemciye varmadan araya girilerek bakılması çok önemlidir [18]. Burp Suite işte burada devreye girer ve istemci-sunucu arasındakı tüm verileri kendi üzerinden geçirerek kendi içerisinde bulunan özellikler ile test edilmesini sağlar. İçerisinde bulundurduğu özellikler sayesinde pentest işlemlerinde hız ve test etme kolaylığı sağlar.

3.5.1.2 Kaba kuvvet (Brute Force) saldırıları

Brute Force Attack olarak bilinir ve Türkçe'ye Kaba Kuvvet Saldırıları olarak çevrilir. Kaba kuvvet saldırısı, kimlik doğrulamada kullanılan kullanıcı adı ve parola değerleri için alt ve üst sınırları belirli olacak şekilde karakter, uzunluk ve çeşitlerinin oluşturulup denenmesi ile yapılmaktadır [6].



Şekil 3.19 : Kaba kuvvet saldırısının denenmesi

3.5.1.2.1 Kaba kuvvet (Brute Force) saldırılarını önlemek

Kaba kuvvet saldırıları hedeflenen uygulamanın kritik formlarında CAPTCHA (*Completely Automated Public Turing test to tell Computers and Humans Apart*) kullanılmalıdır [3]. CAPTCHA, bilgisayar ile insanların davranışınlarının ayırt edilmesi için kullanılmaktadır. Buradaki mantık resim üzerinde insan tarafından okunabilen fakat bilgisayar programları tarafından okunması zor olan bir sözcük oluşturmaktır. Eğer forma girilen sözcük resimdeki ile aynı değilse ya kişi formu yanlış doldurmuş yada formu dolduran bir programdır denilebilir. Bu sayede, uygulamanın herhangi bir sayfasına kötü niyetli yazılım tarafından otomatik olarak gönderilen veri önlenebilir. Ayrıca CAPTCHA kullanımı sözlük saldırılarını da engeleyebilir.

CAPTCHA ile ilgili sorunlar;

- Zayıf algoritmalar
- Zayıf resimler

- CAPTCHA tekrarlama

CAPTCHA uygulaması kullanılan uygulamalara yapılan saldırılardan biri, saldırganın HTML kaynak koduna bakarak image etiketi bulunan kısımları incelemesi ve image'ların ALT attributlarına bakılarak CAPTCHA tahmin edilmeye çalışılır [3].

Görünmez Link form tuzakları: Genelde web uygulamalarında otomatize araçlar ile bilgi çalmayı önlemek için kullanılan yöntemdir. Bu tuzağın temel amacı bir kullanıcı ile otomatize aracı bir birinden ayırt etmektir. Dolayısıyla görünmez link form eklentileri ile otomatize araç tespiti yapılabilir. Bu link ve formlar kullanıcı tarafından görüntelenemeyecek ancak otomatize araçlar bu link ve formları bularak istek yapacaktır. Dolayısıyla bu isteği bir kullanıcı yapmış olamaz [6]. Bu isteklerin client IP bilgisi karantinaya alınıp belirli bir süre bu IP'den gelen isteklere CAPTCHA gösterilebilir.

Kimlik Doğrulama İşlemlerine Gecikme Süresi Ekleme: Deneme yanılma saldırılarının başarı oranları seçilen yönteme bağlı olarak ve test edilecek veri kümesinin büyüklüğüne orantılı olarak değişir [6]. Çünkü hedef uygulama üzerindeki kullanıcı adı ve parola politikası karmaşık ise yani harf ve rakamlardan oluşuyor ise daha büyük veri kümesi test için kullanılacaktır ve buda geçerli kullanıcıyı bulma süresini artıracaktır. Örneğin, her istek için bir saniye harcanıyorsa 1000 istek 1000 saniye harcanması sözkonusudur. Sistem güvenliğini sağlamak için belirli hatalı giriş sayısından sonra cevap süresi artırıla bilir. Örneğin, ilk iki hatalı girişden sonra bir saniye ve toplamda 7 hatalı girişden sonra cevap süresi 5 saniye ile sınırlandırıla bilir. Bu sayede servis kesintisi zafiyeti için birinci önlem alınmış olur fakat mevcut formun her zaman bir captcha ile kullanılması tavsiye edilir.

Doğru Bilinen Yanlışlar;

Birçok sızma testi esnasında yazılım geliştiricilerin saldırılara karşı almış olduğu önlemler her ne kadar bu problemi çözse bile farklı sorunlara neden olabilmektedir. Bunları sıralayacak olursak;

Captcha; İlk giriş esnasında captcha kullanılması önerilmez. En azından bir hatalı girişden sonra kullanılması önerilir [6].

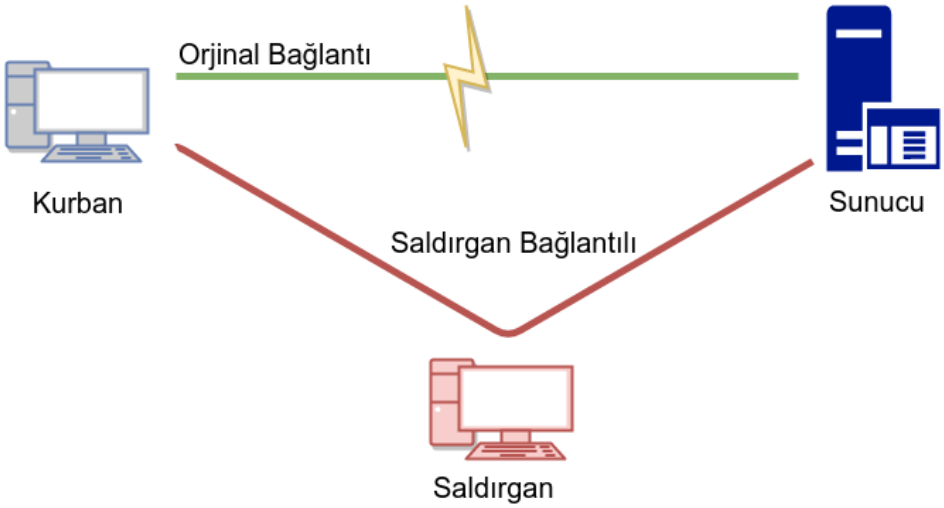
Form Token; Captcha ve form token kullanımı güvenliğin sağlandığına inanılıp zayıf kullanıcı adı ve parola politikası seçmek yanlıştır [6].

3.5.2 SSL saldırıları

SSL bağlantılarında araya girme aşağıdaki gerekli şartlar sağlandığında mümkün olmaktadır:

1. İlk olarak hedef sistemin trafiği saldırganın üzerinden geçecek şekilde ayarlanmalıdır;
2. Hedef sistemin iletişim kurmak istediği HTTPS sayfasına ait bilgileri ile sahte bir sertifika oluşturulmalıdır;

Sahte oluşturulan bu sertifika tüm modern tarayıcılarda kullanıcıya uyarı verecektir. SSL güvenliğindeki en önemli bileşen sertifika otoritesidir. Sertifika otoritesi tek başına işe yaramaz, client yazılımları tarafından güvenilir olarak kabul edilmelidir. SSL'in karşı karşıya kaldığı ilk ve önemli saldırı tipi MITM (*Man in The middle*) saldırılarıdır [17].



Şekil 3.20 : MITM saldırısı

MITM saldırısı ile saldırgan kendisini istemci ve sunucu arasındaki trafiğin kendi üzerinden geçmesini sağlayarak bütün iletişimi kontrol etmeyi hedefler [17]. Saldırgan, kurban olan istemciye sunucu gibi davranırken, Sunucu'ya ise normal bir kullanıcı gibi davranır. SSL' de araya girme saldırılarının başarılı olmama sebeplerinden biri de, araya giren saldırganın ürettiği sahte sertifikayı bir şekilde

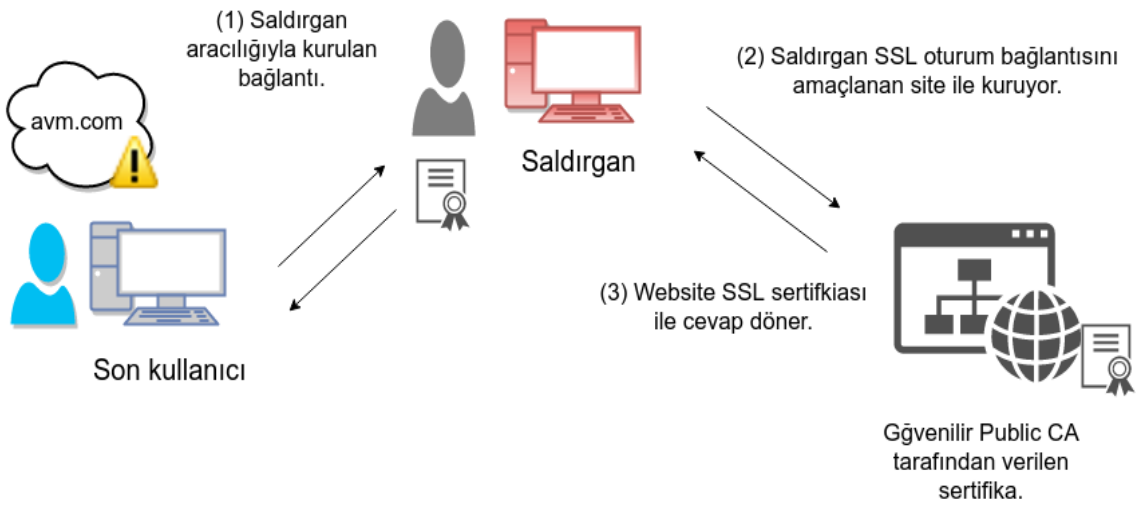
kurbana kabul ettirmesi gerekliliğidir. Saldırgan sahte sertifika ile trafiği dinlemeye başladığında onaylı bir sertifika olmadığı için kurban tarayıcı tarafından uyarı ile karşılaşır ve trafiği dinlemesi engellenir.

SSL'in sertifikalar yoluyla gerçekleştirdiği kimlik denetiminin MITM saldırılarına engel olduğu düşünülüyordu. Fakat 2002 yılında ilk defa birçok tarayıcı uygulamasının bir sertifikanın CA sertifikası olup olmadığını gösteren BasicConstraints: CA uzantısını kontrol etmediği tespit edildi [18]. Bu yetersiz kontrol sonucu sayesinde istenilen bir domain için geçerli bir sertifika oluşturulabilir. Birçok kullanıcı da bunun farkına varmadan gelen uyarıyı kabul edip işlemlerine devam edebilir.

Şekil 3.21 : Güvensiz bağlantı bilgisi

3.5.2.1 Zafiyetin istismarı

2009 yılında HTTPS trafiğini HTTP'ye yönlendirmek suretiyle MITM saldırısı yapmayı sağlayan sslstrip uygulaması yayınlandı. Bu araç, her iki protokolün birlikte desteklendiği sistemlerde ortaya çıkan güvenlik zafiyetini kullanarak aradaki trafiği dinlemek ve değişiklik yapmak için geliştirilen bir araçtır.

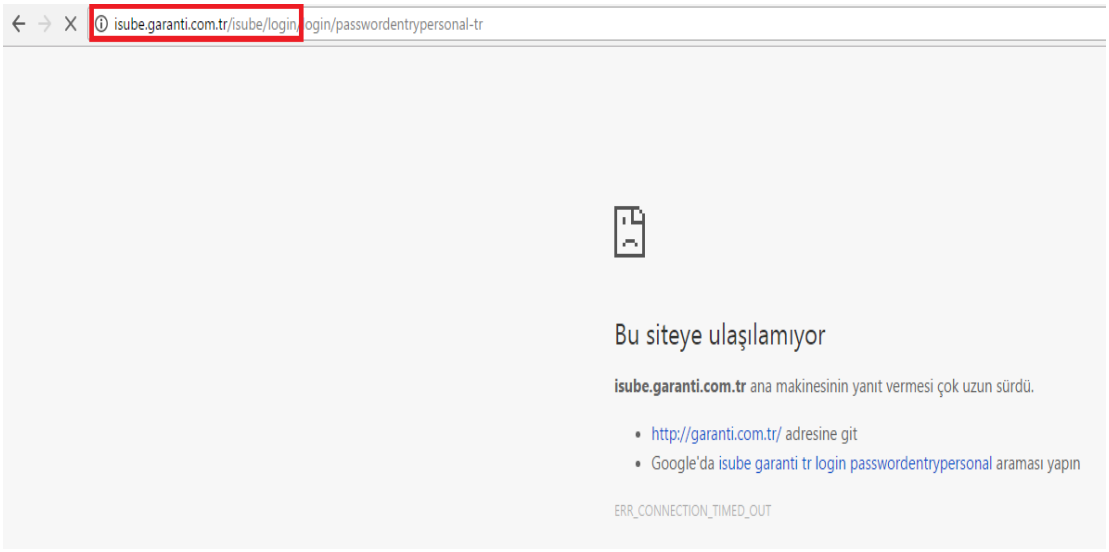


Şekil 3.22 : SSL zafiyetinin istismarı

Kurban her iki protokolün de desteklendiği herhangi bir uygulamaya öncelikle HTTP üzerinden giriş yapar. Bu durumda araya giren saldırgan SSL Strip aracı ile kurbanın yapmış olduğu istek içerisinde bulunan HTTPS bağlantılarını HTTP'ye çevirerek kurban ile arasında HTTP trafiği başlatır. Aynı zamanda kurbanın yapmış olduğu isteği de sunucu ile kendisi arasında HTTPS kurarak başlatır. Bu aşamada kullanıcı bir istekte bulunduğunda saldırgana HTTP ile bağlanır ve saldırgan yapılan bu isteğin içerisindeki değişiklikleri algılayarak sunucuya yönlendirir. Saldırgana gelen sunucu cevabı da kurbana HTTP'ye çevrilerek yönlendirilir [6].

Alınacak Önlemler;

1. Kullanıcı kritik bir uygulamaya giriş yaparken URL üzerinden gidilen adresin HTTPS olduğundan emin olmalıdır [6]. Genellikle bankalar, internet bankacılığı uygulamalarından bu tür saldırılara önlem olmak amaçlı HTTP desteği vermezler. Aşağıdaki resimde görüldüğü gibi kritik uygulamaya HTTP üzerinden erişilmek istendiğinde sayfaya erişimin olmadığı ile ilgili **404** veya **403** hata sayfası gösterilmektedir:

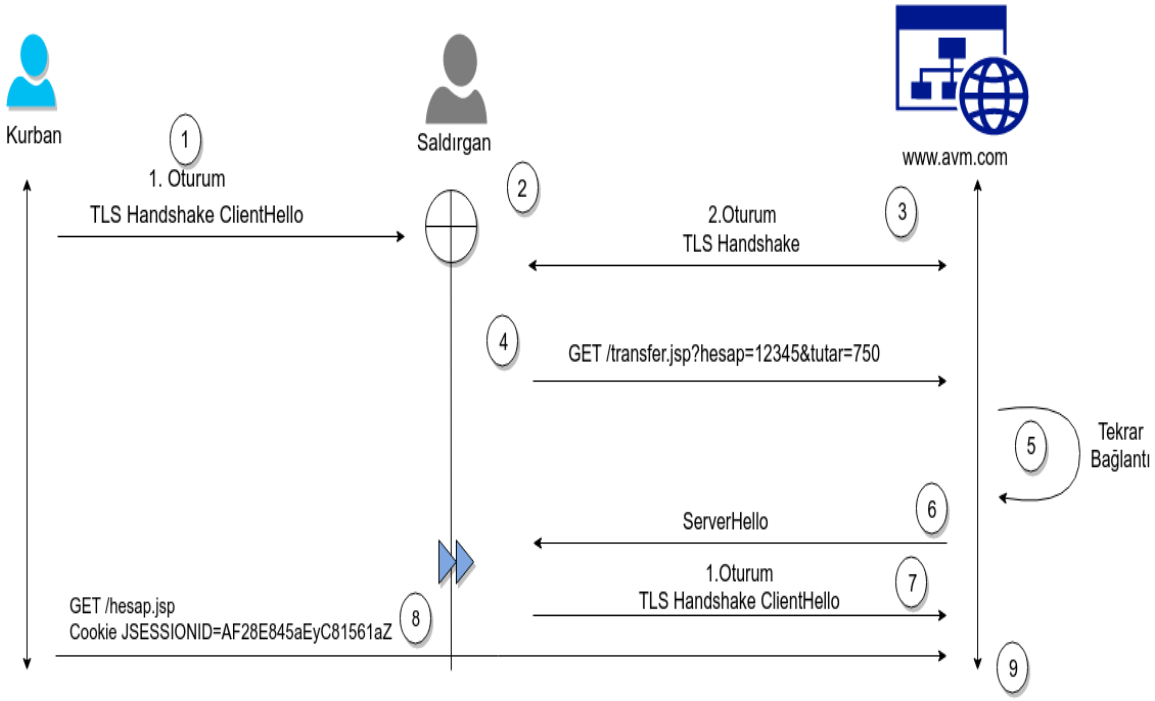


Şekil 3.23 : SSL saldırısını önlemek

Bu sayede SSL Strip gibi saldırılara maruz kalınması engellenmiş olur. Ayrıca tarayıcı bazlı güvenlik önlemi almak için her tarayıcıya özel noScript uzantısı kullanılabilir. Bu sayede belirlenen sitelere HTTPS dışında erişim engellenebilir [18].

3.5.2.2 Renegotiation saldırısı

SSL protokolü, renegotiation özelliği sayesinde el sıkışma işlemi gerçekleştirdikten sonra istendiği durumda el sıkışma işlemini tekrarlayarak şifreleme için yeni gizli anahtar kullanılmasını desteklemektedir [6]. Renegotiation özelliğine bağlı bir güvenlik açığını gösteren senaryoyu aşağıdaki gibi gösterebiliriz:



Şekil 3.24 : Renegotiation saldırı senaryosu

Bu zafiyetin oluşması için ilk önce saldırganın bir MITM ortamı oluşturması gerekmektedir. Bu resmi inceleyecek olursak [6];

1. Adımda kurban gerçek sunucu ile bir TLS- handshake işlemi başlatır.
2. Adımda saldırgan kullanıcıdan gelen isteği durdurur ve handshake paketlerini daha sonra sunucuya göndermek için kaydeder (7. adım).
3. Adımda saldırgan, sunucu ile TLS-handshake işlemi başlatır ve sonlandırır. Bu sayede saldırgan ile sunucu arasında güvenli bir iletişim kurulmuş olur.
4. Adımda saldırgan sunucuya HTTP üzerinden aşağıdaki gibi bir istekte bulunmaktadır:

GET/transfer.jsp?hesap=150

ABC:

Fakat GET isteğinin sonlanması için gerekli olan son CRLF karakteri gönderilmez. Bu sayede ilgili istek sunucuda son CRLF gelene kadar bekletilmiş olur (genellikle bir dakika).

5. Sunucu kendisine gelen isteğin güvensiz kanal üzerinden geldiğini görür ve sertifika ile kimlik doğrulama işlemine tabi tutar. Bunun için renegotiation başlatır.

6. Sunucuda başlayan renegotiation sayesinde saldırgana ServerHello isteği gelir.

7. Saldırgan kendisine gelen ServerHello isteğine karşılık, kurbandan daha önce gelen ve beklettiği ClientHello mesajını döner. Dolayısıyla kurban ve sunucu arasında güvenli iletişim kanalı kurulmuş olur.

8. Kurban uygulamaya HTTP üzerinden aşağıdaki gibi bir istekte bulunmaktadır:

GET/ hesap.jsp

Cookie: JSESSIONID=AF28E845aEyC81561aZ

Sunucu 4. adımda kendisine gönderilen ve bekletilen istek için hala CRLF karakteri beklediği için gelen isteği bir önceki isteğin arkasına ekler.

GET/transfer.jsp? hesap =150

GET / hesap.jsp

Cookie: JSESSIONID=AF28E845aEyC81561aZ

Son istekte görüldüğü üzere ABC kısmı olan satırı sunucu dikkate almayacak ve 1. satırdaki saldırganın isteği ile 3. satırdaki kurbanın geçerli Cookie bilgisi birleştirilecektir. Dolayısıyla başarılı bir saldırı gerçekleştirilmiş olacaktır. Renegotiation'a engel olmak için ya sunucu tarafındakı SSL ayarlarında istemci'den gelen renegotiation istekleri reddedilmeli ya da IETF tarafından yayınlanan yama programın TLS tarafından desteklendiğinden emin olunmalıdır. Zayıf SSL sürümleri kapatılmalıdır. Sunucunun renegotiation'ı destekleyip desteklemediğini test etmek için OpenSSL aracı kullanılabilir [18].

Son olarak SSL güvenliğini test etmek için;

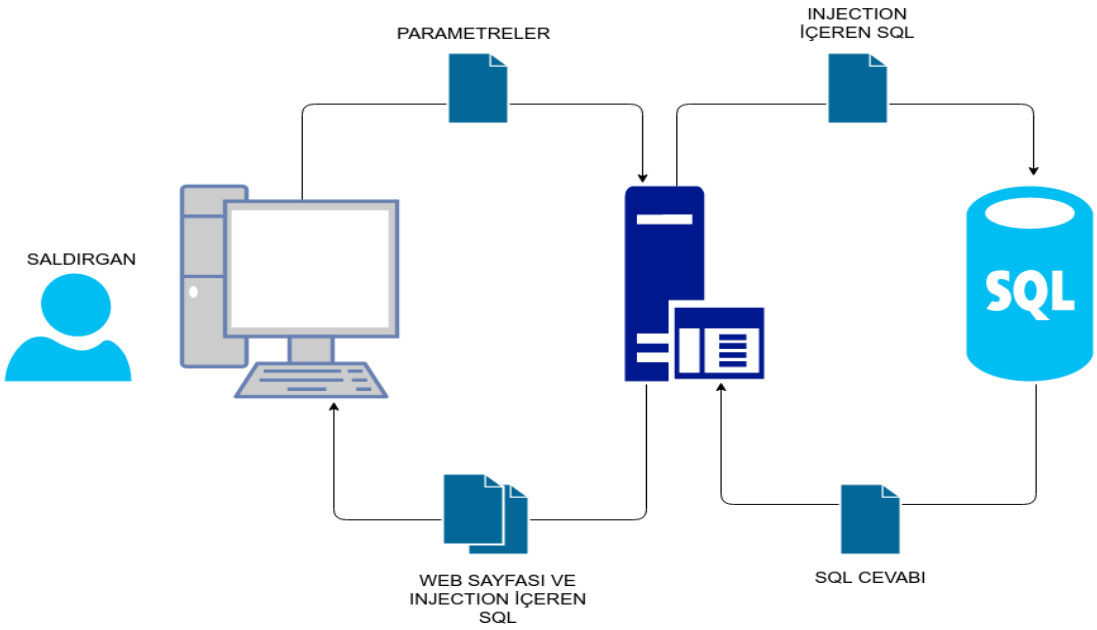
1. Zayıf algoritmalar kapatılmalı;
2. TLS'in güncel sürümleri kullanılmalı;
3. Dijital sertifikanın durumu kontrol edilmelidir.

3.5.3 SQL saldırıları

Web uygulamaları pek çok teknoloji katmanından ve entegre çalışan pek çok sistemden oluşmaktadır. Dolayısıyla farklı katmanlarda ve farklı iletişim aşamalarında farklı yazılım dillerine yönelik enjeksiyon açıkları bulunmaktadır. Ama asıl önemli olan açık, kullanıcı tarafından girilen verilerin yetersiz kontrolüdür. SQL tüm dillerde ortak kullanılan bir dil olduğundan dolayı SQL ile işlem yapan uygulamalarda açık bulunma olasığı yüksektir. Injection saldırıları, web uygulamalarındai en ciddi ve tehlikeli açıkların başında gelmektedir. Injection **veritabanı** ve **dilden bağımsız olarak** her türlü uygulama-veritabanı ilişkisine sahip sistemde bulunabilir ve bu açık **veritabanı veya sunucu açığı değildir.** Uygulama dinamikleri içerisinde ki hatalı kodlama, zararlı karakterlerin filtrelenmemesi, değişkenlerin yanlış atanması gibi hatalar uygulama zafiyeti olarak nitelendirilir. SQL yorumlanabilir bir dil olduğundan dolayı uygulama'lar tarafından oluşturulan SQL ifadelerini çalıştırabilmekte ve yorumlayabilmektedir. Eğer bu işlem güvensiz bir yol ile yapılırsa, uygulama SQL Enjeksiyonuna karşı savunmasız duruma gelecektir. Başarılı bir injection saldırısı sayesinde veritabanı içerisindeki, kullanıcıların kredi kartı bilgileri, kullanıcı hesapları, kişisel bilgileri vb. gibi önemli bilgilere ulaşılabilir, bu bilgiler üzerinde istenilen değişiklik yapılabilir ve hatta veritabanı'nın üzerinde çalıştığı sunucu kontrol altına alınabilir.

3.5.3.1 SQL injection

Kullanıcıdan (istemciden – client) alınan dataların hiçbir denetlemeden geçmeden izinsiz olarak arka tarafda SQL sorgusuna manipüle edilip amaca hizmet edecek şekilde uygulamada kullanılarak çıktılar üretmesini sağlayan zafiyet türüdür. TOP 10 listesindeki en riskli güvenlik açığıdır [20].



Şekil 3.25 : SQL enjeksiyon senaryosu

Web uygulamalarında bir çok işlem için kullanıcıdan alınan veri ile dinamik SQL sorguları oluşturulur. Mesela "SELECT * FROM Products" örnek SQL sorgusu basit şekilde veritabanından web uygulamasına tüm ürünleri döndürecektir. Bu SQL cümlecikleri oluşturulurken araya sıkıştırılan herhangi bir *meta-karakter* SQL Injection' a neden olabilir. Meta-karakter, bir program için özel anlamı olan karakterlere verilen isimdir. Örneğin; (\) backslash karakteri bir meta-karakterdir. Compiler (*derleyici*) ya da Interpreter (*yorumlayıcı*) bu karakteri görünce ondan sonraki karakteri ona göre işler. SQL' için kritik metakarakter **(')** tek tırnak' tır. Diğeri ise (;) noktalı virgüldür. SQL Enjeksiyonu için olan ilk testlerde tek tırnak (') veya noktalı virgül (;) kullanılmaktadır. Tekli tırnak SQL sorgusunda string sonlandırıcısı olarak kullanılır ve eğer uygulama tarafından filtrelenmiyorsa geçersiz bir sorgu ile sonuçlanabilir. Noktalı virgül ise SQL sorgusunun sonlandırılmasında kullanılır ve eğer filtrelenmemiş ise bu da bir hata mesajı dönderecektir. (*Unclosed quotation mark before the character string "*. gibi) [20] Bundan başka kendinden sonra gelenlerin çalıştırılmaması için – karakteri ve AND ve OR gibi kelimeler de sorguyu değiştirmekde kullanılmaktadır.

- Dinamik SQL sorgusu:

  String query = "SELECT * FROM TABLE where ID = ' " + id + " ' ; "

- Statik SQL sorgusu:

SELECT * FROM TABLE where ID = ‘100’;

- Injection içeren SQL sorgusu:

  SELECT * FROM TABLE_1 where ID_1 = ‘100’ UNION SELECT * FROM TABLE_2 WHERE ID_2 = ‘1’;

Web uygulamasında olası bir üye girişi işlemleri aşağıdaki gibidir;

1. Formdan gelen kullanıcı adı ve şifre bilgisi ile ilgili SQL cümleciği oluşturulur.

   SELECT * FROM TABLE WHERE username=’admin’ AND password=’1234’

2. SQL cümleciği kayıt döndürüyorsa böyle bir kullanıcının var olduğu anlamına gelir ve session(*oturum*) açılır, ilgili kullanıcı üye girişi yapmış olur.

3. Eğer veritabanından kayıt dönmediyse "*kullanıcı bulunamadı*" veya "*şifre yanlış*" gibi bir hata ile ziyaretçi tekrar üye girişi formuna gönderilir.

Örnek bir üye girişi kodu;

Aşağıdaki url ile example.com adresine test/123456 kullanıcısı ile üye girişi yapmak istediğimizi belirtelim.

http://www.example.com/login.jsp?username=test&password=123456

Java/JSP ile yazılmış örnek bir üye girişi kodu:

1. String username = request.getParameter (“username”);
2. String password = request.getParameter (“password”);
3. String query = “SELECT * FROM members WHERE username = ‘ ” + username + “ ’ AND password = ‘ ” + password + “ ’ ” ;
4. ResultSet rs = stmt.execute (query);
5. **if** (userExist != **null**){
6. request.getSession().setAttribute("username", username);
7. response.setStatus(200);
8. response.sendRedirect("index.jsp");

9. }**else**{
10. request.getRequestDispatcher("login_error.jsp").forward(request,response);
11. }

1. ve 2. satırda form değişkenlerinin değerleri alınıyor.

3. satırda sql cümleciği içerisine yerleştirilip kullanıcı kontrolu yapılıyor.

5. satırda sonucun boş olup olmadığına bakılıyor. Eğer boş ise, yani kullanıcı veritabanında bulunmadıysa, 10. satırda gösterildiği gibi kullanıcı login_error.jsp sayfasına yönlendiriliyor. Eğer bulunduysa, 6.7.8. satırdakı işlemler yapılıyor. Yani, kullanıcıya username içeren bir session açılıyor ve bu sayede kullanıcı index.jsp sayfasına giriş yapmış oluyor.

Bu isteğin veritabanı sunucusundaki çalıştırılma şekli aşağıdaki gibidir:

SELECT * FROM TABLE WHERE username = ' test ' AND password = ' 123456"

Klasik bir üye giriş işlemi yukarıda belirtildiği gibidir. Saldırgan, uygulama yöneticisinin username bilgisine sahip ise (bu örnekde administrator'un kullanıcı adının "admin" olduğunu varsayalım) o zaman şifre kısmını bypass edebilir. Saldıran kullanıcı adı olarak admin ' -- girdiğinde sql sorgusu üzerinde bir injection gerçekleştirmiş olur. ' -- ifadesi bir injection ifadesi olup, kendisinden sonra gelen sql ifadelerini yorum cümleciklerine çevirir. Bu sayede sorguda bulunması gereken şifre kontrolü bypass edilmiş olur ve sisteme şifresiz bir biçimde giriş yapılmış olur.

Kullanıcı değerleri ile oluşan SQL sorgusu;

SELECT * FROM TABLE WHERE username = ' admin ' -- 'AND password = ' " + password + " ';"

Veritabanı üzerinde çalışacak olan SQL sorgusu;

SELECT * FROM TABLE WHERE username = ' admin '

Şimdi ise saldırganın kullanıcı adı bilgisine sahip olmadığını varsayalım. Genel olarak uygulamalarda, veritabanı üzerindeki ilk kullanıcı hesabı uygulamayı yönetme yetkisine sahip olan kullanıcının olmalıdır, çünkü bu hesap normal olarak elle oluşturulur ve sonra bu hesap üzerinden uygulama aracılığıyla diğer tüm hesaplar oluşturulur.

Saldırgan sık sık bu davranışdan yararlanarak kullanıcı adı aracılığıyla veritabanı üzerindeki ilk kullanıcı hesabıyla login olmaya çalışmaktadır. Bir önceki örneğimizde yer alan injection ifadesine benzer olarak aşağıdaki injection ifadesi girilir;

’ or 1=1

http://www.example.com/login.jsp?username=test’ OR ‘1’=’1--&password=123456

Bu istek aşağıdaki biçimde SQL sorgusuna dönüşecektir.

SELECT * FROM TABLE WHERE username = ‘test‘ or 1 = 1 --' AND password = ' 123456’

Bu isteğin veritabanında çalışma şekli;

SELECT * FROM TABLE WHERE username = ‘test‘ or 1=1

Ilgili sorgu kullanıcı adını bulamasa bile OR bağlacı kullanıldığından ve 1=1 herzaman TRUE olduğundan uygulama tüm kullanıcılarının ayrıntılarını döndürüyor olacaktır. Herhangi bir siteye giriş yapıldığında login işlemi gerçekleştirildikten sonra kullanıcıya bir COOKIE verilir ve oturum başlar. Oturum bilgisine eklenen datalar veritabanından gelen bilgiler olabilmektedir. Bu durumda veritabanından birden fazla kullanıcı döneceği için sistemler farklı şekilde çalışabilirler.

Bu durumda LIMIT komutunu kullanarak tablodakı kullanıcıları sınırlandırıp bir tanesini getirebiliriz.

SELECT * FROM TABLE WHERE username = ' " + username ' or ‘1’ = ‘1 + " ' AND password = ' " + password ' or ‘1’ = ‘1--’ + " LIMIT 0,1;"

Tablodaki ilk kayıt admin kullanıcısına ait olduğu için sisteme admin olarak giriş yapmış olacağız.

String parametrelere enjeksiyon

SQL sorgularında string ifadeler bildiğimiz gibi tek tırnak içine yerleştirilir. Böyle bir alana enjeksiyon yapabilmek için bu tek tırnağın kapatılması ve ek kodların enjekte edilmesi gerekmektedir. Burada aşağıdaki adımlar uygulanabilir;

1. HTML formlarına, URL parametrelerine kullanıcı girdisi olarak tek tırnak girilip uygulamanın detaylı hata mesajı verip vermediği veya normal davranışından farklı bir biçimde davranıp davranmadığına bakılır. Bu sayede çalışan uygulama içerisinde SQL injection'a karşı herhangi bir önlemin olup olmadığı hakkında bilgi sahibi olunur [21].
2. Eğer hata veya farklı bir davranışla karşılaşılmışsa o zaman 2 tek tırnak gönderilir. 2 tek tırnak veritabanları tarafından escape karakteri olarak kullanılıp tırnak karakterini gerçek anlamıyla anlaması için kullanılır. Bu durumda hatalı durum ortadan kalkıyorsa web uygulaması büyük ihtimalle SQL Enjeksiyonuna açıktır diyebiliriz [20].
3. Açıklığı doğrulamak için ek olarak string birleştirme karakterleri yapılarak geçerli bir girdi verisi oluşturulur. Eğer uygulama bu duruma normal girilen veride olduğu gibi cevap veriyorsa o zaman bu uygulama SQL Enjeksiyonuna açıktır diyebiliriz [18]. Bu duruma örnek olarak;

Oracle: '||'ABC

MS-SQL: '+'ABC

MySQL: ' ' ABC (iki tek tırnak işareti arasında boşluk karakteri bulunmaktadır)

Birleştirme karakterleri, HTTP protokolunde kullanılırken URL'de encode edilmelidir. Yani,

1. Parametre adı ve değeri çiftini oluşturmak için kullanılan & ve = karakterleri URL'de sorgu metnini oluştururken sırasıyla %26 ve %3d olarak encode edilmelidir.
2. Sorgu metinleri içerisinde boşluk karakteri kullanılamaz. Eğer kullanılırsa, HTTP sorgu metni o anda sonlandırılır. O yüzden de boşluk karakteri %20 veya + olarak encode edilmelidir.

3. + karakteri boşluk kodlama amacıyla kulanıldığında gerçekten + işaretinin sorgu içinde kullanılması için %2b şeklinde encode edilmesi gerekmektedir.
4. Noktalı virgül, cookiler'i ayırmak için kullanıldığından %3b olarak kodlanmalıdır.

Numeric parametrelere enjeksiyon

Kullanıcının girdiği veri numeric olsa bile uygulama bu veriyi string olarak ele alıyor olabilir. Bazı durumlarda da numeric veri veritabanına numeric olarak gönderilir ve tek tırnak içine alınmaz. Bu nedenle önceki adımlar bir sonuç üretmiyorsa, aşağıdaki adımlar ele alınmalıdır:

1. Girdi olarak numeric değere eşit olan aritmetik işlem girilir. Örneğin, 5 girilen bir alana 3 + 2 girilir. Eğer uygulama 5 değeri için verdiği yanıtı veriyorsa, zafiyet barındırabilir [21].
2. Bir önceki adımdaki testin açıklığa işaret olabilmesi için değiştirilen verinin uygulama davranışı üzerinde önemli bir etkisi olması gerekir. Örneğin; Değiştirilen değer numeric bir kullanıcı id bilgisine ait ise doğru veya yanlış birşey ifade etmesi uygulama açısından önemlidir. Ama değiştirilen veri yerine tamamen rastgele bir değer de girilmesi ile sonuç değişmiyorsa injection açığı var diyebilmek için ilk adım yetersiz kalacaktır [18].
3. İkinci adımda ifade edildiği gibi uygulama üzerinde herhangi bir etki gerçekleşti ise daha komplike SQL ifadeleri ile ek kanıt aranabilir. Buna örnek olarak ASCII fonksiyonu verilebilir. Örneğin 'F' karakterinin ASCII değeri 70 olduğundan 5 değerine ulaşmak için şu ifade test edilebilir: 75 – ASCII('F') [19]
4. Yukarıdaki test tek tırnakların uygulama veya ara bir katman kontrolleri tarafından temizlenmesi durumunda çalışmayacaktır. Bu aşamada veritabanı sistemlerinin aslında implicit olarak numerik veriyi string'e çevirdikleri gerçeğinden yola çıkarak numerik bir değerin karakter olarak ASCII değerini kullanabiliriz. Örneğin '1' karakterinin ASCII değeri 49 olduğundan şu ifade 2 ile sonuçlanacaktır: 51-ASCII (1) [19]
5. Numeric paratmetrelerde, String parametreler de olduğu gibi URL encoding'e dikkat edilmelidir. Özel karekterler sunucuya HTTP üzerinden encode edilerek gönderilmelidir [20].

SQL saldırı zafiyet çeşitleri

Zafiyetin tespiti diğer zafiyetlerde olduğu gibi yapılan isteğe karşı sunucudan alınan cevabın analiziyle mümkün olmaktadır. Bu sonuçların farklılıklarına göre de SQL enjeksiyonu zafiyet çeşitleri bir birinden ayrılmaktadır [3]:

1. Basit SQL Enjeksiyonu
2. Union Tabanlı (Based) SQL Enjeksiyonu
3. Kör Tabanlı (Blind Based) SQL Enjeksiyonu
4. Zaman Tabanlı (Time Based) SQL Enjeksiyonu
5. Hata Tabanlı (Error Based) SQL Enjeksiyonu

Basit SQL enjeksiyonu

Yaygın olarak bilinen bir senaryo üzerinden açıklayalım;

http://www.example.com/index.jsp?id=1 AND 1=1

SELECT * FROM TABLE WHERE ID = ‘ “ + id + “ ’ “ AND 1=1”

http://www.example.com/index.jsp?id=1 AND 1=0

SELECT * FROM TABLE WHERE ID = ‘ “ + id + “ ’ “ AND 1=0”

Ilk sorguya baktığımızda AND yapısına gelmeden önceki kısımın doğru çalıştığını görürüz. AND mantıksal ifadesini kullanmaktakı amacımız string olarak veritabanına gönderdiğimiz mantıksal ifadenin veritabanı tarafından işlenip işlenmediğini görmektir. Veritabanı AND 1=1 sorgusuna TRUE yanıtını dönerken AND 1=0 kısmına FALSE yanıtını dönecektir. Bu iki sorgu için tarayıcıdan dönen cevap farklı ise potansiyel bir SQL Injection zafiyetinin mevcut olduğunu söyleye biliriz.

Union tabanlı SQL enjeksiyonu

Union bazlı SQL Enjeksiyonunda yapılacak testlerden biri Union işlemi ile orjinal SQL sorgusuna başka bir SQL sorgusu eklemektir.

Bu zaman eklenen sorgunun sonuçları Orijinal sorgunun sonuçlarına eklenir ve saldırganın diğer tablodakı verilerin değerlerini alabilmesini sağlar. [18]

SELECT * FROM TABLE_A **UNION** SELECT * FROM TABLE_B

Yukarıdaki sorguda verilen Union ifadesi sağındaki ve solundaki iki sorguyu ayrı ayrı çalıştırıp, her ikisinden dönen sonucu birleştirip tekil olan kayıtları çekmek için kullanılır. Bu sayede saldırgan SQL ifadesini manipüle ederek uygulama'nın verisini getireceği tablo'ya ek olarak diğer tablo'dan da verileri alabilecektir.

Union ALL ise DISTINCT komutu kullanımını aşmak için kullanılır. Yani kayıtları tekil olmayacak şekilde listeler. Union bazlı SQL enjeksiyonunda çıktı verilebilmesi için kullanılan iki tablodaki kolon sayılarının aynı olması gerekmektedir. Kolon sayılarının bulunmasında bize ORDER BY keyword'ü yardımcı olacaktır [6].

Örneğin, tablonun 3 kolondan (ID, USERNAME, PASSWORD) ibaret olduğunu varsayalım.

SELECT * FROM TABLE ORDER BY 1

yukarıdaki sorgu ile ID kolonuna göre sıralama yapabiliriz.

SELECT * FROM TABLE ORDER BY 3

PASSWORD kolonuna göre sıralama yapabiliriz. Fakat eğer aşağıdaki gibi bir sorgu çalıştırırsak;

SELECT * FROM TABLE ORDER BY 4

O zaman hedef veritabanı üzerinde mevcut örnek için hata elde ediyor olacağız. Bu hata mesajı çeşitli veritabanları için farklılık gösterecektir. Bu hata mesajı bize kullanılan tablonun üç kolondan oluştuğunu bildiriyor. Bu sayede UNION için kullanacağımız ikinci tabloda maksimum kaç adet kolon kullanmamız gerektiğini öğrenmiş oluyoruz.

UNION Operatörü kullanarak bilgi toplama bize bazı kısıtlar vermektedir;

1. Iki select cümlesinin yapısı aynı olmalıdır. Yani, her iki SQL sorgusu aynı sayıda kolon sayısına sahip olmalı ve bu kolonların veri tipi aynı ya da bir biri ile uyumlu olmalıdır [19].
2. Enjeksiyon yapmak için eklediğimiz UNION sorgusunun anlamlı sonuçlar döndürebilmesi için veritabanında bulunan tablo ve kolon isimlerinin bilinmesi gerekmektedir. Kör SQL Enjeksiyonu yöntemi 2. kısıt için kullanılabilir [19].

UNION operatörü kullanarak SQL Enjeksiyonu zafiyetini tespit ederken aşağıdakı adımlar takip edilmelidir;

1. Öncelikle ilgili istekdeki SELECT cümlesinde kolon sayısı tespit edilmelidir. Bunun için NULL veya ORDER BY kullanılabilir. NULL herhangi veritipine dönüştürülebildiğinden UNION tabanlı enjeksiyonda kolon sayısını deneme yaparak bulabiliriz. Sorgumuz hatasız çalışırsa ve geriye NULL kelimesini veya boş metin alanları içeren ek bir satır dönerse gerekli kolon sayısını bulmuş oluruz [3].

‘UNION SELECT NULL - -

‘UNION SELECT NULL, NULL - -

‘UNION SELECT NULL, NULL, NULL - -

...

ORDER BY operatorü de aynı NULL gibi deneme yapılarak kolon sayısının bulunmasında kullanılır. Bu adımda da deneme hata mesajı alınana kadar devam ettirilir. Kolon sayısı aşılmadığı sürece farklı kolonlara göre sıralanmış orijinal sorgu sonuçları görüntülenir.

‘ORDER BY 1 - -

‘ORDER BY 2 - -

‘ORDER BY 3 - -

…

2. Kolon sayısını bulduktan sonra sıra istediğimiz bilgileri elde etmek için metin veri tipinde bir kolon tespit etmektir. Kolon sayısını bildiğimiz için birinci kolondan başlayarak son kolona kadar her bir NULL’u ‘test’ veya başka karakter ile değiştirerek hata almadan ‘test’ metnini döndüren kolon(lar) tespit etmektir. Metin tipi içeren kolonlar veritabanından elde edilecek bilgileri toplamak için kullanılabilir [23].

‘UNION SELECT ‘test’, NULL, NULL - -

‘UNION SELECT NULL, ‘test’, NULL - -

‘UNION SELECT NULL, NULL, ‘test’ - -

Tek bir metin tipinde kolon tespit edilse bile bu alan farklı bilgilerin birleştirilmesi şeklinde kullanılabilir. Örneğin, ORACLE'da farklı kolon bilgileri aşağıdaki gibi aynı kolon altında birleştirilebilir:

username||':'||password

MS-SQL'de birleştirme işlemi + (%2b olarak encode edilen) operatorüyle yapılmaktadır. Örnek olarak tespit ettiğimiz metin tipli alanda veritabanının versiyonunu gösterebiliriz;

MS-SQL ve MySQL için: 'UNION SELECT @@version, NULL, NULL - -

Oracle için: '*UNION SELECT banner, NULL, NULL FROM v$version* - -sorguları kullanılabilir. Oracle veritabanında her SELECT cümlesinin FROM özelliği bulunması gerekir. Bu amaçla tablo ismi olarak global olarak erişilebilen DUAL tablosu kullanılabilir [18]. Örneğin:

'UNION SELECT NULL FROM DUAL - -

Yukarıdakı örnekte veritabanı versiyon bilgisini öğrenmek kritik bir bilgi sayılabilir. Ancak asıl değerli bilgi 2. kısıtta belirtmiş olduğumuz tablo ve kolon isimlerinin bulunmasıdır.

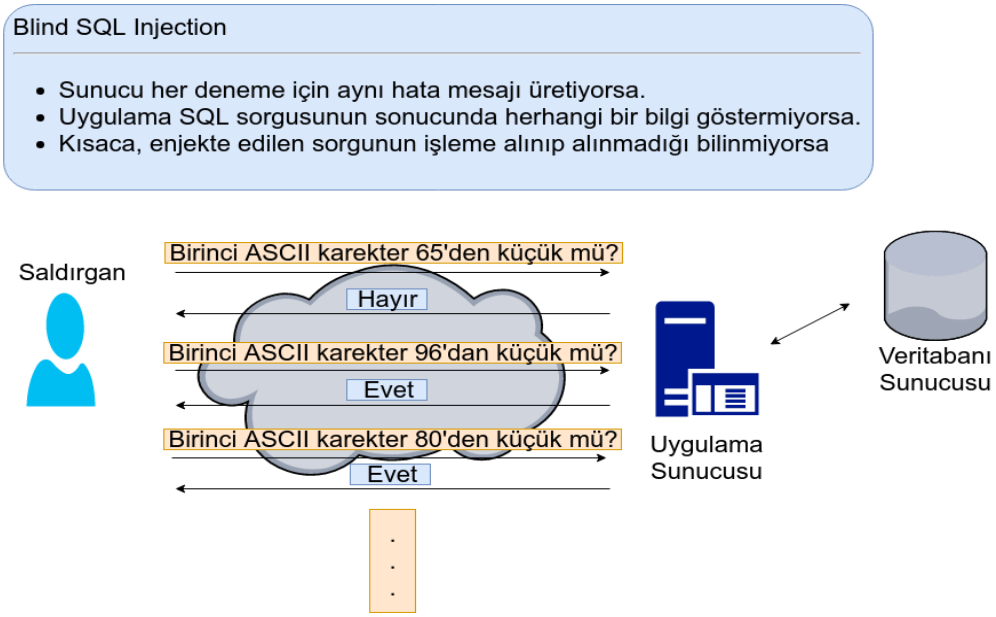
Kör SQL enjeksiyonu

Günümüzdeki uygulamaların çoğu hata mesajlarını ayrıntılı olarak göstermek yerine özelleştirilmiş bir hata mesajı (500 server error) veya sayfası döndürmektedir [20]. Bu durumda hata mesajlarından yola çıkılarak SQL enjeksiyonu zafiyetinin tespiti ve istismarı yöntemleri yetersiz kalmaktadır. Tam da bu noktada Kör SQL Enjeksiyonu teknikleri devreye giriyor. Kör SQL Enjeksiyonu, uygulamanın verdiği hata mesajlarına bakmak yerine zafiyet barındıran sayfaların/servislerin yapılan manipülasyonlara verdiği cevapların **doğru/yanlış (true/false)** olması ile ilgilenir. O yüzden normal SQL Injection saldırılarına göre daha zordur. Nedeni ise herhangi bir SQL hatası veya sonucu döndürmemesi ve sadece bir HTTP status döndürmesidir. Bu saldırı türünde işlem sonuçları, uygulama geliştiricinin özel bir hata sayfası oluşturması ile birlikte veritabanı hakkında bilgi göstermemesi ile oluşmaktadır. İşlem sonucu, özel olarak oluşturulmuş HTTP status içeren hata sayfaları olacaktır [21].

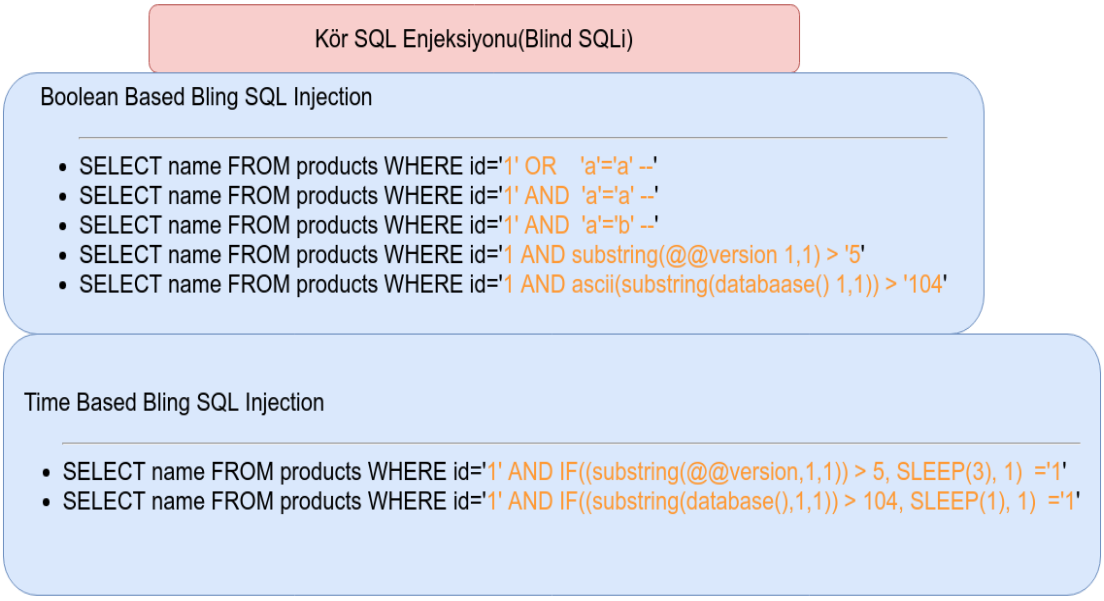
Bu saldırı tekniği iyi bir veritabanı bilgisi gerektiriyor. Kör SQL enjeksiyonunda boolean değer dönecek sorgular sunucuda çalıştırılır, cevapları analiz edilip anlamı çözülür. SQL sunucularında veritabanı adını çekmek için SELECT database () komutu kullanılmaktadır. İlgili komutu SQL Server üzerinde farklı yollarla çalıştırıp sonuçlarını inceleyelim:

Çizelge 3.2 : SQL enjeksiyon saldırılarında kullanılan sorgular

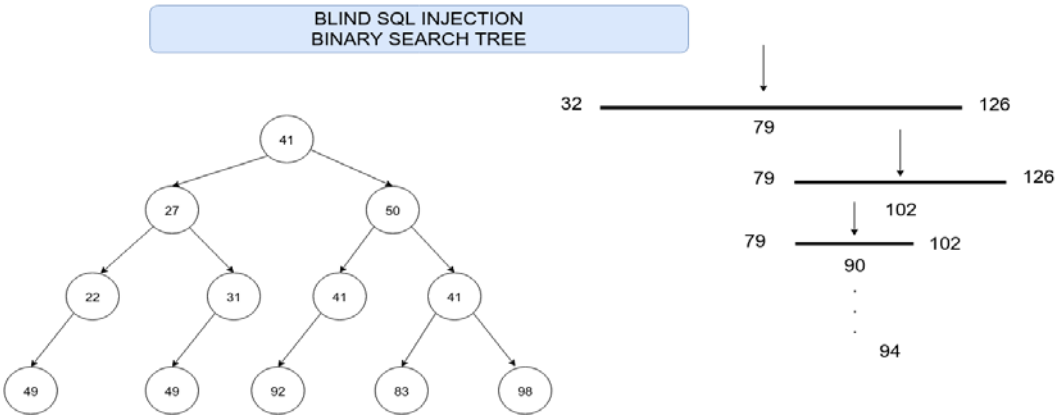
Sorgu	Çıktı	Yorum
SELECT db_name()	hale	Veritabanı İsmi
SELECT SUBSTRING ((SELECT db_name ()),1,1)	h	Veritabanı isminin ilk karakteri
SELECT ASCII (SUBSTRING ((SELECT db_name()),1,1))	104	Veritabanı isminin ilk karakterinin ASCII değeri



Şekil 3.26 : Kör SQL enjeksiyonu zafiyeti senaryosu



Şekil 3.27 : Boolean ve Time Based Kör SQL enjeksiyonu



Şekil 3.28 : Kör sql enjeksionunda arama ağacı

Hata tabanlı SQL enjeksiyonu

Bu zafiyetin tespiti için veritabanı hata mesajının ekrana basılıyor olması yeterlidir. Bazı uygulamalarda hata mesajları kullanıcıların görebileceği şekilde olmayıp HTML yorum satırı içerisinde saklanıyor olabilir. Dolayısıyla hata mesajı tarayıcıda gözükmese bile isteğe karşı gelen cevabın HTML kaynak kodu incelendiğinde hata mesajı görüntülenebilir.

Zaman Tabanlı SQL Enjeksiyonu

Şu ana kadar ele aldığımız SQL Enjeksiyonu zafiyetleri, alınan hata mesajları, uygulamanın doğru/yanlış gibi bir izlenim bırakması sonucu tespit edilmekteydi. Fakat kimi zaman uygulamalar bu davranışları sergilemezken yine de SQL Enjeksiyonu zafiyetine sahip olabilirler. Yani aslında bir zafiyet söz konusu iken

yokmuş gibi görünebilir ancak bu aldatıcı olacaktır. Zaman Tabanlı SQL Enjeksiyonu zafiyetinde, yapılan isteğe karşı sunucudan dönen cevabın içeriği yerine cevabın dönüş süresi göz önünde bulundurulur [19]. Bu dönüş süresi test parametrelerinde kullanılan zaman ile neredeyse aynı değerde (+1,2 saniye sapma olabilir) olmalıdır [19]. Bir nevi Kör SQL Enjeksiyonuna benzemektedir. Çünkü Kör SQL Enjeksiyonunda yapılan isteğin sonucunda kullanıcının doğru/yanlış içerik görmesiye ilgilenilirken, Zaman Tabanlı SQL Enjeksiyonunda yapılan isteğe karşın cevabın dönüş süresinin belirtilen saniyeden fazla veya az olması ile ilgilenilmektedir. Bu zafiyetin tespiti için uygulamanın kullandığı veritabanının zaman içerikli fonksiyonları / komutları tahmin edilmelidir [26]. Örneğin, MSSQL veritabanları için WAITFOR DELAY komutu yardımıyla veritabanına belirtilen süre kadar bekleme yaptırmak mümkün olabilmektedir [19]. Eğer istenilen süre kadar cevap bekletiliyor ise uygulama üzerinnde injection'a uygun bir ortam olduğu anlaşılabilir.

WAITFOR DELAY '0:0:20'; (20 saniye bekle)

Çizelge 3.3 : SQL enjeksiyon zafiyeti ile database bilgilerinin alınması

Sorgu	Çıktı	Yorum
SELECT user	dbo	Veritabanı kullanıcısı.
SELECT SUBSTRING ((SELECT user,1,1))	d	Veritabanı kullanıcı isminin ilk karakteri.
SELECT ASCII (SUBSTRING ((SELECT user),1,1))	100	Veritabanı kullanıcı isminin ilk karakterinin ASCII değeri.
SELECT 1 IF (ASCII (SUBSTRING ((SELECT user),1,1))) = 100 WAITFOR DELAY '0:0:20'	1	Veritabanı kullanıcı isminin ilk karakterinin ASCII değeri 100 ise 20 san bekle ve ekrana 20 san gecikmeli olarak 1 döndür.
SELECT 1 IF (ASCII (SUBSTRING ((SELECT user),1,1))) = 97 WAITFOR DELAY '0:0:20'	1	Veritabanı kullanıcı isminin ilk karakterinin ASCII değeri 97 değilse ekrana hiç bekleme yapmadan 1 döndür

Istismar adımları yapılacak sql sorguların dönüş sürelerine bakılarak karar verilir.

3.5.3.2 Farklı SQL ifadelerinde SQL enjeksiyonu zafiyeti

Şu ana kadar SQL Enjeksiyonu zafiyetini hep SELECT sorgularında tespit ettik, ancak DELETE/INSERT/UPDATE vb. gibi SQL sorgularında da SQL Enjeksiyonu zafiyeti bulunmaktadır ve otomatize araçların bu sorgulardaki zafiyeti tespit etmesi

SELECT sorgularına göre daha zor olduğundan dolayı testlerin manuel olarak yapılması gerekecektir.

Insert: Veritabanının ilgili tablolarına herhangi bir veri eklemek için kullanılır. Örneğin; yeni bir kullanıcı hesabı oluşturduğumuzda, sitelere yorum eklediğimizde, alış veriş sepetlerine ürün eklediğimizde arka tarafda ilgili verileri uygulamadan alıp veritabanına eklemek için insert komutu kullanılır [19]. INSERT işleminde SQL Enjeksiyon saldırısı genellikle values kısmında yer alır.

Update: Veritabanındakı herhangi verileri güncellemek için kullanılır. Update için örnek vermek gerekirse kullanıcı bilgilerinin değiştirilmesini, alış veriş sepetindeki ürün miktarının değiştirilmesini vb. gibi işlemleri gösterebiriz [19]. Update sorgusunda – yorum satırı kullanırken dikkatli olmak gerekmektedir. Örnekle göstermek gerekirse;

UPDATE CREDITCARD SET name = ‘ “ +name+” ’ WHERE USERID = 1

Yukarıdakı SQL sorgusu SQL Enjeksiyonu zafiyetine sebep olacak name parametresini içermektedir. Eğer bu sorguya aşağıdaki gibi bir parametre gönderecek olursak;

a’--

Ilgili sorgumuz aşağıdaki gibi olacaktır:

UPDATE CREDITCARD SET name = ‘a’-- WHERE USERID = 1

-- yorum satırı olduğundan CREDITCARD tablosundakı tüm name değerleri a olarak değiştirilecektir. Bu yüzden – kullanırken dikkatli olmalı ve her ihtimali düşünmeliyiz.

Çünkü – yorum satırı kendinden sonrakı kısımları dikkate almadığından tablonun tüm kayıtları üzerinde değişiklik yapılır ve buda veri bütünlüğünün bozulmasına neden olur.

Delete: Veritabanı üzerinde veri silme işlemi gerçekleştirmek için kullanılmaktadır. Eğer bu aşama da uygulama SQL Enjeksiyonu zafiyeti barındırıyor ise yapılacak olan injection saldırısı ile veritabanı üzerindeki tüm kayıtlar silinebilir [19]. Bu zafiyetin var olduğuna kanaat getirmek için ise silinen veriyi daha sonra SELECT

sorgusunda listelemek istediğimiz de veritabanı üzerinde kayılar bulunamaz ise ilgili zafiyetin olduğuna kanaat getirebiliriz.

INSERT ve UPDATE sorgularına göre kullanımına azami dikkat gösterilmelidir. Örneğin kredi kartı kaydı silinmesini göstermek gerekirse;

http://www.example.com/delete.jsp?cmd=delete&cardId=30 OR 1=1

Yukarıda gösterdiğimiz GET isteğine karşın uygulama arka tarafda veritabanı sunucusu üzerinde aşağıdaki sorguyu çalıştıracaktır:

DELETE FROM CREDITCARD WHERE userid = 1 AND cardid = 30 OR 1=1

Bu sorgu çalıştırıldığında WHERE sonrası her zaman TRUE döneceği için CREDITCARD tablosunun tüm verileri silinmiş olacaktır. O yüzden de kullanımına dikkat edilmelidir.

3.5.3.3 SQL saldırı zafiyetinin otomatizasyonu

Şimdiye kadar gösterdiğimiz örneklerde basit veriler bile ciddi sayıda istek yapılarak elde ediliyordu. Veritabanı içerisindeki tüm bilgilerin alınması istenilen durumlarda bu istek sayılarını manuel olarak yapmak ciddi iş ve zaman yükü getireceğinden ilgili zafiyeti otomatize etmek için araçlar kullanılmalıdır. Bu araçlardan biri de SQLMAP'tir [19]. SQLMAP, sızma testi çalışmaları sırasında SQL Enjeksiyonu zafiyetinin tespiti ve istismarı için kullanılan ve Python ile yazılmış açık kaynak kodlu bir araçtır. Normalde

Linux üzerinde çalışmaktadır fakat Python ile yazıldığından bir çok işletim sistemi üzerinde de çalışabilir. Çoğu Veritabanı Yönetim sistemleri desteği bulunmaktadır [21]. Diğer araçlardan farkı;

- Desteklediği SQL Enjeksiyonu türlerinin çok olması;
- Tüm veritabanı verisinin alınması;
- Deneme yanılmalar için içerisinde sözlükler bulundurması;

Bu aracın nasıl çalıştığı konusuna gelince, istekde belirtmiş olduğumuz parametrelere göre hedef web sitesi üzerinde içerisinde mevcut olan komponentleri deneyerek açık arar. Örnek vermek gerekirse;

python sqlmap.py -u "http://www.example.com/index.jsp?id=1" -f –banner

ilgili istekde eğer id parametresi zafiyet barındırıyor ise, o zaman bize veritabanı sürüm bilgisini vermiş olacaktır (Oracle için) [21]. Bu araçtan başka, Havij, Netsparker, sqlninja araçları da kulanılabilir [3].

3.5.3.4 SQL enjeksiyonu zafiyetinin çözümü

SQL Enjeksiyon'dan korunmak için 3 başlıca yöntem vardır [23]:

- Parametrized Queries (Prepared Statement)
- Stored Procedure
- Çıktı Kodlama

Çıktı kodlama yöntemi şayet diğer 2 yöntemin kullanımının mümkün olmadığı durumlarda kullanılmalıdır. Bu yöntemlere ek olarak *Girdi Denetimi* uygulanması da güvenliği artıracağı için tavsiye edilir.

String Ekleme

```
String username = Session["username"];
String password = Session["password"];
String query = "SELECT * FROM users WHERE username='"+username+"' AND
password='"+password+"'";
SqlCommand command = new SqlCommand(query, DataConnection);
SqlDataReader dataReader = command.ExecuteReader();
```

Parametrik Sorgu

```
String query = "SELECT * FROM users WHERE username=@username AND
password=@password;";
command.Parameters.AddWithValue("username", Session["username"].toString());
command.Parameters.AddWithValue("password", Session["password"].toString());
SqlDataReader dataReader = command.ExecuteReader();
```

Şekil 3.29 : SQL enjeksiyonu zafiyetinin çözümü

PreparedStatement

```
String username = request.getParameter("username");
String password = request.getParameter("password");
String query = "select * from USERS where username=? and password=?"

PreparedStatement pstmt = connection.prepareStatement(query);
pstmt.setString(1, username);
pstmt.setString(2, password);

ResultSet rs = pstmt.executeQuery();
```

Şekil 3.30 : SQL enjeksiyonu zafiyetinin Prepared Statement ile çözümü

Stored Procedure

```
String username = request.getParameter("username");
try{
    String query = " { call sp_getUserBalance(?) } ";
    CallableStatement cs = connection.prepareCall(query);
    cs.setString(1, username);

    ResultSet rs = pstmt.executeQuery();
}catch(SQLException e){
    //error handling
}
```

Şekil 3.31 : SQL enjeksiyonu zafiyetinin stored procedure ile çözümü

3.5.4 Siteler arası betik çalıştırma (XSS)

XSS- Cross Site Scripting zafiyeti, hemen hemen her uygulamada karşılaşılabilecek çok eski bir zafiyetlerden biridir. TOP 10 listesinin 2. sırasındadır [2]. Bir uygulamanın kullanıcıdan (saldırgan) gelen girdileri (zararlı kodlar) denetlemeden ve özel karakterleri kodlamadan diğer kullanıcılara göndermesi sonucu kullanıcıların web tarayıcılarında bu zararlı script/ betik kodların çalıştırılması sonucu ortaya çıkar [23]. Betik koda örnek olarak CSS, HTML, Javascript vb. ile hazırlanmış kod parçacıklarını gösterebiliriz. Ağırlıklı olarak javascript kullanılmaktadır.

Bu zafiyet istismar edildiği zaman çoğunlukla kurban tarafından fark edilmemektedir. XSS açığı uygulamanın zafiyeti olup 3 gruba ayrılır [23]:

- Reflected (*Non-Persistent*)
- Stored (*Persistent*)
- DOM Based

Bunların birçok ortak özellikleri olsada, tespiti ve istismarı arasında önemli farklar vardır.

XSS örnek olarak şu risklere neden olabilir:

1. Kullanıcıların cookie'lerine erişerek oturum bilgilerinin çalınması ve böylece bu kullanıcıların hesaplarına izinsiz erişim;
2. Web sayfasının içeriğini değiştirme;
3. XSS Worms (MySpace Samy)
4. Port Tarama
5. Javascript Keylogger yükleme

XSS saldırı araçları için:

- BeEf [3]
- Shell on the Future [3]

XSS zafiyet tarama araçları için:

- Burp [3]
- ZAP [3]
- Netsparker [3]

kullanılabilir.

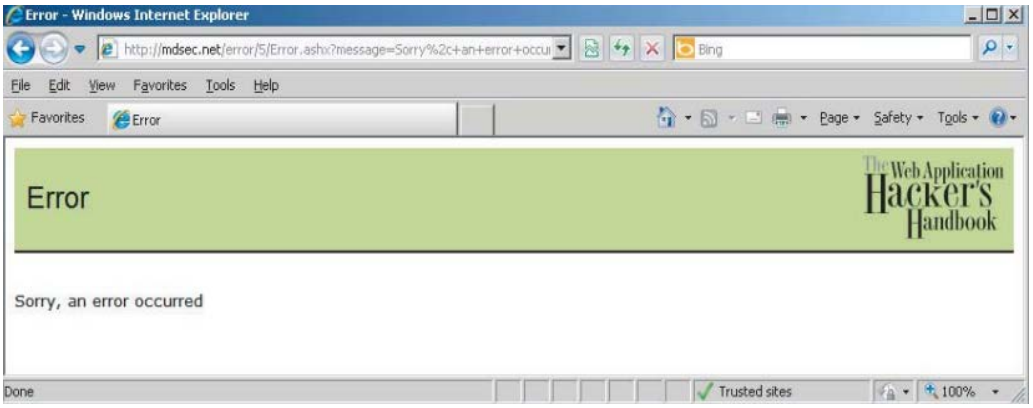
3.5.4.1 Reflected XSS

Reflected XSS zafiyeti, XSS zafiyetleri arasında en çok karşılaşılan türdür ve çok yaygın bir örnek olarak, uygulamanın kullanıcılara veri girmesine ve girdiği verilere göre oluşturulmuş içeriği görüntüleyebilmek için kullandıkları dinamik sayfaları gösterebiliriz. Bu duruma en iyi örnek arama kutucukları gösterilebilir [24]. Kullanıcıdan girdi olarak alınan bilgi yani potansiyel olarak zararlı kod parçacığı veritabanına yazılmadığı için sadece bir defa internet tarayıcısı tarafından çağrılması ile kullanıcıya sunulur. Bu zafiyet türünde saldırgan XSS zafiyeti içeren bir link hazırlar ve bu linki kurbanın kendi tarayıcısında çalıştırmasını ister. Kurban bu linke tıklarsa, zafiyete yakalanmış olur. Kullanıcı herzaman bu linke tıklayacak kadar saf olmayabilir. Böyle kullanıcılar üzerinde saldırının başarılı olması için URL kısaltma (t.co/Vgyf6h7) servisleri kullanılır.

Tespiti: Tipik bir örnek göstermek gerekirse, dinamik bir sayfanın mesaj metni içeren parametreyi alıp daha sonra bu metni kullanıcıya geri döndürmesini gösterebiliriz [3]:

Örnek olarak, hata mesajını döndüren aşağıdaki URL'i dikkate alırsak;

http://mdsec.net/error/5/Error.ashx?message=Sorry an error occurred



Şekil 3.32 : XSS zafiyetinin tespiti [3]

Döndürülen sayfanın HTML kaynak koduna bakıldığında uygulamanın, URL'deki zafiyet içeren message parametresinin değerini kopyalayıp hata sayfasında uygun bir yerde gösterdiğini göre biliriz.

Sorry, an error occurred

Kullanıcıdan alınan girdinin sunucunun cevabında HTML içerisine eklenmesi Reflected XSS açığının olduğuna dair işaretlerden biridir (istismar şuan için zarar verecek noktada olmasa da gerçekleşmiştir) ve hiçbir filtreleme yapılmıyorsa uygulama kesinlikle savunmasızdır diyebiliriz.

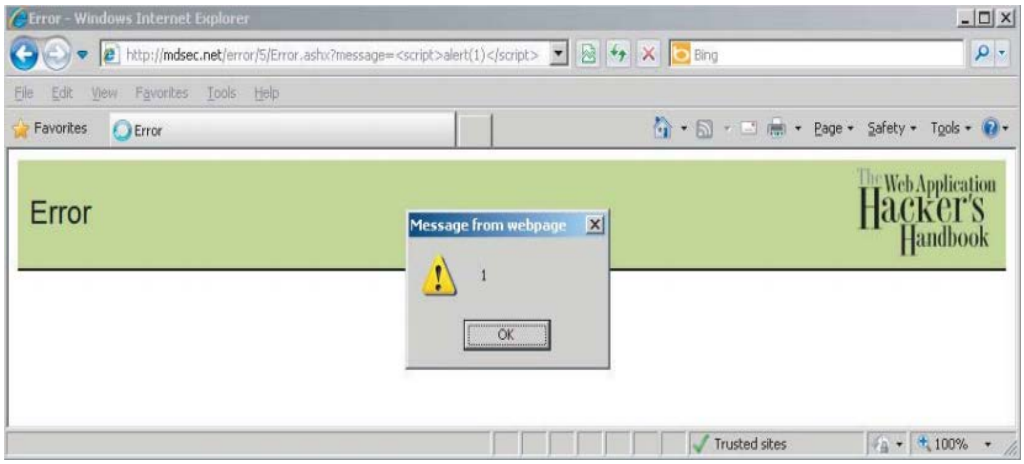
Aşağıdaki URL ilgili test parametresi yerine artık istemci (client) tarafında çalıştırılan Javascript kod parçacığını message parametresine ekleyerek hata mesajını pop up içerisinde göstermek için hazırlanmıştır:

http://mdsec.net/error/5/Error.ashx?message= <script>alert(1)</script>

İstekde bulunulan bu URL orijinal mesajın yerine aşağıdaki kod parçasını içeren HTML sayfa döndürecektir:

<script>alert (1) </script>

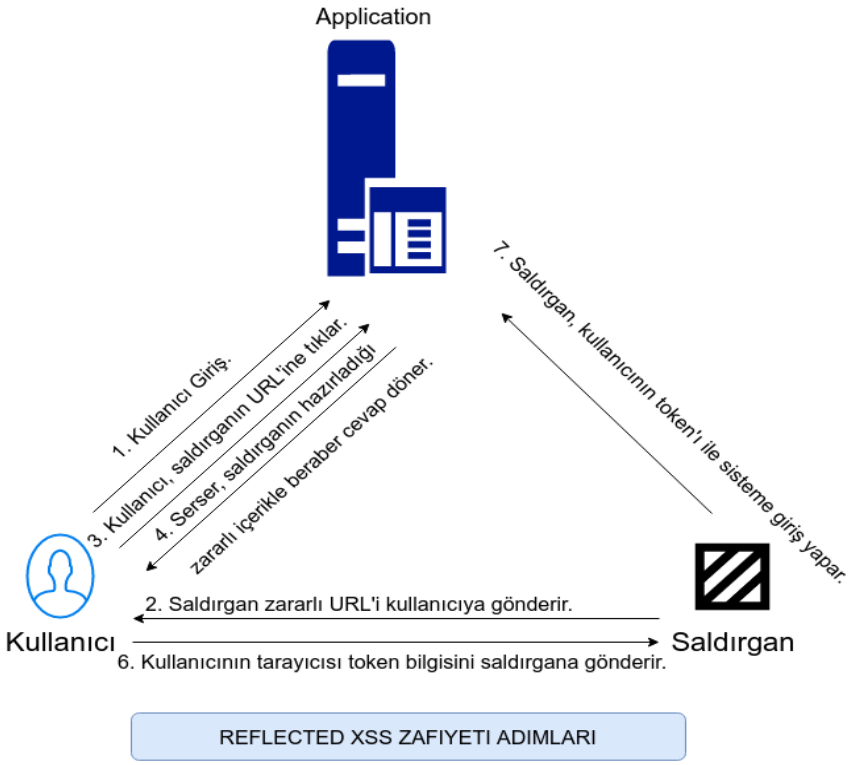
Geri dönüşde tarayıcı ilgili sayfayı işlediğinde, girdi herhangi bir kontrolden geçirilmez ise tarayıcı tarafından script etiketleri okunup, alert metodu çalıştırılacak ve aşağıdaki gibi popup mesajı görüntülenecektir:



Şekil 3.33 : XSS ile zararlı kod parçacığı çalıştırma [3]

XSS istismarının kanıtı; Web uygulamaların çoğunda bu çeşit XSS bulgusu yaklaşık olarak %75 civarındadır [24]. Bu XSS açığı Reflected yani yansıtılan XSS adlanır, çünkü bu açığın istismarında URL'de gömülü Javascript içerisinde hazırlanmış olan kullanıcı girdisi HTML karakterlerinden ayrıştırılmadan yada encode edilmeden isteği yapan herhangi bir kullancıya olduğu gibi yansıtılır. Bu saldırı tipi tek bir request/response üzerinden gidildiği için first-order-XSS de adlanır [22]. Reflected XSS zafiyeti testlerinde önemli olan nokta gönderdiğimiz test parametresinin bize gelen cevap içerisinde aynı şekilde olup olmadığına (yansımasına) bakmaktır. Ancak unutmamak lazım ki her yansıma zafiyet anlamına gelmez. Örnek olarak, arama kutucuğu içerisine girdiğimiz metnin bize hemen isteğimizin ardından gelen cevap içerisinde bir yerlerde gösteriliyor olması gibi.

Istismarı: XSS açıkları, farkli şekillerde uygulamanın kullanıcılarına saldırarak istismar edile bilir. Bu saldırılardan en basiti, saldırganın kimlik doğrulaması yapmış kullanıcının oturum bilgisini (session token) ele geçirmesidir [25]. Saldırgan kullanıcının oturumunu çalarak kullanıcının yetkili olduğu tüm veri ve işlevselliğe erişim sağlaya bilir. Bu saldırının nasıl gerçekleştiğini aşağıdaki resim ile gösterebiliriz:



Şekil 3.34 : XSS zafiyetinin istismar senaryosu

1. Kullanıcı (kurban) normal olarak uygulamaya giriş yapıyor ve uygulama kullanıcının tarayıcısına oturum tanımlayıcısı (session token) içeren cookie gönderiyor:

Set-Cookie: sessId=184a9138ed37374201a4c9672362f12459c2a652491a3

2. Bazı araçlarla saldırgan kullanıcıya daha önceden hazırlamış olduğu Javascript içeren URL'i gönderiyor ve kurbanın bu URL'i çağırmasını sağlıyor (bu URL kurbana e-posta yoluyla da gönderile bilir) [3]:

http://mdsec.net/error/5/Error.ashx?message=
<script>var+i=new+Image;+i.src="http://mdattacker.net/"%2bdocument.cookie;
</script>

3. Kurban kendisine saldırgan tarafından gönderilmiş olan URL'e tıklıyor.
4. Sunucu kurbanın isteğine URL içinde bulunan saldırganın hazırladığı zararlı kod parçası ile cevap döner.
5. Zararlı kod parçasının bulunduğu cevap kurbanın tarayıcısında çalışır.
6. Saldırgan tarafından hazırlanmış olan zararlı kod parçası aşağıdaki gibidir:

var i=new Image; i.src="http://mdattacker.net/"+document.cookie;

Bu kod parçası kurbanın tarayıcısının saldırgan tarafından oluşturulmuş olan domain adresine (http://mdattacker.net) istek yapmasına neden olur. Bu istek hedef uygulama için kurbanın geçerli oturum tanımlayıcısını (session token) barındırmaktadır:

GET /sessId=184a9138ed37374201a4c9672362f12459c2a652491a3 HTTP/1.1

Host: mdattacker.net

7. Saldırgan mdattacker.net sitesine gelen istekleri izler ve çaldığı oturum cookie'sini kullanarak aynı kurban gibi hedef uygulamaya kullanıcı oturumuyla bağlanır.

Bazı uygulamalar kullanıcının her siteye ziyaretinde kimliğini yeniden doğrulamak için kalıcı cookie içermektedir (remember me işlevselliği gibi) [6]. Bu durumda, 1. adım yani kullanıcının giriş yapması gereği ortadan kalkacaktır. Durum böyle iken kurban hedef uygulamada aktif olmasa bile saldırı gerçekleştire bilecektir.

XSS açıkları durum böyle olunca daha da kritik hale gelmektedir. Saldırının yapılma nedenleri ve salrıdığının nasıl yapıldığı tarayıcıların uyguladığı aynı kaynak politikası ile açıklanabilir [26].

3.5.4.2 Aynı kaynak politikası (SOP)

Bu politika tarayıcılar tarafından erişilen farklı kaynakların birbirlerinin metotlarına ve özelliklerine erişmelerini engeller [26]. SOP, bir uygulamanın kaynaklarını ya da bir web sayfasının elemanlarını başka bir uygulamanın nasıl kullanacağını belirten kurallar bütünüdür.

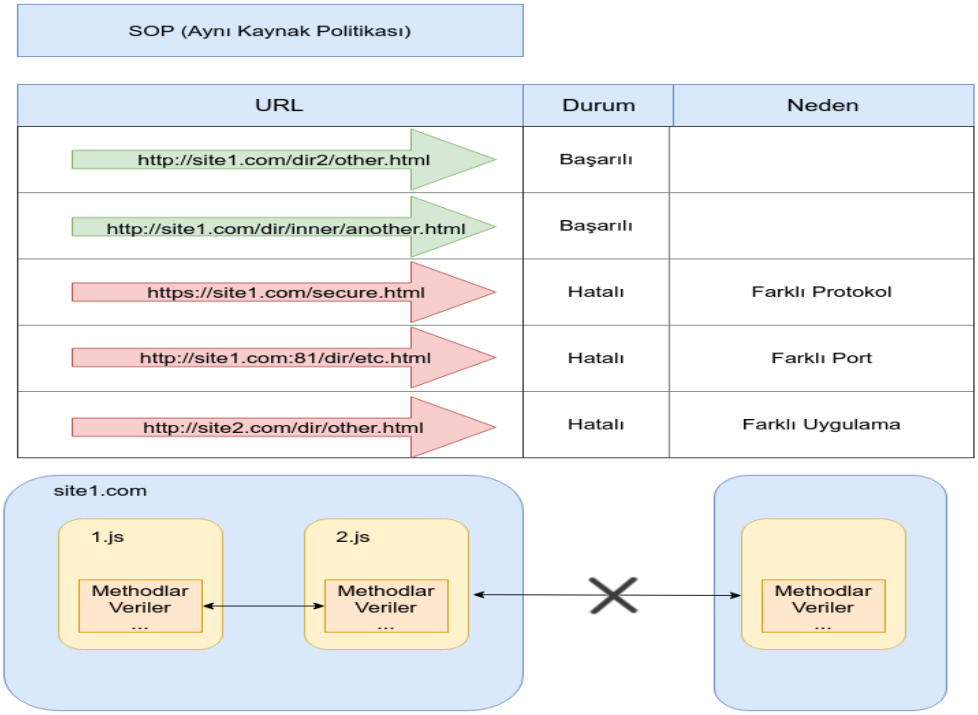
Same Origin = Protokol + Domain + + Port

Politikanın ana maddeleri şunlardır:

1. Bir site, başka bir siteye herhangi bir istek göndere bilir, fakat o siteden gelen veriyi işleyemez [26].
2. Bir site, başka bir sitede js kodu çalıştırabilir. Bunun nedeni js'in veri değil kod içermesidir. Bu yüzden siteler arası erişim hassas verilen sızmasına neden olmaz [26].

3. Bir site başka bir siteye ait olan Cookie veya DOM verilerini okuyamaz veya değiştiremez [26].

Same-Origin Policy mevcut olmasaydı, ve habersiz bir kullanıcı kötü niyetli bir web sitesini gözden geçirseydi, o site üzerinde çalışan script kod veriye erişe bilir ve işlevselliği değiştirebilirdi. Bu sayede kötü niyetli web sitesi kullanıcının online bankasından para transferi gerçekleştire bilir, kişinin email'ni okuya bilir ve kullanıcı online alışveriş yaptığında kredi kartı bilgilerini ele geçirirdi. Bu nedenle tarayıcılar sadece aynı kaynaktan alınmış içerikle etkileşimin bu türüne izin kısıtlamaları uyguluyorlar. Aynı kaynaktan kastımız, protokol, alan ismi ve port kümesidir [26]. Bir tanım yapmak gerekirse, iki kaynağın aynı kaynak olması için gerekli şart, bu değerlerin (protokol, alan ismi ve port) aynı olmasıdır. Yani, bir kaynaktan yüklenen script veya dökümanların diğer kaynaktan yüklenen dökümanların özelliklerini değiştirmesini engeller.



Şekil 3.35 : Aynı Kaynak Politikası

Tarayıcılar farklı kaynaklardan aldıkları içerikleri farklı contextler'de tutarak SOP kuralını uygularlar. Bu contextler arasında bariyer olduğu için bir birlerini etkilemezler, karşı alanlara müdahele edemezler. Bundan dolayı, mdattacker.net domaininde yerleşen script "document.cookie"'yi sorguladığında, mdsec.net tarafından oluşturulan cookie'leri ele geçiremeyeceği için oturum çalma saldırısı

başarısız olacaktır [3]. Kurban saldırganın hazırladığı zararlı URL'i tıkladığında uygulama JS içeren sayfayı geri döndürür. Tarayıcı dönen sayfadakı js'leri sanki aynı domainden gelmiş gibi algılayarak çalıştırır. Çalıştırılan zararlı JS mdsec.net ile aynı kaynakta olduğu için cookie ve benzeri bilgilere saldırganın erişimi olacaktır.

3.5.4.3 Stored / Persistent XSS

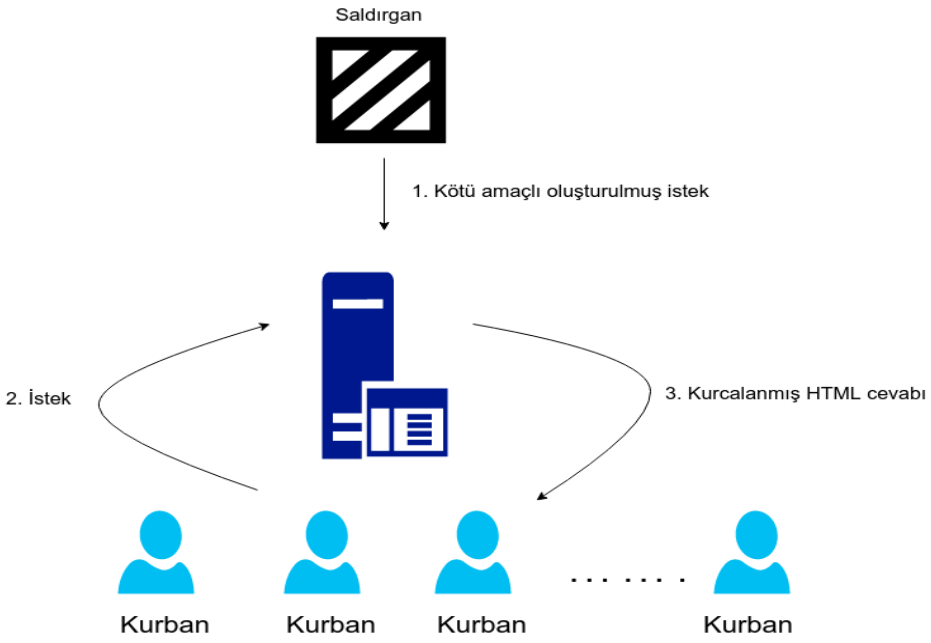
XSS açıklarının diğer bir türü de Stored XSS'dir. Reflected XSS'den farklı olarak kullanıcının bir linke tıklamasına gerek yoktur. Stored XSS içeren bir web sayfasını (örn, forum) ziyaret etmesi yeterlidir. Dolayısıyla Reflected XSS açığına göre daha risklidir. Bu zafiyet türünde saldırgan zararlı kodunu veritabanı veya sunucuya enjekte eder [27]. Bu yüzden de uygulama veritabanında keydedilen bütün girdiler stored XSS'in hedefindedir ve test edilmelidir. Dosya yükleme fonksiyonları ile HTML dosyası yüklenebiliyorsa bu yolla da XSS saldırısı uygulanabilir.

Tespiti; Stored XSS zafiyetinin tespiti Reflected XSS ile benzerlik göstermektedir. Fakat Stored XSS açıkları güvenlik açısında Reflected XSS açıklarına göre daha kritikdir [28]. Bunun 2 nedeni bulunmaktadır:

1. Yansıtılan XSS zafiyetinde saldırgan hazırlamış olduğu zararlı url'i kullanıcıya iletir ve kullanıcınının url'i tıklamasını bekler. Saklanan XSS açıklığında ise veritabanı veya sunucuya enjekte edildiği için zararlı url oluşturma gerek kalmaz ve kullanıcının kendi rızasıyla sadece url'e girmesini beklemektedir [28].
2. XSS açıklıklarının oturum bilgilerini çalmak amacıyla kullanılabilmesi için kullanıcının uygulamaya giriş yapmış olması gerekmektedir. Yansıtılan XSS saldırılarında ise saldırgan kullanıcıyı yine giriş yapmaya yönlendirmek zorundadır. Saklanan XSS saldırılarında ise kullanıcı sisteme giriş yapmış durumundadır. Ayrıca zararlı etki veritabanı veya sunucuya enjekte edildiği için birden fazla kullanıcı kurban olabilir. Hatta zararlı etkiyi sistem yöneticisinin görüntülemesi durumunda uygulamaya yönetici haklarıyla erişim de mümkün olacaktır [28].
3. Genellikle pek bilinmeyen zafiyetlerden biri de JavaScript ve HTML kodlarının içinde bulunduğu dosyaların sunucuya yüklenebilmesi ve bu dosyaların kullanıcılar tarafından görüntülenmesi ile kodların

çalıştırılmasıdır. Explorer kullanarak bir kullanıcı doğrudan bir Jpeg dosyasını çağırırsa (yani bir HTML dosyası içindeki <img> etiketi içinde yer almadan) Internet Explorer bu dosyanın HTML kodu içermesi durumunda bunu HTML kodu olarak yorumlayacaktır. Eğer uygulama JPEG dosyasının içeriğinin gerçekten resim içerip içermediğini kontrol etmezse bu tür bir saldırıyı gerçekleştirmek mümkün olacaktır.

Stored XSS açıklarının Reflected XSS'de gösterdiğimiz örnek gibi oturum bilgisinin çalınmasını aşağıdaki gibi gösterebiliriz [29]:



Şekil 3.36 : Stored XSS zafiyeti senaryosu

Gerçek hayattan bir örnek:

MySpace Worm -2005 (Samy Worm) [28]

Gerçek Hayattan Bir Örnek
Kod Eklenmeden önce : 73 arkadaş
1 saat sonra : 73 arkadaş - 1 arkadaşlık isteği
7 saat sonra : 74 arkadaş - 221 arkadaşlık isteği
1 saat sonra : 74 arkadaş - 480 arkadaşlık isteği
1 saat sonra : 518 arkadaş - 561 arkadaşlık isteği
5 saat sonra : 2503 arkadaş - 6.3 3 arkadaşlık isteği
Birkaç sn sonra : Your Profile is Down for Maintenance

Şekil 3.37 : Samy Worm örneği

Samy, profiline bir kod ekler. Samy'nin profilini ziyaret eden herkes;

1. onu arkadaş olarak ekler,
2. bu kodu kendi profiline ekler.

Bu saldırı tipi 2 istek üzerinden gitmektedir:

1. Saldırgan uygulamanın depoladığı zararlı kod içeren veriyi postalıyor.
2. Kurban saldırganın verisini içeren sayfayı görüntülüyor, script kurbanın tarayıcısında çalıştığında zararlı kod çalıştırılmış oluyor. Saldırı iki adımdan oluştuğu için second-order XSS olarak da adlandırılır.

3.5.4.4 DOM tabanlı XSS

DOM (*Document Object Model*) Tabanlı XSS zafiyeti, girdilerin sunucu tarafından değil de istemci tarafında JS nesnelerinden elde edilip eklendiği durumlarda meydana gelir [29]. Dolayısıyla, diğerlerinden farklı olarak DOM tabanlı XSS'te, ilgili parametrenin sunucuya gönderilmesine gerek yoktur yani sunucudan kullanıcıya gönderilen HTML zararlı bir kod içermez. DOM tabanlı XSS Reflected XSS'e benzemektedir. Her ikisinde de kullanıcı saldırganın ürettiği URL'i tıklamak zorundadır. Tarayıcı DOM objesini çağırır ve DOM objesi zararlı kodu çalıştırır. DOM Based XSS oldukça zor bir zafiyet türüdür. Hatta bu zafiyetin tespiti için otomatize araçlar bile bazen yetersiz kalmaktadır. Manuel olarak yapılan

denemelerin sayısı fazla olduğundan harcanacak zaman da diğer XSS türlerine göre oldukça fazladır [28]. Bu zafiyet türünü diğerlerinden ayıran en büyük fark gönderilen saldırı parametrelerinin sunucuya ulaşmasına gerek kalmadan zafiyetin gerçekleşiyor olmasıdır. Bu zafiyet türünün en güzel yanı istemci tarafında olduğu için sunucuya herhangi bir paket veya zararlı kod gönderilmemektedir. Bu nedenle sunucu tarafında tespiti mümkün değildir.

Tespiti; DOM temelli XSS saldırısının nasıl gerçekleştirilebileceğine dair bir örnek vermek gerekirse, sunucunun döneceği sayfanın içinde aşağıdaki betik olduğunu varsayalım [3]:

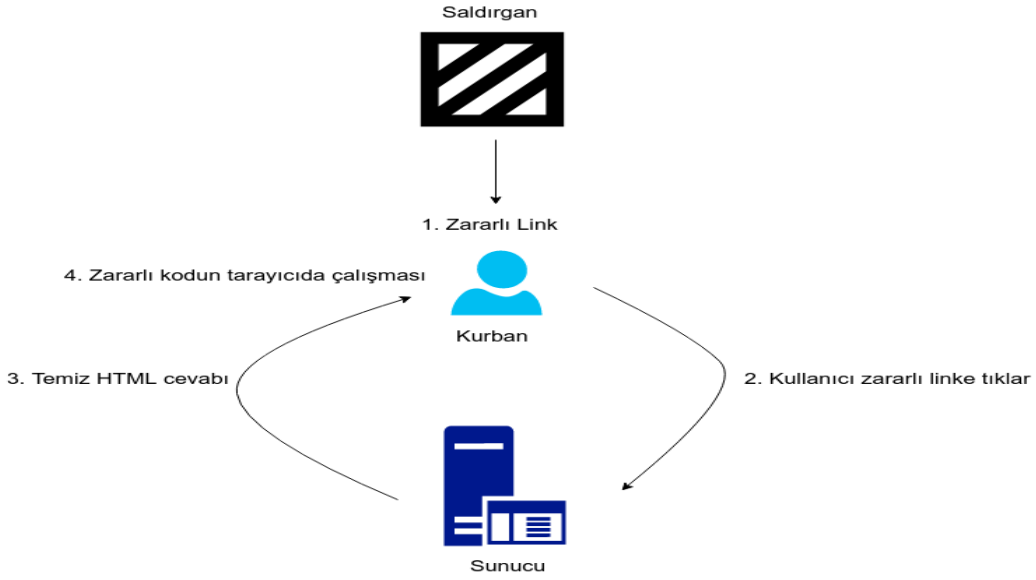
```
<script>
var a = document.URL;
a=unescape(a);   // bu kod URL'deki özel karakterleri encode eder.
document.write(a.substring(a.indexOf("message=") + 8, a.length));
</script>
```

Bu betik URL içinde bulunan mesaj parametresinin içeriğini dinamik olarak sayfanın HTML kodunun içine yazmaktadır. Aşağıdaki gibi bir URL üretirsek bu sayede istediğimiz bir betiğin çalışmasını sağlamış oluruz [3]:

https://www.example.com/error.php?message=<script>alert('xss');</script>

3.5.4.5 Zafiyetin istismarı

Kullanıcının oturum bilgisini çalmayı amaçlayan DOM Tabanlı XSS saldırısını adımları aşağıda gösterilmektedir:



Şekil 3.38 : DOM tabanlı XSS zafiyetinin istismarı

3.5.4.6 XSS zafiyetinin çözümü

XSS zafiyetinin çözümü için genel olarak kullanılan iki kural vardır:

1. Girdi Denetimi
2. Çıktı Denetimi

Ilk yöntem hemen hemen her uygulama güvenliği probleminin temel çözüm noktasıdır. Girdi denetimi, kullanıcının girmiş olduğu girdilerin kontrol edilerek içeri alınması yani uygulamanın beklentisi dışında gelen girdilerin eleneceği anlamına gelmektedir [6]. Tüm kullanıcı girdileri dışa aktarılmadan önce özel karakterlerin kodlanması (escaping) gerekmektedir.

- <script> → < script>

Sadece HTML entity kodlaması yeterli değildir. Bu durumda çıktı denetimi (output encoding) yapılmalıdır. Hatta girdi denetimi yapılmasa bile çıktı denetiminin yapılması XSS zafiyetine karşı çözüm sağlaya bilir. HTTP cevaplarında CSP (*Content Security Policy*) header bilgisi kullanılmalıdır. CSP XSS'e karşı geliştirilmiş W3C standardıdır [28]. CSP, HTTP cevaplarında bulunursa web tarayıcılar JS kodu çalıştırmak için daha kuralcı ve titiz davranırlar. Çıktı kodlama için varolan kütüphanelerden yararlanılmalıdır [10].

Bunlara;

- ESAPI Encoder (encodeFor metotları)

DOM – tabanlı XSS'den korunmak için;

1. innerHTML, outerHTML, document.write gibi HTML rendering metotları kullanılmamalıdır. Kullanılması zorunlu durumlarda gerekli çıktı kodlaması yapılmalıdır:

   element.innerHTML =
   "<%=Encoder.encodeForJS(Encoder.encodeForHTML(untrustedData))%>"

2. JSON için JSON.parse() metodu kullanılmalıdır.
3. Dinamik içerik oluşturmak için document.createElement(), element.SetAttribute(), element.appendChild() gibi metotlar kullanılmalıdır.
4. Eval () gibi kod çalıştırma metotları kullanılmamalıdır.

Eval () metodu string tipinde bir değer almaktadır. Ancak aldığı bu değer çalışabilen bir js kodudur. Bu metot aldığı string parametresinden dinamik olarak bir js kodu üretip çalışmasını sağlar [29].

Örneğin;

document.write (eval("3+2")) // sonuç 5 döner.

3.5.5 Phising

Password (Şifre) ve Fishing (Balık avlamak) kelimelerinin birleşmesiyle oluşan ve türkçeye *oltalama* olarak çevrilmiş bir saldırı türüdür [28]. Genellikle XSS açıklarından yararlanmaktadır. Bu saldırı türünde amac, kurbanların hesap numaraları ve şifreleri, kredi kartı numaraları, internet bankacılığında kullanılan kullanıcı kodu ve şifrelerini çalmaktır. Saldırgan bu yöntemle bankalar (www.bank.com) ,alış-veriş, oyun , sosyal paylaşım ağları vb. siteleri taklit edip kopya web sayfaları hazırlayarak kurbanlara sanki gerçek siteleri kullanıyormuş gibi izlenim vererek kurbanı bu sahte sitelere yönlendirmeyi içermektedir [6]. Kopya siteler taklit edilecek ilgili sitelerin bire bir kopyasının oluşturulması ile mümkün olmaktadır. Örnek verecek olursak; saldırgan bir online kontör sitesi açar ve insanlar online kontör almak istediklerinde kredi kartı bilgilerini girerler ve herhangi bir satış yapmadan saldırgan kurbanlarının kredi kartı bilgilerini toplar [6]. Kurban e-posta

adresine gelen URL'in tamamına dikkat ederse bu sayfanın sahte bir sayfa olduğunu anlayıp phisihing için kullanıldığını fark edebilir.

3.5.5.1 Phising saldırıları tespiti

1. Saldırgan ilk önce hedeflediği kaynağın sahte kopyasını oluşturarak ilk adımı atar. Burada saldırganın hedeflediği siteye kurban kullanıcı adı ve parola bilgilerini kullanarak girmelidir.
2. Saldırgan elinde mevcut olan e-posta adreslerine veya hedeflediği kişilere gönderdiği e-postalarla kurbanlarını hazırladıkları sahte siteye yönlendirirler.
3. Kurbanların sahte siteye girip istenilen bilgileri paylaşmasıyla saldırgan hedefine ulaşmış olur.

3.5.5.2 Phising saldırının önlemi

- Bilgisayarınız'da ki işletim sistemini güncel tutmaya çalışın. Ayrıca güncel ve kaliteli bir anti virüsler programı kullanın.
- Gelen e-posta'nın kimden geldiğinden emin değilseniz dikkate almayınız. Unutmayın hiç bir kurum veya kuruluş mail yoluyla sizden kişisel bilgilerinizi istemez. Mail kutunuzda spam iletilere karşı filtreler oluşturun.
- Mail yoluyla URL adreslerine emin değilseniz tıklamayınız. Özellikle yukarıda belirtildiği gibi bankalar, sosyal paylaşım siteleri, alış-veriş siteleri gibi bağlantılara kendiniz adres barına yazarak giriniz. Phishing saldırılarında en yaygın kullanılan yöntem budur.
- Güvenli olmadığını düşündüğünüz ağlardan kesinlikle elektronik işlem gerçekleştirmeyin. Genel ağlardan uzak durmaya çalışın, bunlar; Havaalanı, metrolar vb yerlerdeki halka açık ağlar olabilir.

3.5.6 Siteler arası istek sahteciliği (CSRF)

Cross Site Request Forgery, kullanıcının giriş yaptığı bir uygulamada kullanıcının bilgisi olmadan, kullanıcının oturum bilgisiyle zararlı isteklerde bulunma saldırısıdır. XSRF-CSRF içinde XSS bulunduran ve XSS alanında görülen güvenlik açığıdır [42]. CSRF, "*Sleeping Giant*" yani uyuyan dev olarak da adlandırılır [42], çünkü web üzerindeki birçok sitede bu saldırı türüne karşı korunma mevcut olmayıp geliştiriciler

tarafından arka plana bırakılmış ve oluşturacağı sonuçlar önemsenmemiştir [2]. Günümüzdeki en yaygın güvenlik açığıdır.

XSS'de kullanıcı bilgilerini saldırgan js kodlarıyla ele geçirmeyi amaçlarken XSRF'de kullanıcı bilgilerini kendi tarafından değiştirmeyi planlamaktadır. Ve gönderilen tek bir link ile bunu sağlamak mümkündür. Bu saldırı türü ile yapıla bilecekler arasına, kullanıcıların internet banka hesabından para transferi yapması, e-mail hesabının ayarlarını değiştirmesi, yönetim paneline sahip olduğu sitelerden bir üyeyi silmesi, bir alış veriş sitesinden birşeyler alması, sosyal medya hesabından başka kullanıcılara mesaj göndermesi gibi pek çok senaryo dahil edilebilir. Yani, yapılabilecekler web uygulamasının kullanıcıya verdiği yetkiyle doğru orantılıdır [40]. CSRF zafiyetini içeren bir uygulama, giriş yapıp oturum açmış kullanıcının (kurban) web tarayıcısının saldırganın istediği herhangi bir HTTP isteğini (örn, para transferi, parola değiştirme) göndermesine imkan sağlar. Bu zafiyet, önceden oturum açılmış web sayfalarına erişilmek istendiğinde web tarayıcıların oturum ID'sini HTTP isteğine otomatik eklemesi özelliğini temel olarak hedef alır [23].

3.5.6.1 XSS ile CSRF arasındakı fark

CSRF'nin oluşmasında web sitesinin kullanıcıya ait web tarayıcısına duyduğu güvenin suistimali söz konusuyken; XSS'nin oluşmasının temelinde ise kullanıcının, web sitesine duyduğu güvenin suistimali yatmaktadır. Bir CSRF saldırısının meydana gelmesi için session (oturum) kimliğinin doğrulanmış olması gerekmektedir [23]. XSS'te ise buna gerek yoktur. Açık bulunan web sitesinde girdiler üzerinde herhangi bir doğrulama ya da escaping işlemi yapılmadığı için ortaya çıkar.

Örnek bir CSRF işleyişi aşağıdaki gibidir;

- Kurban kullanıcı, zafiyeti içeren online bankacılık uygulamasında oturum açar.
- Bu açığı bilen saldırgan, internette sık ziyaret edilen bir forumda şöyle bir girdi oluşturur.

- Kurban kullanıcı, forumdaki bu girdiyi ziyaret ettiğinde haberi olmadan bankacılık uygulamasına istek gönderilir ve havale işlemi gerçekleştirilir.

Tespiti; Hedef sitede CSRF açığının olup olmadığını kontrol etmek için ilk önce herhangi bir web proxy aracı (Paros, Burp Suit, Owasp CSRF Tester, Netsparker, Fiddler) kullanılarak siteden yapılan istekler yakalanır ve kaydedilir [23]. Daha sonra bu kaydedilen istekler web proxy aracının repeater özelliği ile aynı tarayıcıda tekrarlandığında daha önce karşılaşılan aynı sonuçlar alınıyorsa sitenin ilgili kısmında CSRF saldırısına karşı zafiyet vardır sonucuna varılabilir [40]. Bu sitenin tüm sayfaları için veya kullanıcı için önemli sayfalarda denenerek sitedeki tüm açıklıklar bulunabilir.

Istismarı: XSRF-CSRF saldırısını Java ile yazılmış alışveriş sitesi üzerinde küçük bir senaryo ile gösterelim. Site adresi http://www.avm.com olsun.

Genel olarak herhangi bir uygulamaya giriş yapılırkenki işlemler aşağıdaki gibidir:

Kullanıcı ilk önce uygulamaya girer, kullanıcı adı ve şifresini yazarak login olmaya çalıişır. Uygulama geçerli kullanıcıyı tanırsa bu isteğe cevap vermeye başlar.

Ilgili uygulama yapılan her isteğe tarayıcıda bulunan cookie bilgisini de ekler -ki bu da kullanıcının oturum bilgisidir ve ilgili uygulama kullanıcının kim olduğu ancak bu yolla bilebilir. HTTP protokolü uygulamaya gönderilen isteğin uygulamayı açtığımız pencereden mi yoksa farklı farklı pencereden mi gittiğini genel itibariyle kontrol etmez [3]. Dolayısıyla ilgili uygulamadayken, farklı bir uygulamaya girildiğinde eğer aynı tarayıcı kullanılıyor ise yani aynı tarayıcı kapsamı (scope) içerisindeyse, ve farklı uygulama ilk girilen uygulamaya istek yapıyor ise, ilgili uygulamaya ait cookie bilgisi de bu isteğe eklenecektir. CSRF zafiyetinin bulunduğu uygulamalar da ve/veya siteler de genelde zafiyetin temelinde istemciden gelen tüm isteğin geçerli bir kullanıcıdan geldiğini varsayması yatmaktadır.

Aşağıdaki örnek avm.com sitesi üzerinden detaylandıralım;

Sitede kullanıcının hesabıyla giriş yaptığını ve alışveriş sepetine birkaç ürün eklediğini düşünelim. Site içerisinde aşağıdaki gibi bir form bulunmakta ve kullanıcılara alışveriş sepetini silebilmesini sağlasın.

www.avm.com

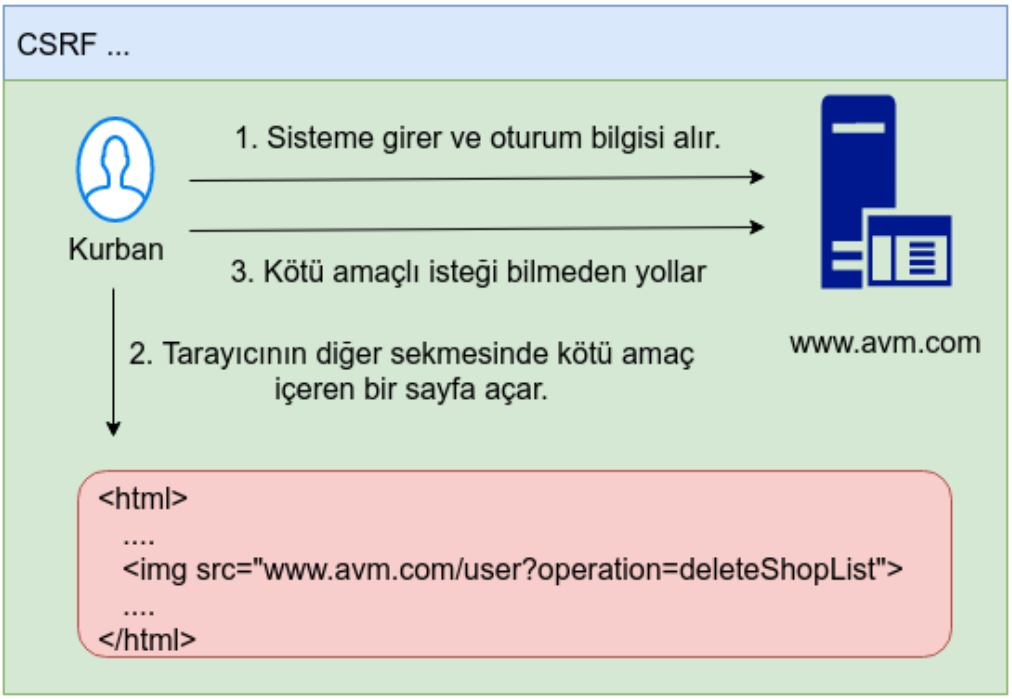
```
<form action="user.php" method="GET">
   <input type="hidden" name="operation" value="deleteShopList"/>
   <input type="submit" value="delete" />
</form>
```

Şekil 3.39 : CSRF zafiyetinin istismarı (1)

```
<html>
   ...
<img style="display: none;" src=www.avm.com/user?operation=deleteShopList />
   ...
<html>
```

Şekil 3.40 : CSRF zafiyetinin istismarı (2)

Saldırgan web sayfasındakı XSRF açığı oluşturan alanı kendine göre düzenleyip zararlı bir link hazırlar ve kullanıcının bu linki ilgili site üzerinden çağırmasını sağlayıp kendisini sisteme giriş yaptırabilir. Saldırganın hazırladığı zararlı link deki içerikte aşağıdaki satırlar bulunabilir; Kullanıcı olan şekilde sisteme login olduktan sonra saldırganın hazırlamış olduğu www.attacker.com/intex.html sayfasını tıklar. Sayfada bulunan img (bazen de bir form) ile birlikte www.avm.com adresine src attribute içerisinde yer aldığı şekli ile kurbandan habersiz istek yapılır. Kullanıcı zaten login olduğu için herhangi bir yetki sorunu oluşmayacaktır. Burada www.avm.com'da herhangi bir CSRF önleyici bulunmadığı için gelen isteğin gerçek kullanıcıdan geldiğini varsayarak işlemi gerçekleştirir. İşlem gerçekleşerek kullanıcının alışveriş sepetindeki ürünler silinmiş olmaktadır. Herşey olağan gibi görünmesine rağmen kullanıcı bilmeden alışveriş sepetini boşaltmış olacaktır. Bu ve benzeri yöntemlerle geçmişte bazı bankaların internet şubelerinde kullanıcının hesabından saldırganın hesabına para aktarılması, youtube gibi siteler de kullanıcıların favori video listesine saldırganın belirleği videoların eklenmesi gibi zafiyetler söz konusu olmuştur. Yukarıdaki tüm akışı aşağıdaki gibi özetleyebiliriz;



Şekil 3.41 : CSRF zafiyetinin saldırı senaryosu

3.5.6.2 CSRF saldırılarına karşı önlemler

Saldırganlar, CSRF saldırıları için hedef olarak çoğunlukla uygulamalarda oturum açmış kullanıcıları seçer. Çünkü, oturum sahibi olmayan kullanıcının sıradan ziyaretçi olacağı ve ziyaterçinin yapacağı işlemi, saldırganın da yapabilmesi olasıdır.

1. Kullanıcının veri okuma işlemleri GET metodu kullanılarak yapılmalı fakat yazma, değiştirme ve silme gibi tüm operasyonlar POST metoduyla yapılmalıdır [38].
2. Yapılan her isteğe karşın HTTP Referer başlığı kontrol edilmelidir. Bu sayede yapılan her isteğin uygulama üzerinden gelip/gelmediği doğrulanmış olur. Fakat bu madde tamamen yeterli değildir. Çünkü XMLHttpRequest ile bu başlık manipüle edilebilir [38]. Ayrıca Referer başlığı kaldırılarak da geçerli istekler yapılabilir. Hatta favorilere (bookmark) eklediğimiz bir sayfa içinde Referer başlığı eklenmeden gönderilebilmektedir.
3. Kritik olan her form, her kullanıcının oturumuna özel rasgele oluşturulmuş CSRF token'i içermelidir [6].

- Kullanıcıdan sunucuya aktarılan bu token'in doğruluğu sunucu tarafında kontrol edilmelidir:

```
<form...><input type="hidden" name="CSRFToken"
    value="56D7865F7907Hf567AA445dE">
```

XSS açığı içeren uygulamalardan CSRF token'inin çalınabileceği göz önünde bulundurulmalıdır.

4. Hayati önem taşıyan işlemler için (örn, sistemde kayıtlı email adresini değiştirme) Re-authentication uygulanmalıdır.

5. URL Rewriting yöntemi kullanılarak URL'lerin geçerli oturumda, karmaşık olarak belirlenmesi sağlanabilir. Yine de tam bir koruma sağlamayacaktır [39].

6. CSRF'ye karşı birçok platformda hazır çözümler bulunmaktadır [38]:

- Struts, SpringMVC, dotNET (*AntiForgeryToken () metodu*)
- OWASP CSRFGuard
- ESAPI User

CSRF Token oluşturma metodu;

```
private String csrfToken =  resetCSRFToken ()
...
public String resetCSRFToken (){
csrfToken = ESAPI.randomizer().getRandomString(8,DefaultEncoder.CHAR_ALPHANUMERICS);
    return csrfToken;
}
```

3.5.6.3 Alınması gereken önlemler

1. Uygulamadan mutlaka logout yapılmalıdır. Yani var olan oturum kapatılmalıdır.
2. Her logout işleminden sonra tarayıcıda cookie'ler silinmelidir.

3. Tarayıcılar için NoScript eklentisi kullanılmalıdır [23].
4. Bilinmeyen e-postalar açılmamalı ve imaj gibi içerikler görüntülenmemelidir.

3.5.7 Web servislerin güvenliği

Web servis, adından anlaşılacağı gibi web üzerinden HTTP protokolü ile hizmet veren ve çoğunlukla XML (*Extensible Markup Language)*, JSON (*JavaScript Object Notation*) standardı ile veri alış verişi yapan, iki uzak cihaz arasındakı iletişimi sağlayan bir haberleşme programlarıdır/yöntemidir/uygulama komponentidir [6]. İçerdiği standartlar sayesinde birbirinden bağımsız sistemlerin, platformların haberleşmesi, mesaj alıp-göndermesi gibi ihtiyaçlardan dolayı geliştirilmiştir. Bu sayede farklı sistemler haberleşebilmektedir ki bu da web servislerin en önemli özelliğidir [34]. Yani, herhangi bir programlama dili ile yazılan bir web servis diğer programlama dillerinin client veya server uygulamaları ile konuşabilir. Web servislerinde veriler XML olarak tutulup SOAP (*Simple Object Access Protocol*) ile bir yerden diğerine taşınır.

Web servislerin iki kullanım şekli vardır:

1. Başka platformda çalışan bir uygulama ile haberleşerek veri alış verişi yapmak;
2. Uygulamanın tekrar tekrar kullanımına ihtiyaç duyduğu özelliklerin web servisler tarafından yapılıp uygulama üzerinden istenilen zamanda kullanılması. Örneğin; döviz kurlarını, hava durumunu anlık öğrenen programlar gibi.

Web servislerin dış dünyaya kontrolsüz bir şekilde açılması beraberinde güvenlik sorunlarını da getirmiştir. Web servislerinde meydana gelen güvenlik zafiyetleri web uygulamalarında meydana gelen zafiyetlerle birbirine benzemektedir.

Web uygulamalarında genel olarak POST ve GET parametreleri manipüle edilmeye çalışılırken web servislerinde kullanılan XML'in aldığı değerler kontrol edilmektedir. Web Servisleri, SOA (*Service Oriented Architecture*) mimarisini temel alırlar [34].

Web Servisleri:

1. Klasik Web Servisleri

2. Web API Web Servisleri

olmakla iki ana kategoride incelenmektedir. Bu iki tipin ortak özelliği genel bir dil olarak XML kullanması ve iletişimi HTTP üzerinden gerçekleştirmesidir.

3.5.7.1 Klasik web servisleri

Klasik web servisleri, XML standartını kullanan ve SOAP mimarisi ile haberleşen servislerdir.

XML

XML *(Extensible Markup Language)* kelimelerinin kısa adıdır. Verilerin düz metin olarak saklanması ve taşınması işlerinde kullanılır. HTML gibi markup dildir fakat HTML veriyi sunmaya yararken XML veriyi taşımak için kullanılır. XML dosyası ağaç yapısına benzemektedir.

Örnek XML dosyası:

```
<kullanıcılar>
    <kullanıcı id = "1">
        <ad>Ali</ad>
        <yas>23</yas>
    </kullanıcı>
    …
</kullanıcılar>
```

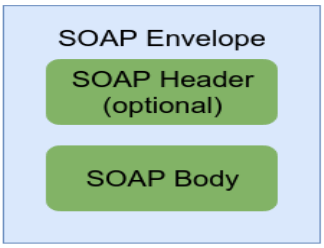
SOAP

SOAP, *(Simple Object Access Protocol)* kelimelerinin kısa adıdır. SOAP, uygulamanın web servislere erişip veri alış verişi yapmasını sağlayan XML tabanlı bir iletişim protokoldur. Yani, SOAP ile ilgili bütün mesajlar XML formatında iletilir ve HTTP üzerinden taşınırlar.

SOAP mesajı 3 şekilde oluşabilir:

1. Metot Çağırımı
2. Cevap Mesajı
3. Hata mesajı

SOAP mesajının yapısı aşağıdaki gibidir:




```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:avm="http://tr.avm.com/">
  <soapenv:Header/>
  <soapenv:Body>
    <avm:getCreditCard>
      <cardId xsi:type="xsd:integer">1</cardId>
      <cardPassword xsi:type="xsd:string">abc1122</cardPassword>
    </avm:getCreditCard>
  </soapenv:Body>
</soapenv:Envelope>
```

Şekil 3.42 : SOAP mesaj yapısı

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Encoding: gzip
Content-Length: 186
Server: Jetty(6.1.26)

<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:avm="http://tr.avm.com/">
  <soapenv:Header/>
  <soapenv:Body>
    <avm:getCreditCard>
      <cardId>1</cardId>
      <cardNo>15156161</cardNo>
      <cardExpiryDate>01-01-2020</cardExpiryDate>
    </avm:getCreditCard>
  </soapenv:Body>
</soapenv:Envelope>
```

Şekil 3.43 : SOAP mesajının cevabı

Envelope- Bütün SOAP mesajlarını içinde bulunduran en üst (root) elemandır. HTML'deki <html>...</html> etiketine benzetebiliriz. İçerisinde *Header* ve *Body* gibi elemanları bulundurur.

Header- HTML'in <head>...</head> etiketine benzetebiliriz. Dolayısıyla meta-data bilgilerini göndermek için kullanılmaktadır. SOAP mesajı içinde bulunması zorunlu değildir. Kullanılacaksa body'den önce kullanılmalıdır.

Body- HTML'deki <body></body> etiketine benzetebiliriz. SOAP mesajının en önemli kısmıdır. Bu bölümde kullanılacak metodun adı ve parametrik bilgileri XML

formatında gönderilir. Cevap mesajında ise metodun geri dönüş değeri bu kısma eklenir [36].

WSDL

WSDL, (*Web Service Description Languag*e**)** kelimelerinin kısa adıdır. XML içeriğe sahip olan WSDL dokümanları dışarı hizmet olarak sunulan bir web servis hakkında teknik detayları barındırırlar. Bu dokümana web servisin interface'i diyebiliriz. Yani, web servisleri, sundukları servisleri WSDL aracılığı ile tanımlarlar ve istemciler bu servislere erişmek için WSDL dosyalarını kullanırlar [34]. Bu da web servisi kullamının önemli avantajlarından biridir. Çünkü oluşturulan web servisleri çok sayıda metot barındırabilir. Bu metotların isimleri, alacağı parametre adları, her parametrede alacağı veri tipi ve döneceği cevabın yapısı vb. Bilgiler WSDL dokümanı yardımı ile yayınlanır.

WSDL dokümanının içeriği 5 ana maddeden oluşmaktadır:

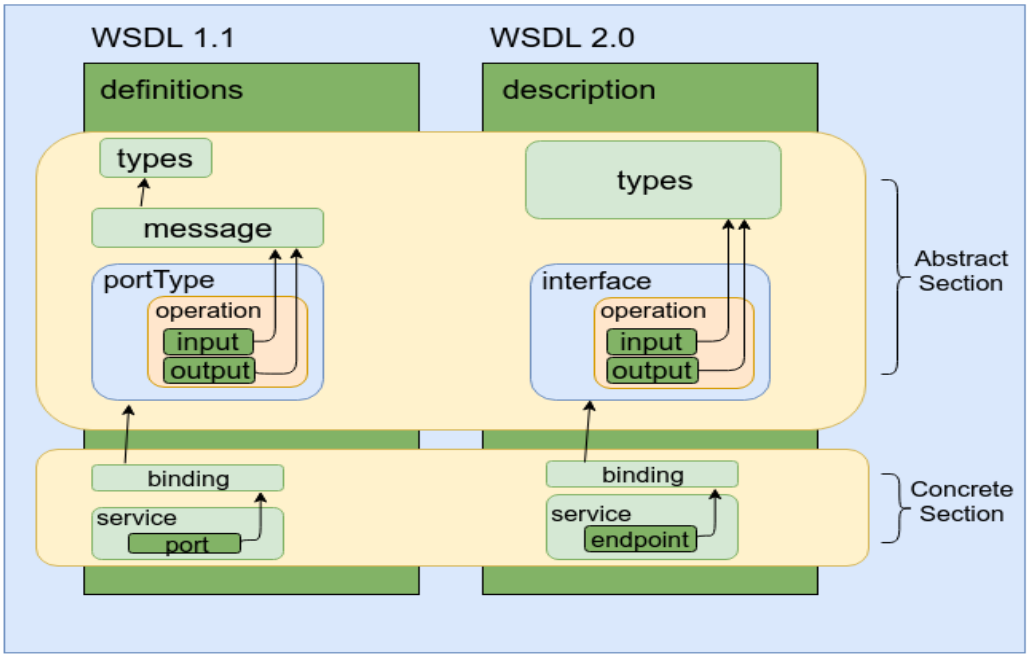
<*types*>: web servis tarafından kullanılan veri tiplerini tanımlamaktadır;

<*message*>: web servisin göndereceği ve alacağı mesajın içeriğini tanımlamaktadır;

<*portType*>*:* web servis ile gerçekleştirilecek operasyonların tanımı yapılmaktadır;

<*binding*>*:* portType'da yapılan operasyona karşılık web servisde çalışacak metotların tanımı yapılmaktadır.

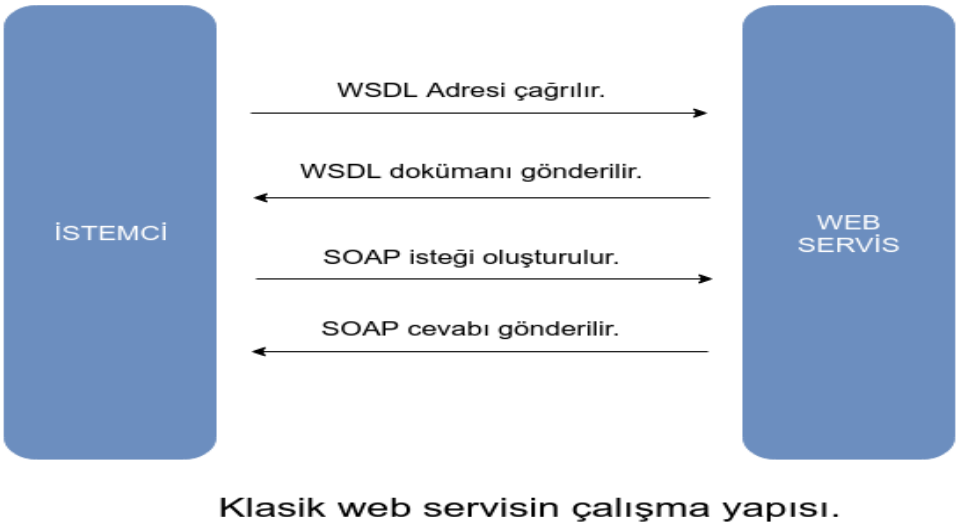
<*service*>: port tanımlarını barındırır. Bu tanımlar binding'de yapılan tanımlara karşılık kullanılacak web servislerin adreslerini içermektedir.



Şekil 3.44 : WSDL version kıyaslaması

SOAP UI

Web Servisi testlerinde kullanılan java ile geliştirilmiş araçlardan biridir. Bu araç yardımı ile web servislerini kontrol edip çağrıları test edebiliriz. Aracın çalışma prensibine değinecek olursak; Web Servisin WSDL adresi verilir, bu sayede web servisin arayüzü görüntülenir, bu arayüz üzerinde olan girdi parametreleri üzerinde değişiklik yapıp servisi çalıştırarak sonuç görüntülenir.



Şekil 3.45 : Klasik web servisinin çalışma mantığı

1. Client WSDL URL'ni çağırır (GET).
2. WSDL dokümanı Client'a gönderilir.

3. Client bu dokümana göre SOAP isteği oluşturur ve sunucuya gönderir (POST).
4. Sunucu da client'a SOAP cevabını döndürür.

Bu işlemlerden 1. ve 2. adım tek seferlik yapılıp 3. ve 4. adımlar defalarca tekrar edebilir.

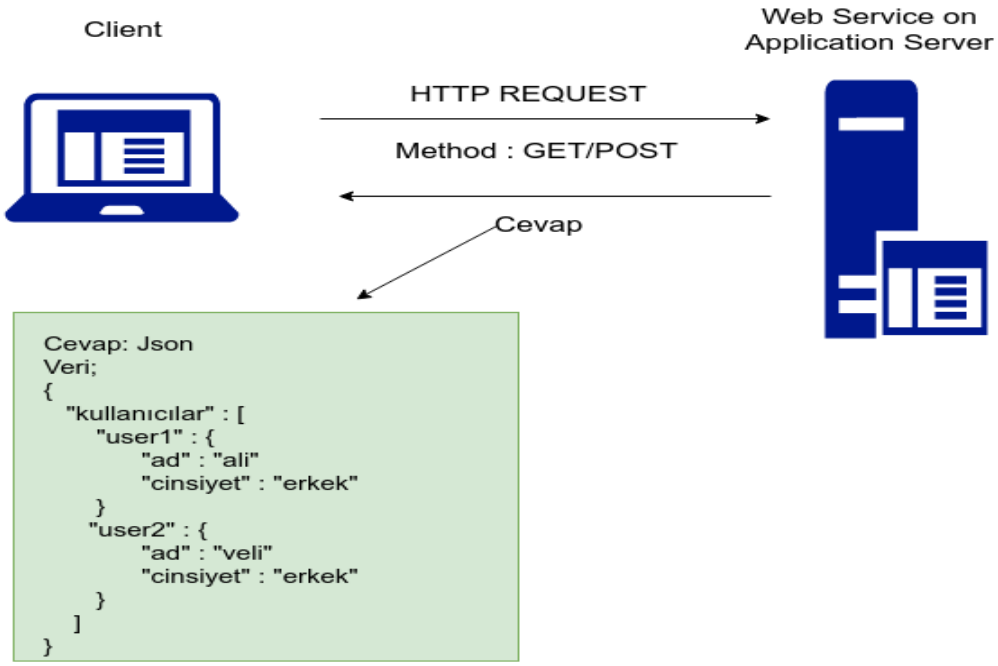
UDDI

UDDI, **(***Universal Description Discovery and Integration***)** kelimelerinin kısa adıdır. Internet üzerindeki web servislerin kayıtlarını tutar ve kayıtlı web servislerini arama özelliği sunar [33]. Saldırganlar açısından UDDI çok yararlı bir kaynaktır.

3.5.7.2 Web API web servisleri

1. Client, GET veya POST talebi ile URL çağırır
2. Web servis XML veya JSON cevabı döndürür.

Aşağıda örnek bir web servis çağrımı görülmektedir:



Şekil 3.46 : Web servisin kullanılması

Web API Web Servisleri, REST (*Representational State Transfer Protocol*) mimarisi ile haberleşen servislerdir. REST mimarisi, client-server arasındakı veri alış verişini SOAP vb. gibi kompleks mimariler yerine HTTP protokolu üzerinden sağlamaktadır

[37]. Esnek ve basit bir yapıya sahiptirler. REST mimarisini kullanan servislere RESTful denilir.

SOAP gibi sıkı standartlara sahip değildir. Aralarındakı farkı ele alacak olursak;

1. REST servislerinin WSDL gibi yapılara ihtiyacları yoktur.
2. SOAP servisleri yapılan isteklere SOAP metotları yardımı ile anlam katarken, REST'de bunun karşılığı URL'de kullanılan resource alanıdır.
3. Yapılan isteğe SOAP'da olduğu gibi illa XML dönülmesi gerekmez. Dolayısıyla istek esnasında dönecek cevabın hangi formatta olmasına karar verilebilir (JSON, XML, text vb.). Şu sıralar genellikle JSON formatı kullanılmaktadır.
4. SOAP servislerine göre istek ve cevap formatlarının esnekliğine göre daha az trafik oluşturur.
5. REST, HTTP metotlarını kullanır (POST, GET, PUT, DELETE).
6. REST servislerinde güvenlik denetimleri yaparken HTTP metotları ile resource kavramı üzerinden uygulanan metotlar iyi analiz edilmelidir.

Klasik yaklaşımda ID'si 100 olan ürünü şu şekilde görebiliriz:

...urunler/goster.jsp?id=100

REST yaklaşımında ise GET talebi aşağıdaki şekilde oluşur:

GET/urunler/100 HTTP/1.1

100 nolu ürünü silmek için:

DELETE/urunler/100 HTTP/1.1

Çizelge 3.4 : SOAP ve REST servislerinin karşılaştırılması

#	SOAP	REST
1	Xml tabanlı protokol.	Mimari tipte protokol.
2	Servert ve Client tarafı Iletişim için wsdl kullanır.	İletişim için xml ve/veya json kullanır.
3	Servisler RPC methodları ile çağrılır.	Servisler url ile çağrılır.
4	Kolay okunabilir cevap dönmez.	Sonuç, kolay okunabilir sade xml ve/veya json içerir.
5	Veri transferi HTTP üzerinden yapılır. Ayrıca SMTP, FTP gibi protokoller de kullanılabilir.	Sadece HTTP üzerinden veri alışverişi yapılabilir.
6	Javascript ile SOAP servisi çağrılabilir ancak bunu yapmak oldukça zordur.	Javascript ile Rest servisleri çağırmak oldukça kolaydır.
7	XML tabanlı protokol olduğundan dolayı performans bakımından REST'in gerisindedir.	SOAP'a göre daha az kod ile daha az kaynak kullanarak daha hızlı iletişim sağlanır.

JSON

JSON, (*Javascript Object Notation*) kelimelerinin kısa adıdır. Programlama dilinden bağımsız ve XML'e alternatif olarak kullanılan JS tabanlı bir veri değişim formatıdır. [34] JSON kullanımındaki amaç, XML'e göre daha küçük boyutlarda veri alıp göndermesidir.

XML ve JSON farkı:

Çizelge 3.5 : XML ve JSON karşılaştırması

XML	JSON

<kullanıcılar>
  <kullanıcı>
  <id>1</id>
  <isim>Ali</isim>
  </kullanıcı>
  <kullanıcı>
  <id>2</id>
  <isim>Hasan</isim>
  </kullanıcı>
</kullanıcılar>
	{
	  "kullanıcılar": [
	  {
	  "id":1,
	  "isim":"Ali"
	  }
	  {
	  "id":2,
	  "isim":"Hasan"
	  }
	  ]
	}

3.5.7.3 Web servis güvenlik testleri ve saldırıları

Web servisleri üzerinde yapılacak testler her ne kadar web uygulamaları ile paralel olsa da doğaları gereği kendilerine özgü adımlar içerebilmektedir.

Bilgi toplama

Bu adımda hedef servis üzerinde girdi noktaları ve bu girdi noktalarının ne tür veri aldığı tespit edilir. Eğer hedef sistem üzerinde herhangi bir bilgi yok ise ilk önce WSDL dökümanı aranmalıdır. Herkesin erişimine sunulan WSDL dosyaları saldırganlar için çok detaylı ve yararlı bilgiler sağlarlar. Bu dosyalar Google Hacking aracılığı ile internet üzerinden aranabilirler [6].

WSDL dökümanı aramak için iki yöntem kullanılır:

1. Arama motorları (Google Hacking)
2. Fuzzing yöntemi

Google Hacking yöntemi ile aramaya örnek olarak;

inurl:wsdl site:avm.com

filetype:wsdl site:avm.com

…

Fuzzing yöntemi tamamen deneme yanılma stratejisine dayalıdır ve örnek vereceğimiz URL adreslerinde uygun değişiklikler yaparak bu yöntem kullanılabilir [33]:

http://www.example.com/<servisadı>

http://www.example.com/<servisadı>.wsdl

http://www.example.com/<servisadı>?wsdl

http://www.example.com/<servisadı>.jsp?wsdl

Web servis güvenlik test aracı olarak ise WSDigger kullanılabilir [33]. Ilgili araç kendisine verilen WSDL dökümanını okuyup ilgili metotların çıkarılması işlemini yerine getirir. Bu sayede ilgili metotların ne tür parametreler aldığı da elde edilebilir.

Güvensiz tasarım

Web Servisleri çeşitli kimlik doğrulama mekanizmaları barındırmaktadırlar. Özellikle Basic kimlik doğrulama yönteminin kullanıldığı gözlemlenmektedir. Kimlik doğrulama için bahsedilen tüm zayıflıklar web servisler için de geçerli olmaktadır. Çünkü hedef web uygulamasını kullanan bir client'ın yapacağı isteklerde kullandığı kimlik doğrulama bilgisini elde edip Base64 ile decode işleminden geçirmek, kullanıcı adı ve parola bilgisinin elde edilmesi için yeterli olmaktadır [33].

Web servis referans saldırıları

XML değişken genişleme (XML Entity Expansion)

XML dokümanı içinde bir çok yerde kullanılması gereken bir veri, değişken olarak tanımlanıp kullanılabilir. Bu değişkenler DOCTYPE içerisinde tanımlanıp ENTITY olarak adlandırılır [6]. Bu saldırı türü, XML parser'ların ilgili referans için kullanmak istedikleri veriyi hesaplarken yaşayacakları bellek problemlerini ortaya çıkarır.

```
<?xml version="1.0" ?>

<!DOCTYPE authors[
   <!ENTITY author "author">
   <!ENTITY author2 "&author;&author;&author;&author;&author;&author;&author;&author;&author;&author;">
   <!ENTITY author3 "&author2;&author2;&author2;&author2;&author2;&author2;&author2;&author2;;&author2;&author2;&author2;">
   <!ENTITY author4 "&author3;&author3;&author3;&author3;&author3;&author3;&author3;&author3;&author3;&author3;&author3;">
   <!ENTITY author5 "&author4;&author4;&author4;&author4;&author4;&author4;&author4;&author4;&author4;&author4;&author4;">
   <!ENTITY author6 "&author5;&author5;&author5;&author5;&author5;&author5;&author5;&author5;&author5;&author5;&author5;">
   <!ENTITY author7 "&author6;&author6;&author6;&author6;&author6;&author6;&author6;&author6;&author6;&author6;&author6;">
   <!ENTITY author8 "&author7;&author7;&author7;&author7;&author7;&author7;&author7;&author7;&author7;&author7;&author7;">
   <!ENTITY author9 "&author8;&author8;&author8;&author8;&author8;&author8;&author8;&author8;&author8;&author8;&author8;">
]>
<authors>&author9;</authors>
```

Şekil 3.47 : XML değişken genişletme örneği

Verdiğimiz örnekte author kısmına a9 değişkeni eklendiğinde ortaya çıkan veri yaklaşık olarak 12GB olacaktır ki bu durumda uygulamanın belleği ilgili veriyi işleyemeyecek (bellek doldurulacak) ve bu istek sonucu uygulama hizmet kesintisi ile karşı karşıya kalacaktır.

External entity attack

Web Servislerine karşı en bilindik saldırılardan biridir. XML dokümanlarının referans tanımları ile kendi içine yerel veya uzak sunucudan veri taşıması sonucu

oluşmaktadır. Harici referans tanımı SYSTEM tanımlayıcısı ile yapılmaktadır [33]. Bu saldırı ile yerel kaynaklardan dosya okunacağı gibi etkin hizmet kesintisi saldırıları da gerçekleştirilebilir ve servis yeni isteklere cevap veremez hale gelir. Örnek verecek olursak;

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE rohit[
   <!ENTITY entityex SYSTEM "file:///etc/passwd">
]>
<abc>&entityex;</abc>
```

Şekil 3.48 : External entity saldırısı

3.5.7.4 Web servis saldırılarına karşı önlemler

1. OWASP TOP 10 risklerine karşı önlem alınmalıdır.
2. Her uygulama güvenliğinde olduğu gibi web servisleri için de girdi denetimi ilk uygulanması gereken önlemlerden biridir. Bunun için XML DoS(*Denial-of-service*) saldırılarına karşı XML mesajlarının geçerliliğini DTD/XML Schema kullanılarak denetlenebilir. Bu dosyalar .XSD uzantılı dosyalardır [33].
3. WSDL dosyaları herkesin erişimine sunulmamalıdır.
4. Gerekmediği sürece servisler UDDI'ye kayıt edilmemelidir.
5. WS-Security protokollerinden yararlanılmalı ve mümkünse XML Security Gateway (*XML Accelerator*) kullanılmalıdır [33].

4. ZAFİYETLERİN UYGULAMA ÜZERİNDE GÖSTERİLMESİ

Tezin konusu olan "Yazılım Güvenliği" günümüzde neredeyse tüm sistemler için geçerli olan bir kavramdır. Tez içerisinde bu kavramların ne oldukları, nasıl tespit edildikleri ve istismar edilme yöntemleri anlatılmıştır. Tez içerisinde "Yazılım Güvenliği" kavramı web üzerinde gösterilmeye çalışılmıştır. Geliştirilmiş olan uygulama içerisinde genelde uygulamaların çoğunda bulunan bir takım zafiyetlere yer verilmiştir. Bu bölümde uygulama içerisinde olan zafiyetlerin tespiti ve istirmarı ele alınacaktır.

4.1 Zafiyetin Tespiti ve İstismarı

Bu kısımda geliştirilmiş olan uygulama üzerinde zafiyetlerin nasıl tespit edilebildiği gösterilecektir.

4.1.1 Kimlik doğrulama zafiyetinin tespiti ve istismarı

Bu bölümde özellikle form tabanlı kimlik doğrulama işlemlerinde yapılan hatalar nedeniyle kullanıcı adı ve şifre bilgilerinin nasıl tespit edildiğini gösteriyor olacağız.

Kimlik doğrulama zafiyetinin tespiti genellikle iki yöntem ile tespit edilir.

Bunlar;

- Sözlük Tabanlı Saldırılar
- Kaba Kuvvet Saldırılar

4.1.1.1 Sözlük tabanlı saldırılar

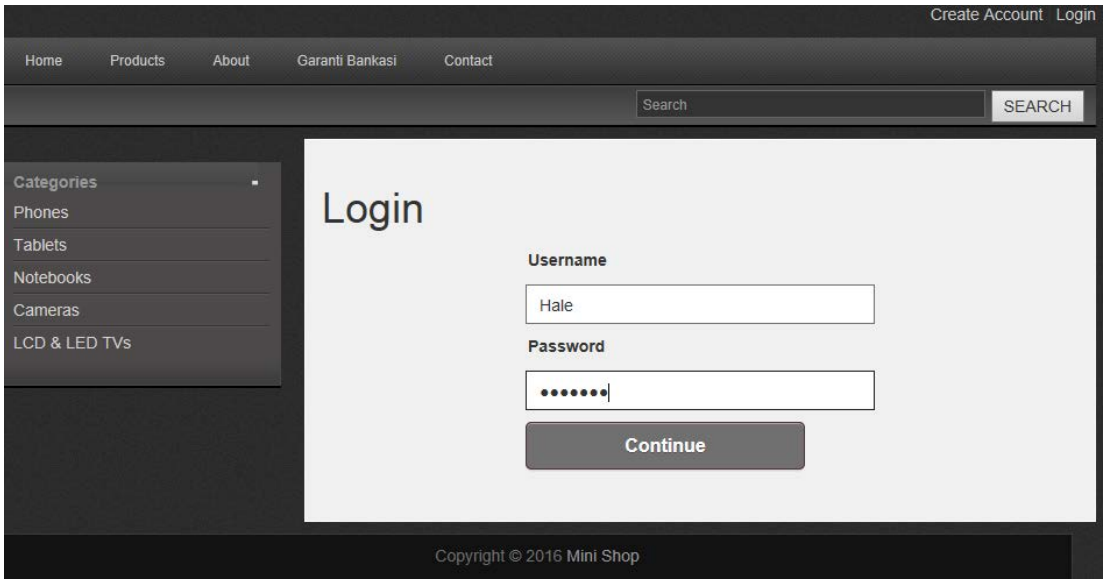
Sözlük tabanlı deneme yanılma saldırılarının belirli bir liste içerisinden yapıldığını öğrenmiştik. Burada geliştirmiş olduğumuz uygulamanın giriş formu, daha önce hazırlamış olduğumuz kullanıcı adı ve parola bilgilerini içeren bir liste yardımı ile istismar edilecektir. İstismar ederken Burp Suit otomatize aracı kullanılacaktır.

İstismar edeceğimiz giriş sayfası için aşağıdaki liste oluşturulmuştur.

Çizelge 4.1 : Sözlük tabanlı saldırı listesi

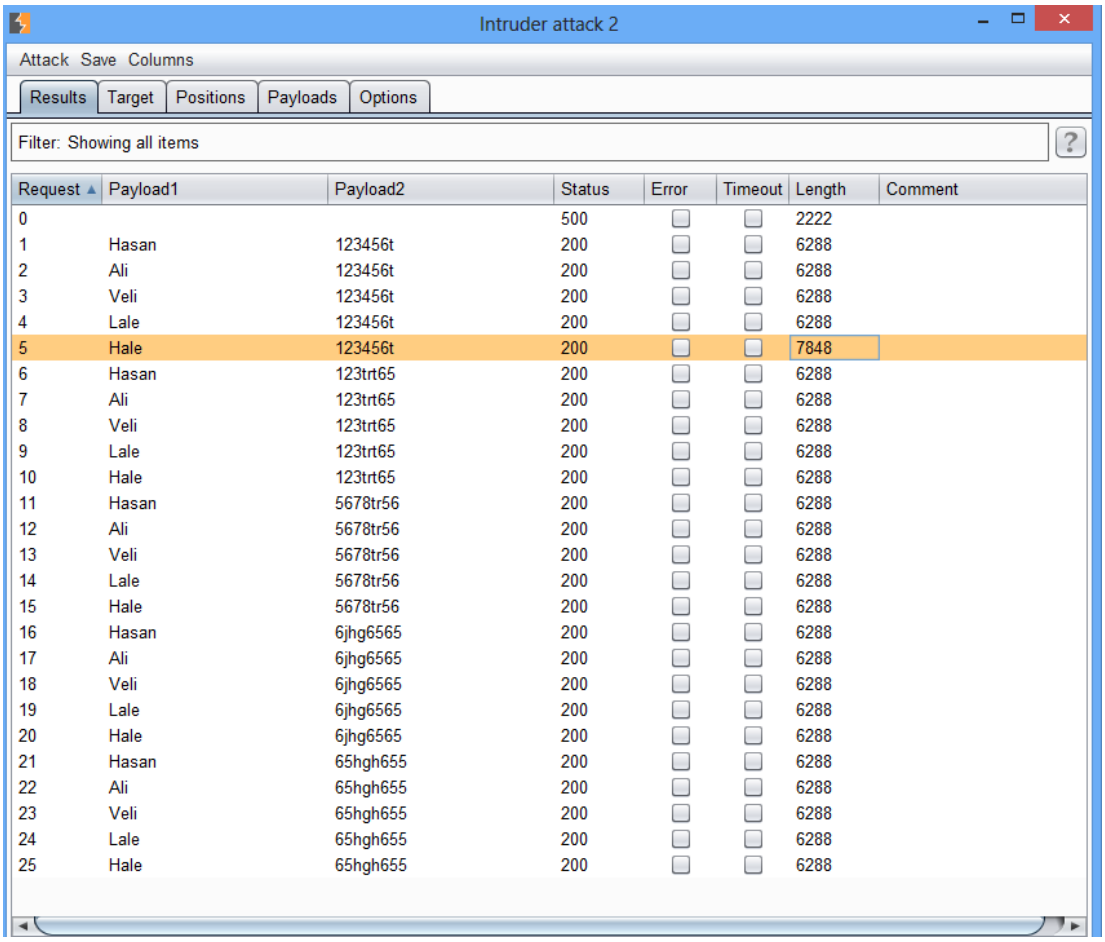
Kullanıcı Adı	Şifre
Hale	123456t
Ali	123trt65
Lale	5678tr56
Veli	6jhg6565
Hasan	65hgh655

Burp Suit aracı ve kullanacağımız tarayıc (Chrome) gerekli ayarlamaları yaptıktan sonra login sayfasına giriş bilgilerini girip Burp Suit aracının bu isteği yakalamasını sağlayalım.



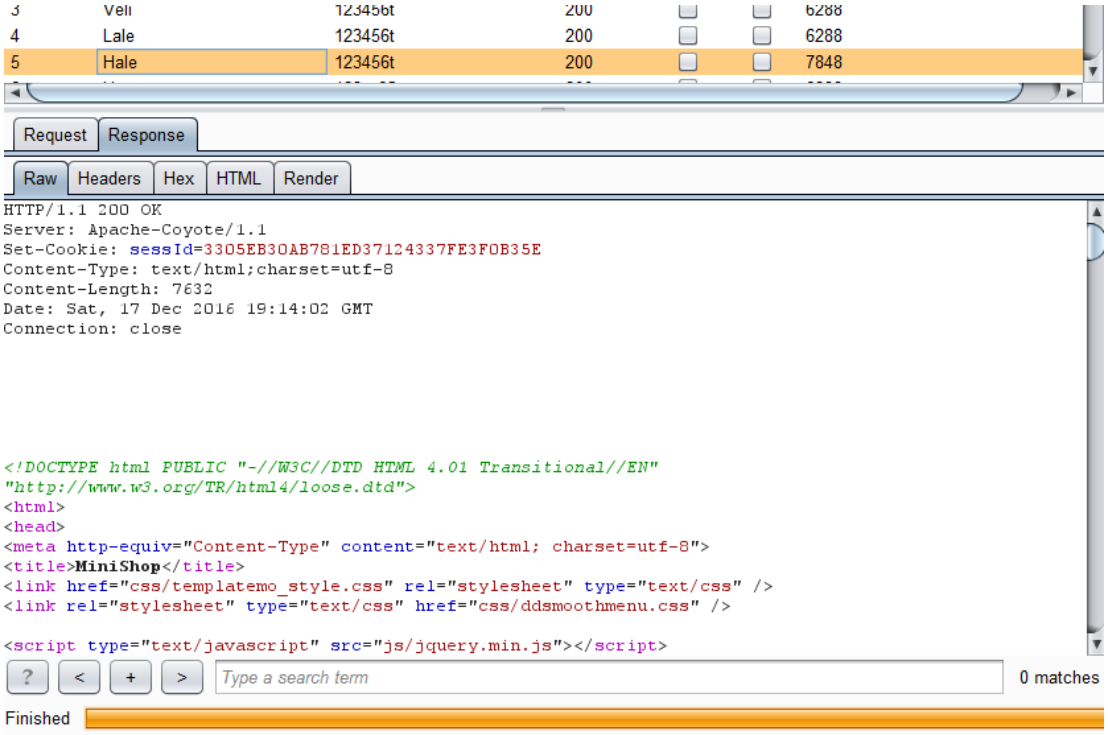
Şekil 4.1 : Örnek login sayfası

Burp Suit aracı liste içerisinde olan kullanıcı adı ve parola bilgilerini kullanarak toplamda 5X5'lik istek yaparak kullanıcının bilgilerini bulmaya çalışacaktır.



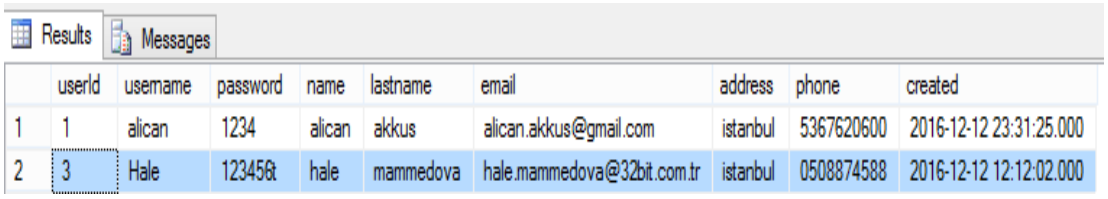
Request	Payload1	Payload2	Status	Error	Timeout	Length	Comment
0			500			2222	
1	Hasan	123456t	200			6288	
2	Ali	123456t	200			6288	
3	Veli	123456t	200			6288	
4	Lale	123456t	200			6288	
5	Hale	123456t	200			7848	
6	Hasan	123trt65	200			6288	
7	Ali	123trt65	200			6288	
8	Veli	123trt65	200			6288	
9	Lale	123trt65	200			6288	
10	Hale	123trt65	200			6288	
11	Hasan	5678tr56	200			6288	
12	Ali	5678tr56	200			6288	
13	Veli	5678tr56	200			6288	
14	Lale	5678tr56	200			6288	
15	Hale	5678tr56	200			6288	
16	Hasan	6jhg6565	200			6288	
17	Ali	6jhg6565	200			6288	
18	Veli	6jhg6565	200			6288	
19	Lale	6jhg6565	200			6288	
20	Hale	6jhg6565	200			6288	
21	Hasan	65hgh655	200			6288	
22	Ali	65hgh655	200			6288	
23	Veli	65hgh655	200			6288	
24	Lale	65hgh655	200			6288	
25	Hale	65hgh655	200			6288	

Şekil 4.2 : Burp Suit aracı ile sözlük saldırısı testi



Şekil 4.3 : Burp Suit aracı ile sözlül saldırısı deneme sonucu

Resimden görüleceği gibi tüm istekler aynı **http 200** status kodu dönmüştür. Tüm isteklerden farklı olarak "**Hale 123456t**" için length alanı farklı gelmiştir ve diğer isteklerden farklı olarak response içerisinde hata sayfası değil, index.jsp sayfası gelmiştir. Buradan anlaşılacağı üzere Burp Suit'in kullanıcının bilgilerini "**Hale 123456t**" olarak bulduğunu söyleye biliriz. Veritabanında kullanıcı listesini çekersek, gerçekten de bulduğumuz bilgiler ile veritabanındaki bilgilerin aynı olduğu görebiliriz:



	userId	username	password	name	lastname	email	address	phone	created
1	1	alican	1234	alican	akkus	alican.akkus@gmail.com	istanbul	5367620600	2016-12-12 23:31:25.000
2	3	Hale	123456t	hale	mammedova	hale.mammedova@32bit.com.tr	istanbul	0508874588	2016-12-12 12:12:02.000

Şekil 4.4 : Veritabanı kullanıcı listesi

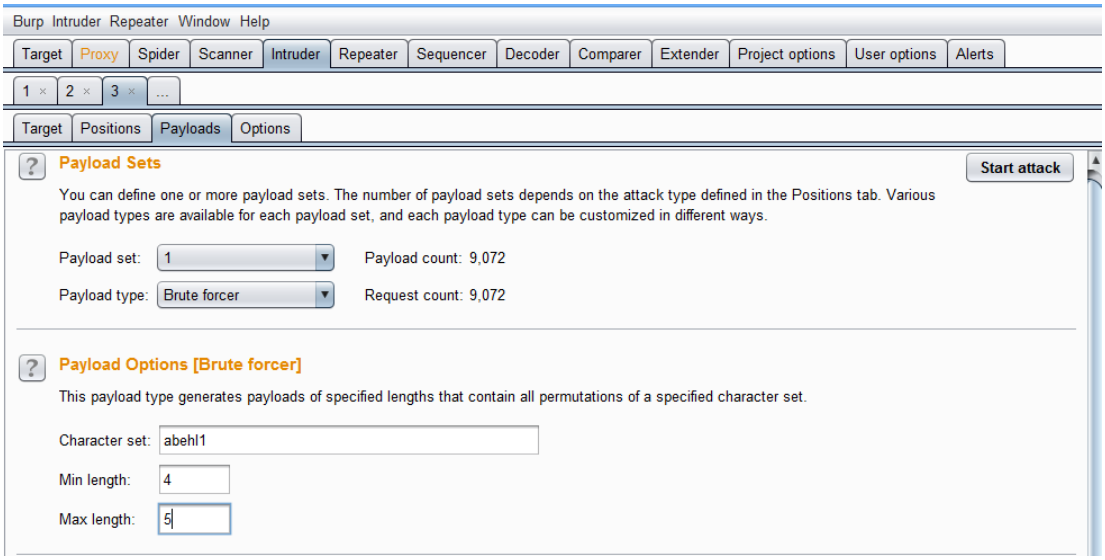
4.1.1.2 Kaba Kuvvet Saldırılar

Kaba Kuvvet saldırılarını 2 yöntem üzerinden deneyeceğiz.

- Yatay Yöntem
- Dikey Yöntem

Yatay Yöntem

Bu yöntemde parola sabit tutulurken değişken olan kullanıcı adı bilgisidir. Dolayısıyla yapılacak deneme sayısını kullanıcı adı kümesi boyutu belirlemektedir. Burp Suitin öntanımlı olarak sunmuş olduğu karakter kümesi "abcdefghijklmnoprstuvwxyz0123456789" şeklindedir ve min/max karakter uzunluğu 4/5'tir. Bu değerlere göre 62.145.792 adet deneme yapılacaktır. Bu çok yüksek bir değer olduğu için bazı harf ve karakterlerden vazgeçilebilir.



Şekil 4.5 : Burp Suit aracı ile yatay yöntem saldırısı testi

Ilgili deneme yanılma testi için yapılacak istek sayısı 9.072'ye düşecektir.

Dikey Yöntem

Bu yöntemde kullanıcı adı sabit tutulurken parola alanı değişkendir. Kullanım şekli yatay yöntem ile benzerdir.

4.1.2 SQL enjeksiyonu zafiyetinin tespiti ve istismarı

Bu bölümde uygulamaların kullanıcıdan girdi alarak oluşturmuş oldukları SQL sorguları sonucu ortaya çıkan SQL enjeksiyonu zafiyetini inceliyor olacağız.

4.1.2.1 Basit SQL enjeksiyonu

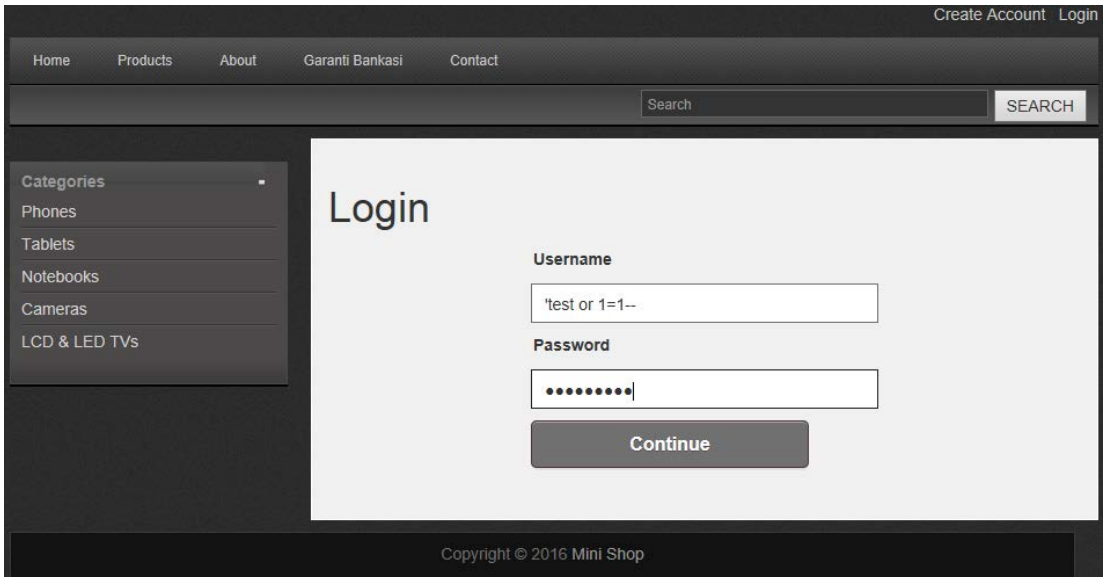
Ilk olarak geliştirilmiş olan uygulamanın giriş sayfasını bu yöntemle deneyerek ilgili zafiyetin varlığını tespit edeceğiz.

Uygulama'da kullanıcı girişini yapan metod aşağıdaki gibidir:

```java
public User login(String username, String password) {

    PreparedStatement preparedStatement = null;
    ResultSet rs = null;
    Connection connection = null;
    User user = new User();

    try {
        ConnectionManager connectionManager = ServiceFacade.getInstance().getConnectionManager();
        connection = connectionManager.getConnection();

        StringBuilder query = new StringBuilder("select * from users where username=");
        query.append("'").append(username).append("'");
        query.append(" and password=").append("'").append(password).append("'");

        logger.info("EXECUTE QUERY : " + query);
        preparedStatement = connection.prepareStatement(query.toString());
        rs = preparedStatement.executeQuery();

        if (rs.next()) {
            user = new User(rs.getInt("userId"), UserType.USER, rs.getString("username"), rs.getString("password"), rs.getString("name"),
                    rs.getString("lastname"), rs.getString("email"), rs.getString("phone"), rs.getString("address"), rs.getDate("created"), true);
        }

    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        JDBCUtility.close(rs, preparedStatement, connection);
    }

    return user;
}
```

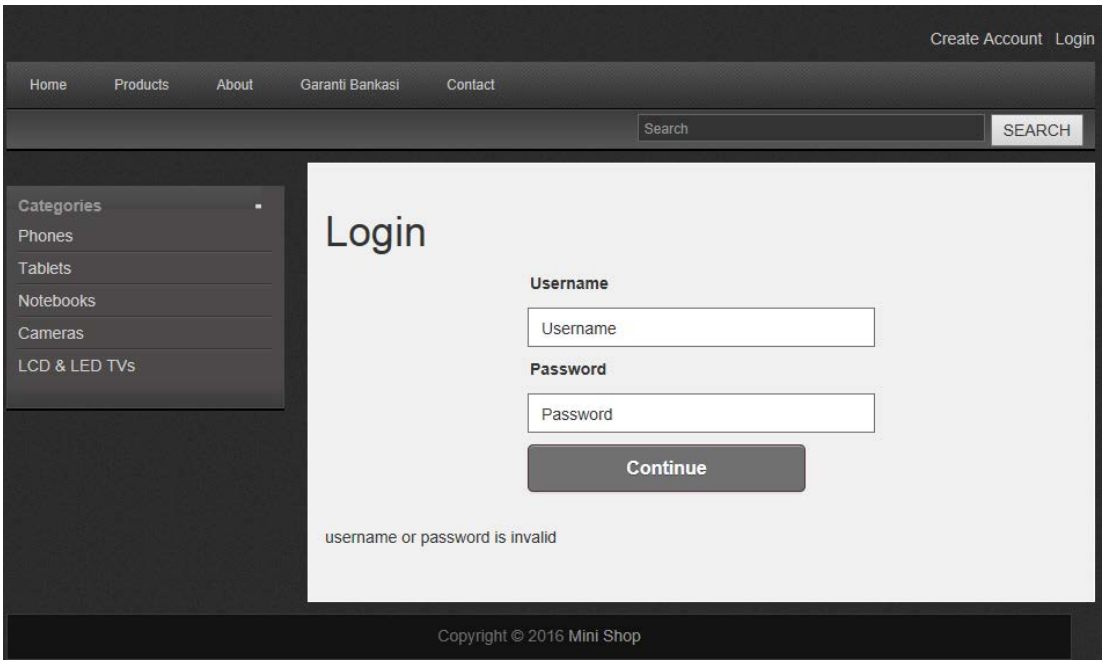
Şekil 4.6 : SQL enjeksiyonu için hazırlanmış kod örneği

Zafiyeti tespit etmek için ilk önce kullanıcı adı ve şifre olarak sql ifadelerini içeren değerler girelim. Girdiğimiz bu değerlere karşın uygulamanın verdiği tepkiyi inceleyelim.



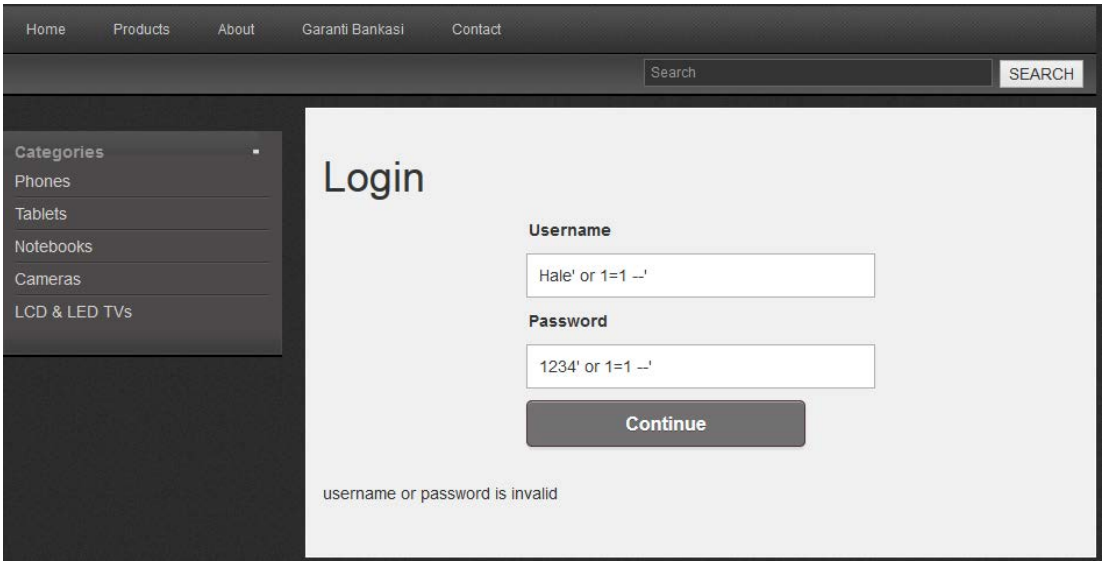
Şekil 4.7 : Örnek login sayfası'nda SQL enjeksiyonu denemesi

Yukarıda kullanıcı adı olarak test' or 1=1 ifadesini girerek kullanıcı girişi işlemini yapalım.



Şekil 4.8 : Örnek login sayfası'nda SQL enjeksiyonu tespiti

SQL ifadesi içeren girdiye karşılık olarak uygulama kullanıcı adı veya şifre hatası olduğunu belirtmiştir. Burada zararlı ya da SQL ifadesi içeren girdilerin kullanılmamasına dair herhangi bir uyarı bulunmamaktadır. Bu sayede girilen değerlerin herhangi bir kontrolden geçirilmeden sorgu içerisinde çalıştırıldıkları görülebilmektedir. Zafiyeti istismar edecek şekilde girdileri düzenleyelim.



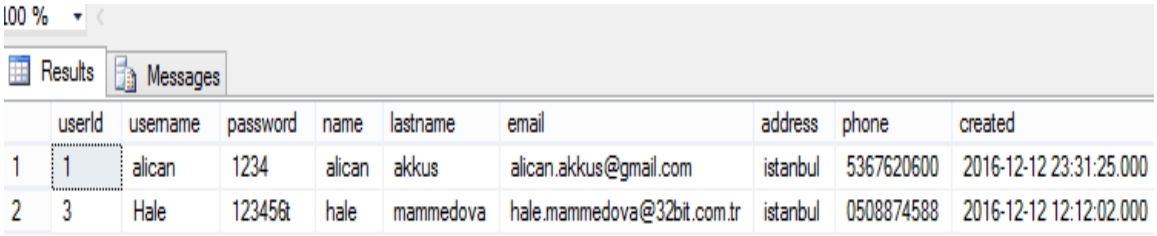
Şekil 4.9 : Örnek login sayfasında SQL enjeksiyonu istismarı

Note: Yukarıda SQL ifadesi görünecek şekilde *password* input alanı düzenlenmiştir. Çalıştırılan sorgu ise aşağıdaki şekilde dinamik bir biçimde oluşturulmuştur:

SELECT * FROM users WHERE username='Hale' OR 1=1--‘ AND password='1234' OR 1=1 --'

Kullanıcı adı veya şifre yanlış olmasına rağmen sisteme giriş yapılabilmiştir. Fakat giriş yapılan kullanıcı başka bir kullanıcıdır. Yapılan SQL enjeksiyonu sayesinde veritabanında bulunan ilk kullanıcı bilgileri ile sisteme giriş yapılmıştır.

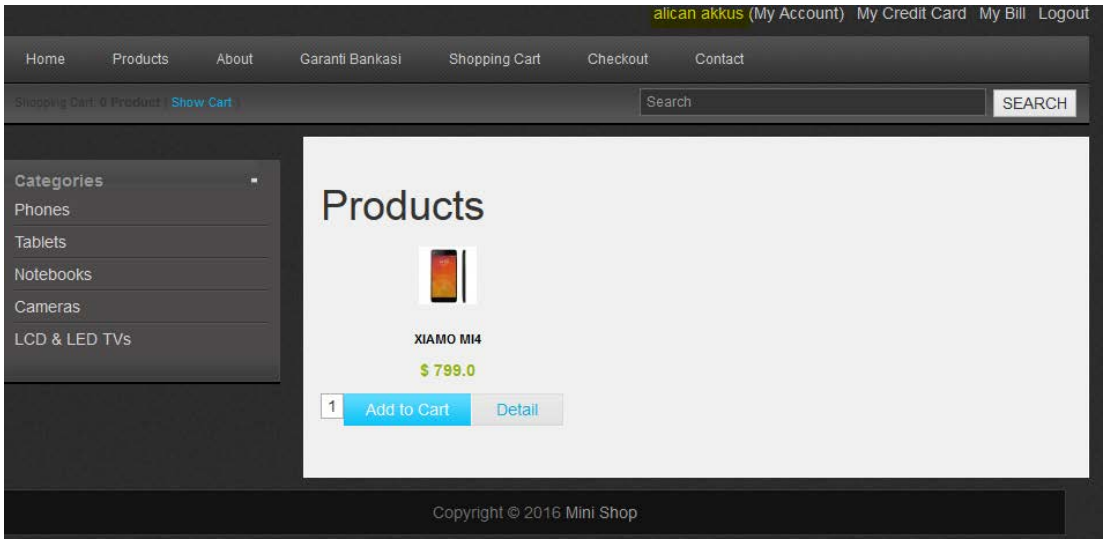
Yukarıdaki sorgunun sonucu aşağıdaki şekildedir:



	userId	username	password	name	lastname	email	address	phone	created
1	1	alican	1234	alican	akkus	alican.akkus@gmail.com	istanbul	5367620600	2016-12-12 23:31:25.000
2	3	Hale	123456t	hale	mammedova	hale.mammedova@32bit.com.tr	istanbul	0508874588	2016-12-12 12:12:02.000

Şekil 4.10 : Sonucun veritabanı tablo görüntüsü

Yapılan SQL enjeksiyon saldırısından dolayı sisteme veritabanındaki ilk kullanıcı ile giriş yapılmıştır:



Şekil 4.11 : SQL enjeksiyonu istismarı sonucu kullanıcı profili

4.1.2.2 Hata tabanlı SQL enjeksiyonu

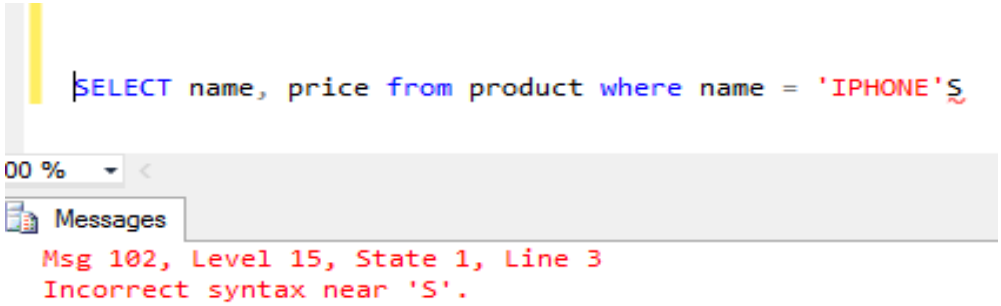
İlgili enjeksiyon türünün tespiti için veritabanı hata mesajının ekrana basılıyor olmasının yeterli olduğu incelenmişti. Uygulama üzerinden hata tabanlı enjeksiyonun tespitine örnek verelim. Klasik test parametresi olan tek tırnak karakterini uygulamanın zafiyet barındırdığını düşündüğümüz girdi noktasına gönderelim.

Farzedelim ki, kullanıcı IPHONE'S telefonlarını aratmak istiyor. Bu zaman uygulama aşağıdakı sorguyu gerçekleştirecektir;

SELECT name, price FROM product WHERE name = 'IPHONE'S

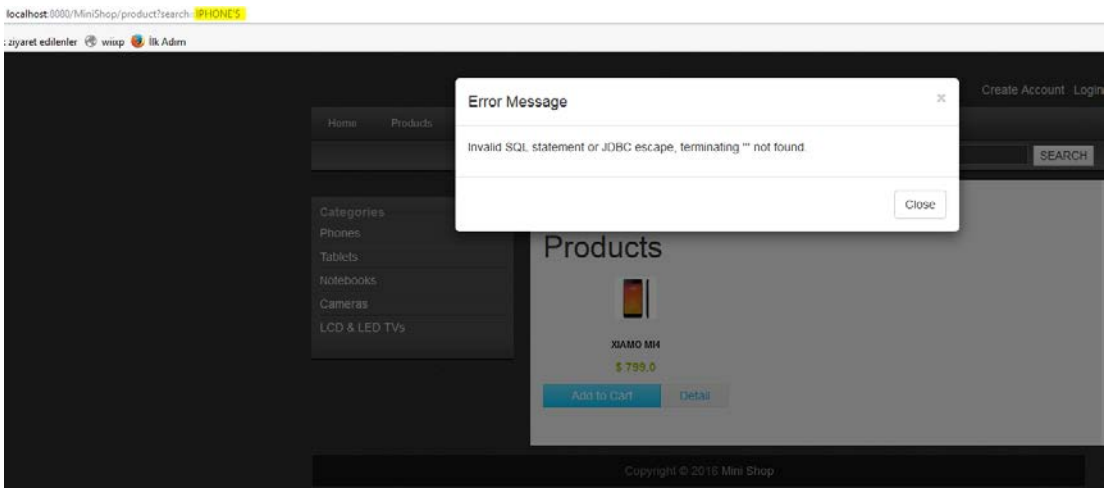
Bu durumda, sorgu yorumlayıcısı (query interpreter), String datasına ulaştığında tek tırnak (') içeren verini ayrıştıracak ve IPHONE değerini elde edecektir. S ifadesi ile karşılaştığında, geçerli SQL syntax içermediğini görüp hata döndürecektir:

İlgili hatanın veritabanı ekran görüntüsü aşağıdaki gibidir:



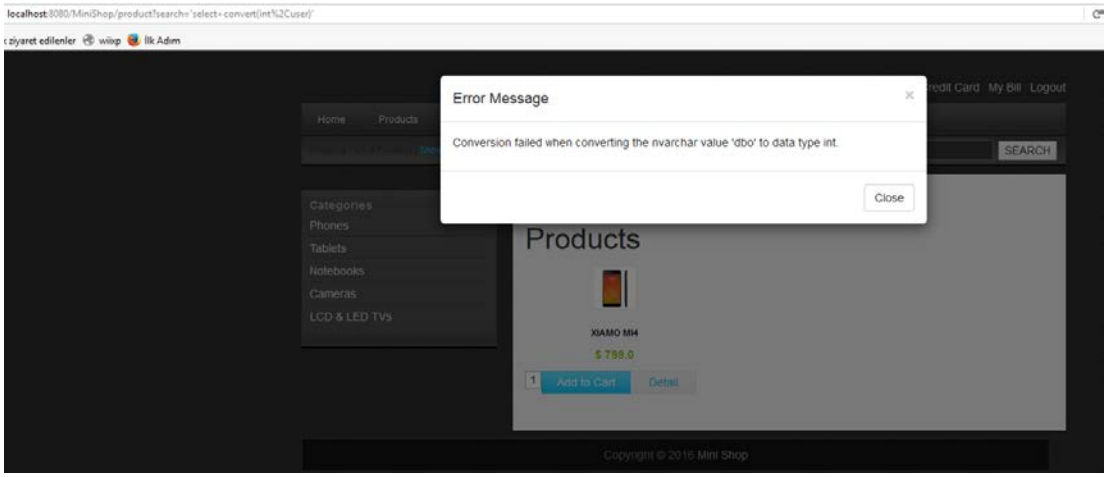
Şekil 4.12 : Hata Tabanlı SQL enjeksiyonunda tek tırnak hatasının görüntüsü

Uygulamanın yapılan isteğe verdiği tepki ise aşağıdaki gibidir:



Şekil 4.13 : Hata Tabanlı SQL enjeksiyonu tespiti

Hata mesajları yardımı ile veritabanı hakkında bilgi de almak mümkündür. Arama kısmına 'select convert (int, user)' yazdığımızda veritabanı kullanıcı bilgisine ulaşmış oluruz. Ekran görüntüsü aşağıdaki gibidir:



Şekil 4.14 : Hata Tabanlı SQL enjeksiyonu istismarı

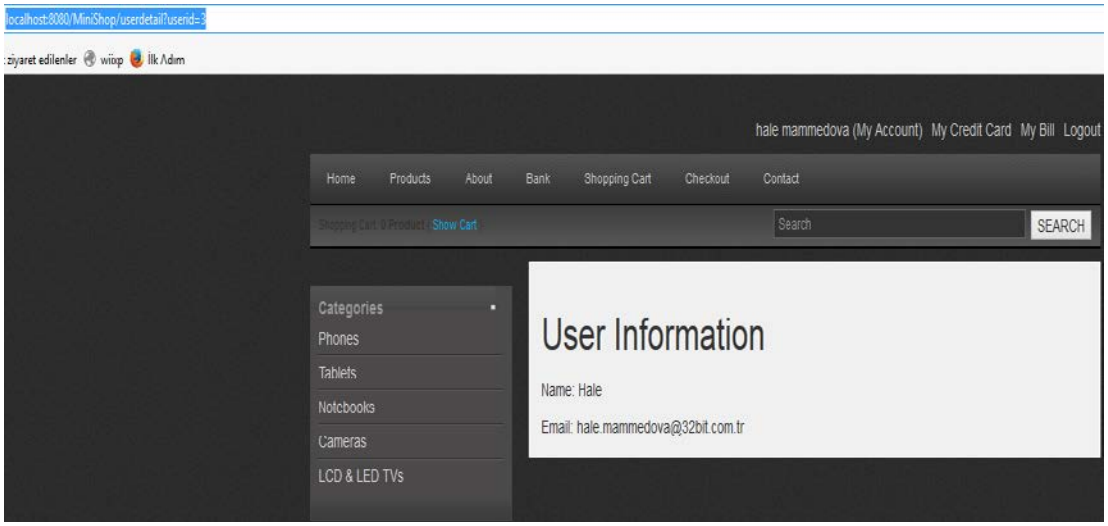
Uygulama bu şekilde davrandığında SQL Enjeksiyonuna karşı savunmasız olduğunu gösterecektir.

4.1.2.3 Kör (Blind) SQL enjeksiyonu

Kör SQL enjeksiyonun'da, uygulamanın hata mesajlarına bakmak yerine zafiyet barındıran sayfanın manipülasyonlara verdiği cevapların **true/false (doğru/yanlış)** olması ile ilgilendiğini incelemiştik. Uygulamada kullanıcı profilini aşağıdaki URL ile görüntülüyor olalım:

http://localhost:8080/MiniShop/userdetail?userid=3

Yukarıda belirtmiş olduğumuz URL, veritabanında userId=3 olan kullanıcının bilgisini getiriyor olacaktır:

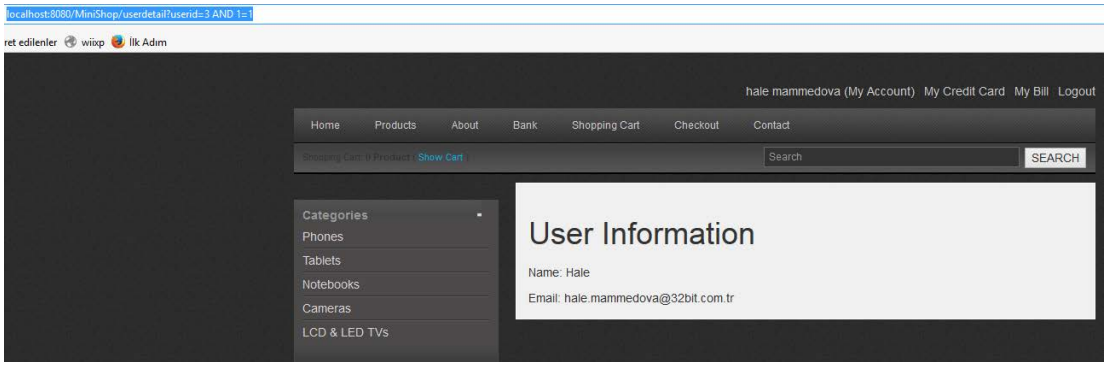


Şekil 4.15 : Örnek kullanıcı bilgileri sayfası

userId parametresinde kör SQL enjeksiyon zafiyetini tespit edecek olursak, aşağıdakine benzer şekilde **true/false (doğru/yanlış)** cevaplar bulmamız gerekecektir.

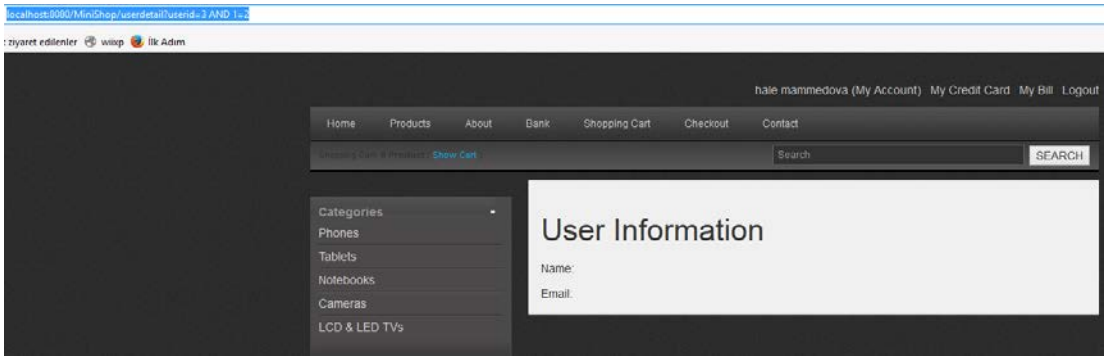
http://localhost:8080/MiniShop/userdetail?userid=3 AND 1=1

İlgili istek için yine aynı kullanıcının profilini görüntülüyorsak (True)



Şekil 4.16 : Kör Tabanli SQL enkejsiyonu tespiti (1)

ve ardından http://localhost:8080/MiniShop/userdetail?userid=3 AND 1=2 isteği için aşağıdaki cevabı görüntülüyorsak (False);

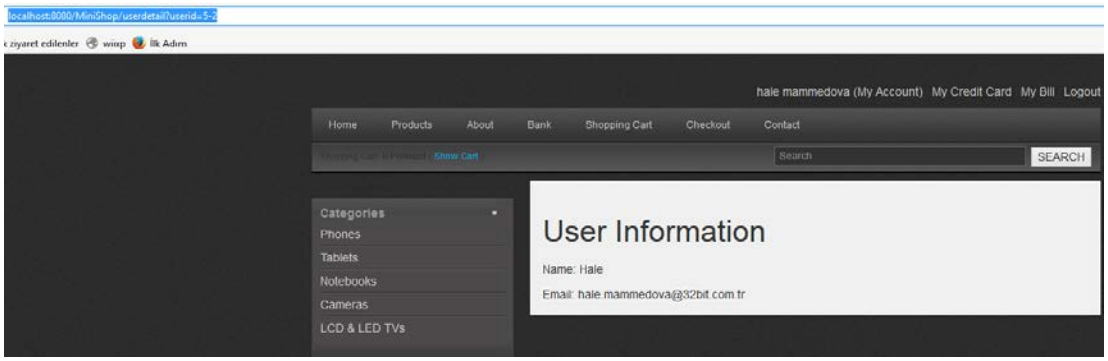


Şekil 4.17 : Kör Tabanlı SQL enjeksiyonu tespiti (2)

Burada bir kör SQL enjeksiyonu zafiyetinin varlığını söyleye biliriz. Kör SQL enjeksiyonu zafiyetini tespit etmek için farklı yöntemler de kullanılabilir.

Örneğin,

http://localhost:8080/MiniShop/userdetail?userid=5-2 isteği ile yine aynı kullanıcı profile görüntülenebiliyor ise de burada zafiyetin varlığı söz konusu olacaktır.



Şekil 4.18 : Kör Tabanlı SQL enjeksiyonu tespiti (3)

Çünkü, 5-2=3 değeri dinamik oluşturulan SQL cümleciği içerisinde hesaplanacak ve userId=3 olan kullanıcı bilgisini ekrana getirecektir. Bu da ilgili userId parametresinin SQL cümleciği içerisinde dinamik bir parameter olarak kullanıldığının işareti olacaktır.

Zafiyet barındıran ilgili kodu inceleyecek olursak;

```
String sql = "select username,email from users where userId= " + userid;
statement = connection.createStatement();
rs = statement.executeQuery(sql);

if (rs.next()) {
    user.setUsername(rs.getString("username"));
    user.setEmail(rs.getString("email"));
    return user;

}
```

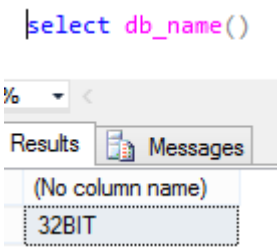
Şekil 4.19 : Zafiyet barındıran kod örneği

userId değişkeni SQL sorgusu içerisine dinamik olarak eklenmiştir ve hiçbir kontrolden geçmediği için ilgili kod parçacığı SQL enjeksiyonu zafiyeti barındırmaktadır.

Kör SQL enjeksiyonu zafiyeti istismarında Çizelge 4'de belirtmiş olduğumuz sorguları kullanarak veritabanından çekeceğimiz herhangi bir verinin karakterlerinin ASCII değerlerine erişme gibi bir imkanımız olmaktadır. İlgili sorgular ile istismar gerçekleştirecek olursak;

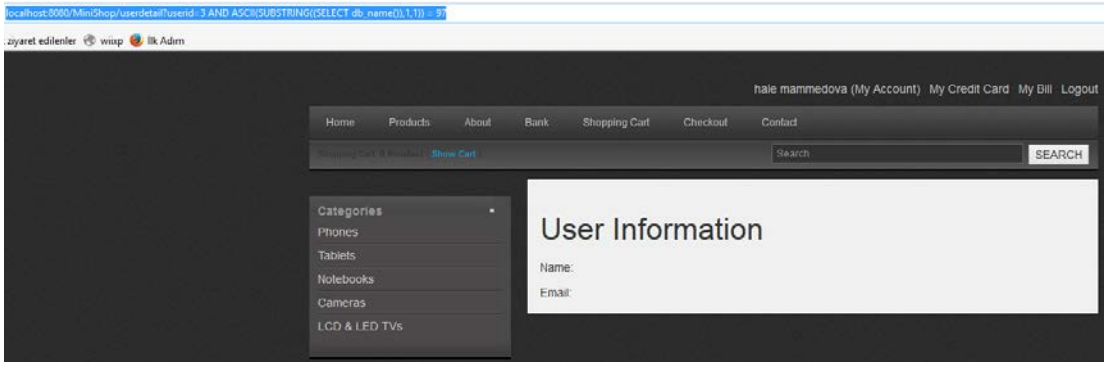
http://localhost:8080/MiniShop/userdetail?userid=3 AND ASCII (SUBSTRING ((SELECT db_name()),1,1))=97

97 değeri **a** karakterinin ASCII karşılığıdır. SQL Server'da **SELECT db_name ()** komutu çalıştırdığımızda kullandığımız database adını elde ederiz.



Şekil 4.20 : Veritabanı adının görüntülenmesi

ASCII (SUBSTRING ((SELECT db_name ()),1,1)) sonucu bize kullandığımız veritabanının ilk karakterini veriyor olacaktır. Dolayısıyla 51==97 eşitliği sağlanmadığı için AND'in sağ tarafı false dönecektir ve userId=3 değerindeki kullanıcı bulunsa bile ekrana boş sayfa yani False olarak belirttiğimiz cevap dönecektir.



Şekil 4.21 : Kör SQL enjeksiyonu istismarı

Bunun gibi **b**'nin, **c**'nin sırasıyla ASCII karşılıkları denenip Doğru olarak belirttiğimiz sayfayı görüntülediğimizde veritabanı isminin karakterlerini sırasıyla buluyor olacağız. Kör SQL enjeksiyonu zafiyetinin istismarı alınacak her bir karakterin taker taker kontrolü ile mümkün olmaktadır. İkinci karakter için sorguyu aşağıdaki gibi yaza biliriz;

http://localhost:8080/MiniShop/userdetail?userid=3 AND ASCII (SUBSTRING ((SELECT db_name()),2,1))=97

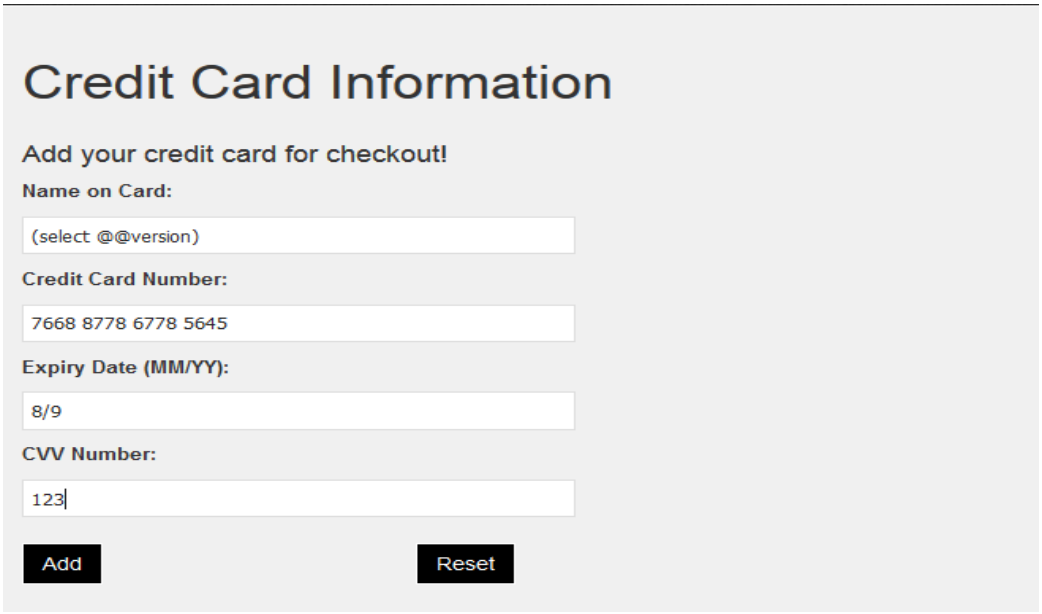
Kör SQL enjeksiyonunda her karakter için deneme yapılacağı muazzam sayıda istek yapılabileceğine neden olacaktır. Bu yüzden de Kör SQL enjeksiyonu başlığında belirttiğimiz üzere İkili Arama Algoritması kullanılmalıdır.

4.1.2.4 Farklı SQL ifadelerinde SQL enjeksiyonu zafiyeti

INSERT Cümleciği

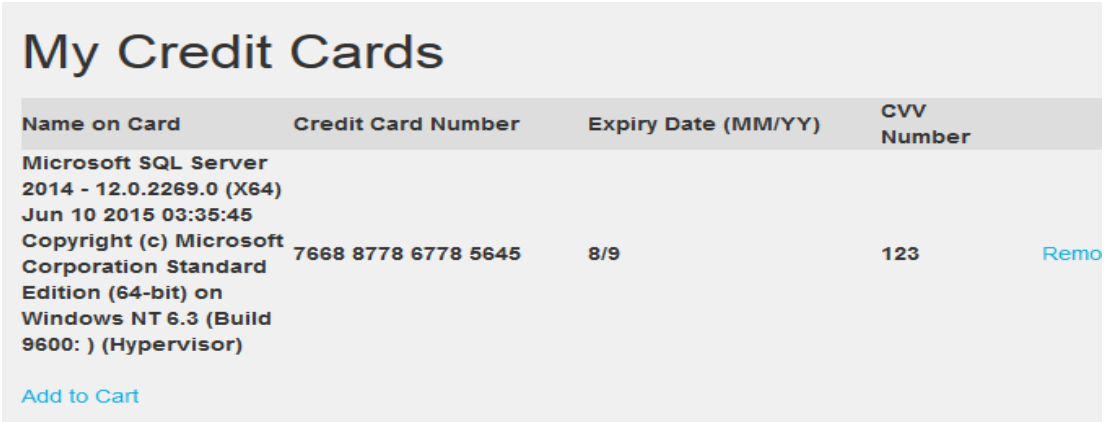
Uygulama'da satış işleminden önce kredi kartı bilgilerini giriyor olalım. Sonra girdiğimiz bilgileri görüntüleyebiliyor olalım.

Kredi kartı ekleme formunu aşağıdaki gibi doldurduğumuzda;



Şekil 4.22 : Örnek kredi kartı bilgileri giriş formu

İlgili name parametresinde SQL enjeksiyonu zafiyeti mevcut ise, eklenen değerden sonra gösterilecek kredi kartı bilgisi aşağıdaki gibi olacaktır:



Şekil 4.23 : Insert cümleciğinde SQL enjeksiyonu zafiyetinin tespiti

Bu tür zafiyetin oluşmasına sebep olacak kod parçacığı aşağıdaki gibidir:

```
String sql = "INSERT INTO [32BIT].[dbo].[creditcard] (name, creditcardnumber, expirydate, cvv, userId)" + " VALUES (" + name + ",'"
        + cardnumber + "','" + expiry + "','" + cvv + "', "+ userId +")";
statement = connection.createStatement();
logger.info("Execute Query : " + sql);

boolean success = statement.execute(sql);
if(success){
    rowCount = 1;
}else{
    rowCount = -1;
}
```

Şekil 4.24 : Zafiyet içeren insert cümleciği kod örneği

Kullanıcıdan control edilmeden alınan name parametresi değeri SQL cümleciği içerisine eklenir ise aşağıdakine benzer bir sorgu oluşacaktır:

INSERT INTO [32BIT].[dbo].[creditcard] (name, creditcardnumber, expirydate, cvv, userId) VALUES ((select @@version),'7668 8778 6778 5645','8/9','123', 3)

Dolayısıyla ilgili sorgu çalıştırıldığında veritabanı sürüm bilgisi alınıp, CREDITCARDNUMBER kolonuna yazılacaktır.

İlgili zafiyet yardımı ile veritabanından öğrenilmek istenilen veriler ekrana basılarak okunulabilir.

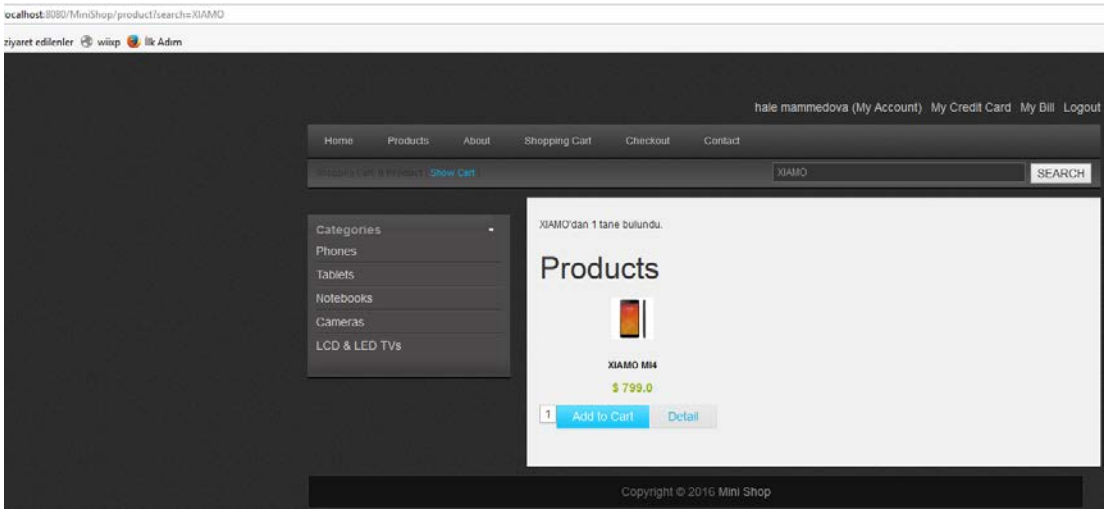
4.1.3 XSS zafiyetinin tespiti ve istismarı

Bu bölümde yapmış olduğumuz uygulama üzerinde Javascript ile zararlı kod parçacıklarının ne tür problemlere yol açabileceğini göstereceğiz.

4.1.3.1 Reflected XSS

Uygulama üzerinde arama kutucuğu kısmında normal bir ürün arama işlemi için kullanılacak GET isteği için URL ve ilgili istek için dönen mesajın ekran görüntüsü aşağıdaki gibi olacaktır:

http://localhost:8080/MiniShop/product?search=XIAMO

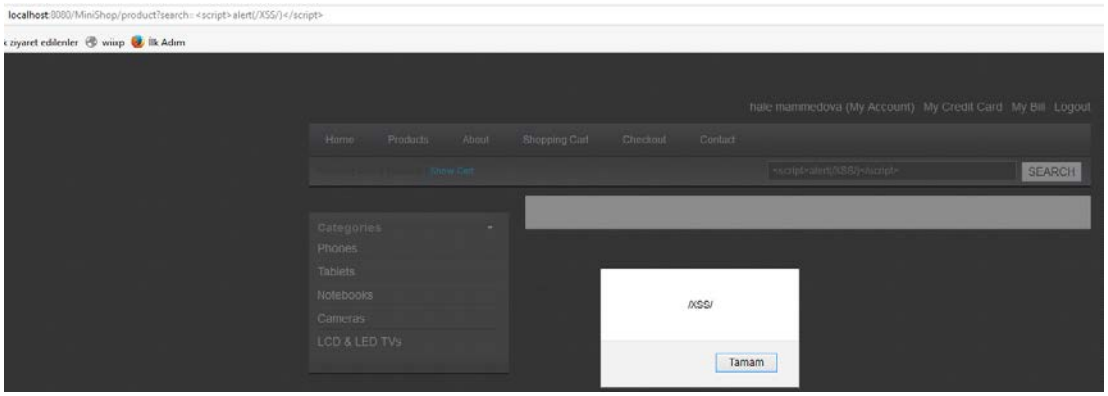


Şekil 4.25 : XSS zafiyeti tespiti (1)

Burada Reflected XSS zafiyetini tespit etmek için yazdığımız URL'i aşağıdaki gibi değiştirelim;

http://localhost:8080/MiniShop/product?search<script>alert(/XSS/)</script>

Yukarıdaki URL'in ekran görüntüsü aşağıdaki gibidir:

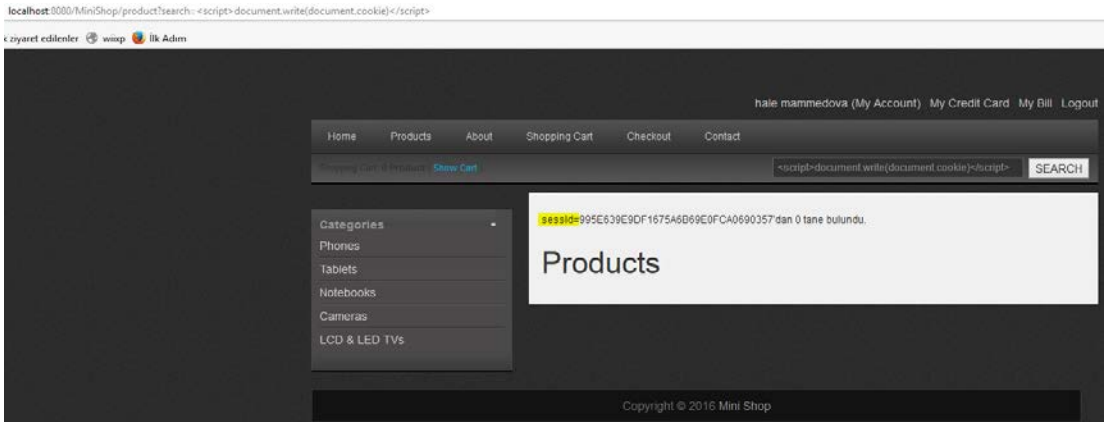


Şekil 4.26 : XSS zafiyeti tespiti (2)

Reflected XSS zafiyetinin varlığını tespit ettikten sonra zafiyeti istismar etmek için kullanıcının oturum bilgisini ele geçirebiliriz. Kullanıcının oturum bilgisini almak için DOM içerisinde document objesinin cookie değerini kullanıyoruz. URL'I aşağıdaki gibi düzenleyelim:

http://localhost:8080/MiniShop/product?search=<script>document.write(document.cookie)</script>

Yukarıdaki URL'in cevabı olarak ürün arama sonucu değerinde kullanıcının cookie bilgisi ekrana yazdırılmaktadır.



Şekil 4.27 : XSS zafiyetinin istismarı

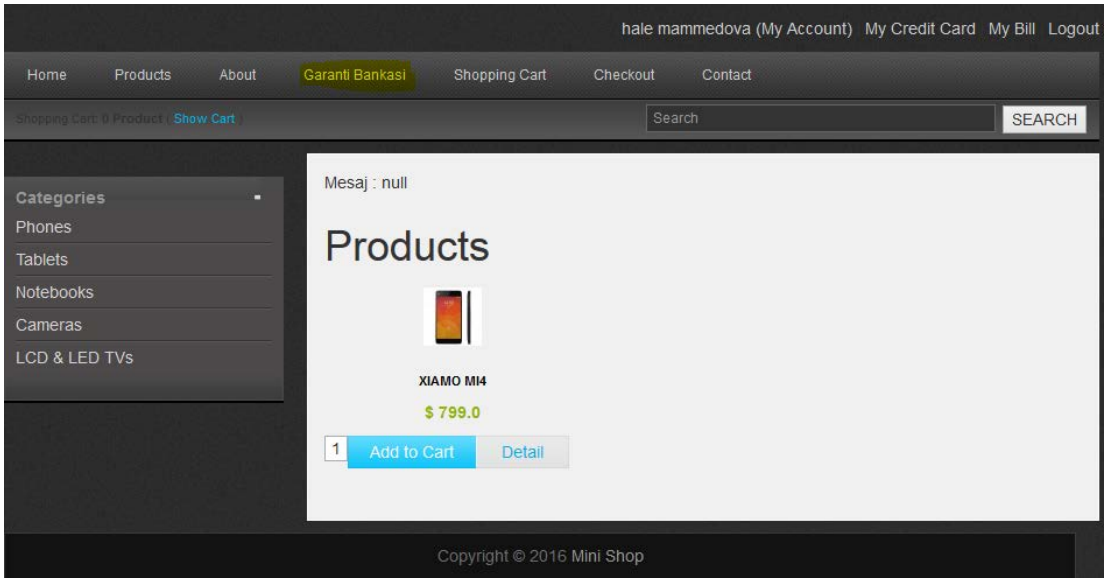
Reflected XSS zafiyetinin tespitinde gönderdiğimiz test parametresinin bize gelen cevap içerisinde aynı şekilde olup olmadığına bakarız. Test parametresinin değerini ayrıca tarayıcı yardımı ile HTML kaynak kodu görüntüleyerek de görebiliriz.

```
        <div id="templatemo_main">

<div id="sidebar" class="float_l">
    <div class="sidebar_box">
        <span class="bottom"></span>
        <h3>Categories</h3>
        <div class="content">
            <ul class="sidebar_list">
                <li class="first"><a href="product?category=1">Phones</a></li>
                <li><a href="product?category=2">Tablets</a></li>
                <li><a href="product?category=3">Notebooks</a></li>
                <li><a href="product?category=4">Cameras</a></li>
                <li class="last"><a href="product?category=5">LCD & LED TVs</a></li>
            </ul>
        </div>
    </div>
</div>

            <div id="content" class="float_r">

                <h6>

                    <script>document.write(document.cookie)</script>'dan 0 tane bulundu.

                </h6>

                <h1>Products</h1>


            </div>
            <div class="cleaner"></div>
        </div>

        <div id="templatemo_footer">
    Copyright © 2016<a href="#"> Mini Shop</a>
</div>


    </div>
</body>
</html>
```

Şekil 4. 28 : HTML kaynak kod görüntüsü

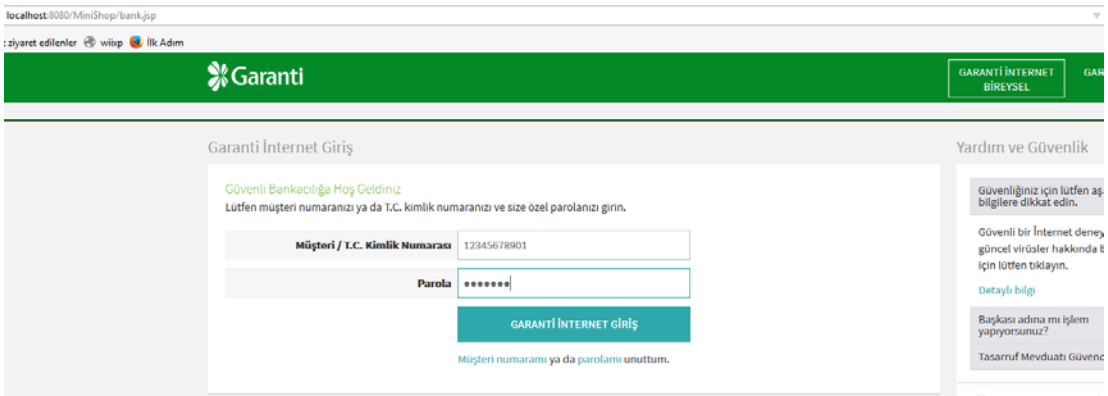
4.1.4 Phishing zafiyetinin tespiti ve istismarı

Phishing saldırısının bankalar, alış-veriş siteleri, sosyal ağlar vb. gibi sistemlerin bir kopyasının oluşturularak yapıldığını incelemiştik. Kurban, saldırganın hazırlanmış olduğu sahte siteye yönlendirilerek normal şekilde bilgilerini paylaşır, saldırgan ise kurbandan aldığı verileri istismar etmiş olacaktır.



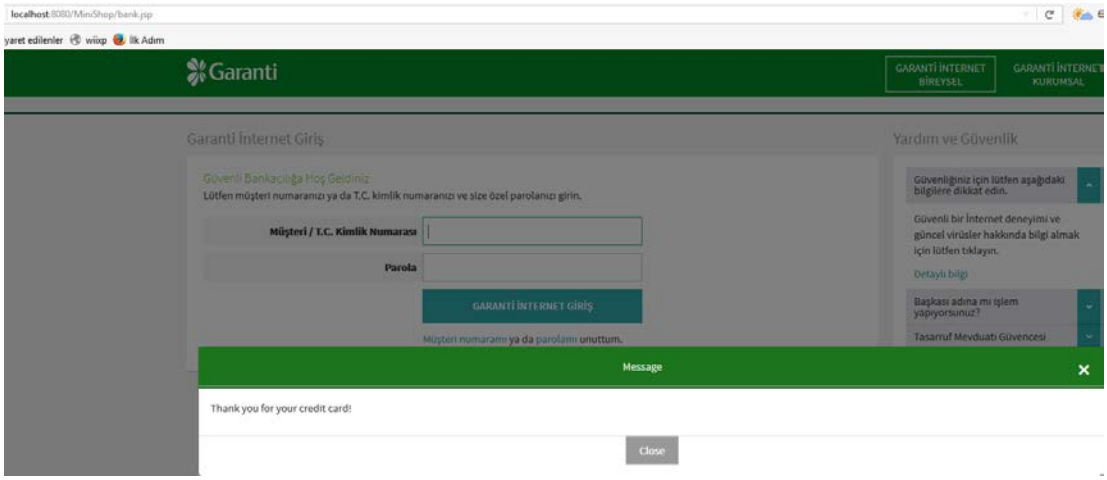
Şekil 4. 29 : Phising saldırısı tespiti

Yukarıdaki resimde Garanti Bankası linki sahte oluşturulmuş bankanın kopyasıdır. Bu linke tıkladığımızda gerçekten Garanti Banka'sına giriş yapıyormuş gibi sahte sayfa açılacaktır.



Şekil 4.30 : Phising saldırısının istismarı (1)

Kurban, olması gereken bir siteye girdğini zannederek bilgilerini girecektir. Saldırgan ise aldığı bilgileri istediği gibi kullanabilme imkanı elde etmiş olur. Saldırgan elde ettiği bilgiler ile gerçek siteye isteklerde bulunabilir ve/veya elindeki bilgileri saklayabilir.



Şekil 4.31 : Phising saldırısının istismarı (2)

5. SONUÇ

Web'in gelişmesi ve yaygınlaşması ile beraber birtakım sorunlar da meydana çıkmıştır. Günümüzde Web üzerinde bir günde yaklaşık milyarlarca işlem yapılmaktadır. Geçmişte web üzerinden yapamadığımız birçok işlev ve görevi artık yapabilir durumdayız. Büyüyen ve gelişen tüm sistemlerde güvenlik problemlenin bulunması gibi Web üzerinde de birtakım güvenlik zafiyetleri ön plana çıkmıştır. Bu zafiyetler zaman içerisinde giderilmeye çalışılsa bile günümüzde hala bu zafiyetlere rastlamak mümkündür. Bankaların, kamunun, birçok kurumun hizmetlerini artık web üzerinden sunmaya başlamasıyla birlikte web yazılım güvenliği alanındakı saldırıları da arttırmıştır. Doğrudan kullanıcıları ilgilendiren sorunların bulunması ve ayrıca kurum ve/veya kuruluşlara yaılan saldıların gün geçtikçe artması nedeniyle tez konusu "Web uygulamalarında güvenlik" seçilmiştir. Tez içerisinde web'in gelişiminden başlayarak, web sistemlerinde yaşanmış ve yaşanabilecek olan zafiyetler ele alınmıştır. Zafiyetlerin tespiti, istismar edilmesi gibi konular ele alınırken, neden oldukları sorunlar da irdelenmiştir. Belirtilen zafiyetlerin engellenmesi ve çözümlenmesi için farklı yaklaşımlar önerilmiştir. Bunun yanı sıra zafiyetlere olan çözüm yaklaşımları ifade edilmiş ve gerçeklenmiştir. Tez ile birlikte güvenli yazılım geliştirme modelleri araştırılmış olup, sistemler üzerinde potansiyel zafiyet olabilecek bölümlerin analizinin yapılabilmesi için gerekli olan yazılım geliştirme süreçleri konularında bilgi sahibi olunmuştur.

Web sistemlerinde bulunan güvenlik zafiyetleri için güvenlik stratejisinin bulunması gerektiği ifade edilmiş olup, geliştirme süreçlerinin tamamında bu stratejinin bulunması gerektiği belirlenmiştir. Yukarıda ifade edilen kazanımlar sayesinde ilerde geliştirilecek olan yazılımlar daha dikkatli geliştirilebilecek ve kullanıcılara güven veren bir anlayış benimsenerek, güvenlik politikası bulunan ve güvenlik politikalarına uyan sistemler geliştirilebilecektir.

KAYNAKÇA

[1] <https://www.bilgiguvenligi.gov.tr/yazilim-guvenligi/yazilim-guvenligi-programi>, 29.05.2016.
[2]<http://www.owasp.org.tr>, 29.05.2016.
[3] **Dafydd Stuttard and Marcus Pinto. (2011).** The Web Application Hacker's Handbook: *Finding and Exploiting Security Flaws, Second Edition*
[4] **Mario Heiderich. (2011).** *Web Application Obfuscation.*
[5] <https://www.maxcdn.com/one/visual-glossary/keep-alive>, 01.06.2016.
[6] **Bünyamin Demir. (2013).** *Yazılım Güvenliği.*
[7] <https://www.nist.gov/cyberframework>, 05.06.2016.
[8] **Dafydd Stuttard and Marcus Pinto. (2009).** Web Security Testing Cookbook: *Systematic Techniques to Find Problems Fast.*
[9] **Steven Splaine. (2002).** *Testing Web Security-Assessing the Security of Web Sites and Applications.*
[10] <http://www.owasp.org.tr/index.php/ESAPI>, 10.07.2016.
[11] **Gary McGraw. (2006).** Software Security: *Building Security In.*
[12] **Ari Takanen, Jared DeMott and Charlie Miller. (2008).** *Fuzzing for Software Security Testing and Quality Assurance.*
[13] <https://www.owasp.org/index.php/Session_Management_Cheat_Sheet>, 25.07.2016.
[14] <https://www.owasp.org/index.php/Authentication_Cheat_Sheet>, 30.07.2016.
[15] <https://www.owasp.org/index.php/Session_fixation>, 04.08.2016.
[16]<https://www.owasp.org/index.php/SSL_Best_Practices>, 14.08.2016.
[17]<https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet>, 22.08.2016.
[18] **Michael Cross. (2007).** *Developer's Guide to Web Application Security.*
[19] **Justin Clarke. (2009).** *SQL Injection Attacks and Defense.*
[20] **Michael Howard, David LeBlanc and JohnViega. (2010).** 24 Deadly Sins of Software Security: *Programming Flaws and How to Fix Them.*
[21] **J.D. Mejer and Carlos Farre. (2009).** *Performance Testing Guidance for Web Applications.*
[22] **Patrick Engebretson. (2011).** The Basics of Hacking and Penetration Testing: *Ethical Hacking and Penetration Testing Made Easy.*
[23] **Joel Scambray, Vincent Liu and Caleb Sima. (2011).** *Web Application Security Secrets and Solutions.*
[24] **Rich Cannings, Himanshu Dwivedi and Zane Lackey. (2008).** Hacking Exposed Web 2.0: *Web 2.0 Security Secrets and Solutions.*
[25] **Jon Erickson. (2008).** Hacking: *The Art of Exploitation.*
[26]<https://www.owasp.org/index.php?title=SQL_Injection_Prevention_Cheat_Sheet>, 04.09.2016.
[27] **Ian Molyneaux. (2009).** *The Art of Application Performance Testing.*
[28] **Abhay Bhargav and B.V. Kumar. (2011).** Secure Java: *For Web Application Development.*
[29]<https://en.wikipedia.org/wiki/Phishing>, 15.09.2016.
[30]<https://en.wikipedia.org/wiki/Cross-site_request_forgery>, 17.09.2016.

[31]<https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)>, 22.09.2016.
[32]<https://www.owasp.org/index.php/Top_10_2010-A5-Cross-Site_Request_Forgery_(CSRF)>, 25.09.2016.
[33] **Leonard Ricbardson and Sam Ruby. (2007).** *RESTful Web Services.*
[34] **David Chappell and Tyler Jewell. (2002).** *Java Web Services.*
[35]<http://edergi.sdu.edu.tr/index.php/utbd/article/viewFile/5931/4674>, 01.10.2016.
[36]<https://tr.wikipedia.org/wiki/SOAP>, 09.10.2016.
[37]<http://edergi.sdu.edu.tr/index.php/utbd/article/viewFile/5931/4674>, 14.10.2016.
[38] **B. Jesse. (2007).** Cross-Site Request Forgeries: *An introduction to common web application weakness.*
[39]<http://www.turksiberguvenlik.net/konu-csrf-cross-site-request-forgery-nedir.html>, 22.10.2016.
[41] **Z. William, F. Edward. (2008).** Cross-Site Request Forgeries: *Exploitation and Prevention.*
[42]< http://jeremiahgrossman.blogspot.com/2006/09/csrf-sleeping-giant.html>, 10.09.2016.

ÖZGEÇMİŞ

Ad-Soyad: Narmin Mammadova

Doğum Tarihi ve Yeri: 03.04.1991 Azerbaycan

E-Posta: hale.mammedova@gmail.com

ÖĞRENİM DURUMU:

- **Lisans:** 2012, Baku Devlet Üniversitesi, Kimya Mühendisliği
- **Yüksek Lisans:** İstanbul Aydın Üniversitesi, Bilgisayar Mühendisliği