DETECTİNG WEB ATTACKS **-Ahmet Arif Arslan-**

Web Saldırıları nedir?

Web uygulamaları, bir tarayıcı arayüzü aracılığıyla kullanıcılara hizmet sağlayan uygulamalardır. Günümüzde web uygulamaları internet kullanımının büyük bir bölümünü oluşturmaktadır. Google, Facebook ve YouTube gibi siteler (mobil uygulamalar hariç) aslında web uygulamalarıdır.

Web uygulamaları pek çok kuruluş için internet üzerinde bir arayüz olduğu için saldırganlar bu uygulamaları sömürebilir ve cihazlara sızabilir, kişisel verileri ele geçirebilir veya ciddi maddi hasara yol açan hizmet kesintilerine neden olabilir.

Acunetix tarafından yapılan bir araştırma, gerçekleştirilen tüm siber saldırıların %75'inin web uygulaması düzeyinde olduğunu belirledi.

Aşağıda, web uygulamalarına sızmak için kullanılan bazı saldırı yöntemlerini bulacaksınız. Bu yöntemleri “Web Saldırıları 101” kursumuzda ele alacağız; bu yöntemlerin neler olduğunu, saldırganların bunları nasıl ve neden kullandığını ve bu tür etkinlikleri nasıl tespit edebileceğimizi anlatacağız.

- SQL Enjeksiyon
- Siteler Arası Komut Dosyası Oluşturma
- Komut Enjeksiyonu
- kimlik
- RFI ve LFI
- Dosya Yükleme (Web Kabuğu)

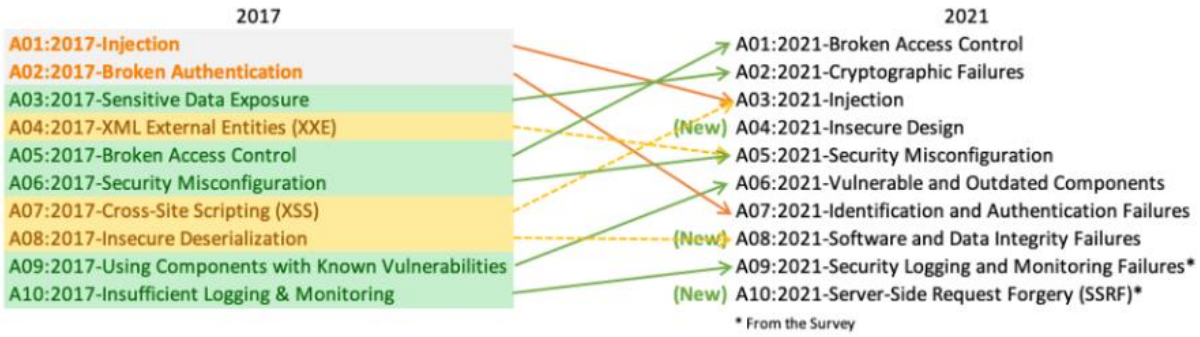
OWASP

Açık Web Uygulama Güvenliği Projesi (OWASP), yazılım güvenliğini artırmak için çalışan, kar amacı gütmeyen bir kuruluştur.[1]

OWASP'ın web uygulaması güvenliği hakkında bilgi edinmek için en iyi kaynaklardan biri olduğu şüphesizdir.

OWASP İlk On

OWASP, birkaç yılda bir en kritik güvenlik risklerine sahip 10 web uygulaması güvenlik açığının bir listesini yayınlar. Bu makalenin yazıldığı tarih itibariyle en son yayın 2021 yılında yapılmıştır.



2021'de yayınlanan OWASP listesi şu kritik güvenlik risklerini içerir:

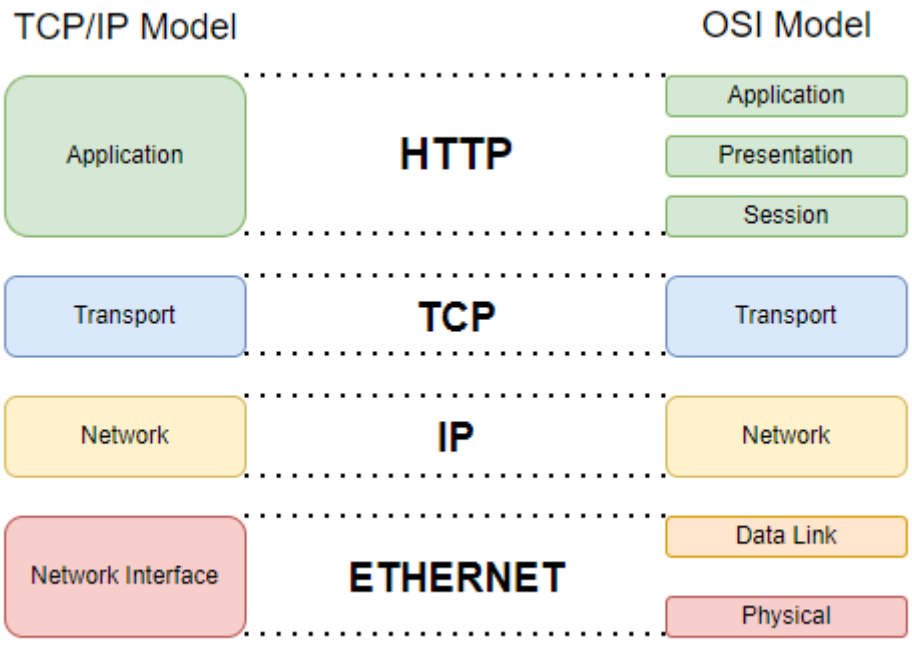
1. Bozuk Erişim Kontrolü
2. Şifreleme Hataları
3. Enjeksiyon
4. Güvensiz Tasarım
5. Güvenlik Yanlış Yapılandırması
6. Savunmasız ve Eski Bileşenler

7. Tanımlama ve Kimlik Doğrulama Hataları
8. Yazılım ve Veri Bütünlüğü Hataları
9. Güvenlik Günlüğü ve İzleme Hataları
10. Sunucu Tarafı İstek Sahteciliği (SSRF)

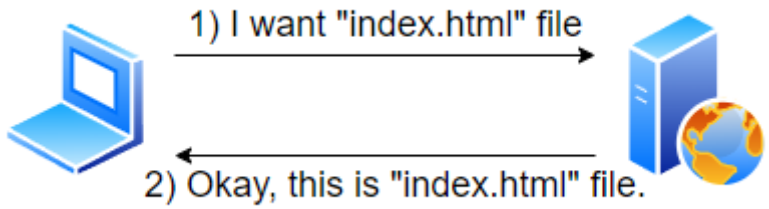
Web Uygulamaları Nasıl Çalışır?

Bir anormalliği tespit etmek için önce teknolojinin nasıl çalıştığını anlamamız gerekir. Uygulamalar, birbirleriyle doğru bir şekilde iletişim kurmak için belirli protokolleri kullanır. Web uygulamaları, Köprü Metni Aktarım Protokolü (HTTP) aracılığıyla iletişim kurar. HTTP protokolünün nasıl çalıştığına bakalım.

Başlangıç olarak, HTTP protokolünün OSI modelinin 7. katmanında olduğunu bilmek önemlidir . Bu, HTTP protokolünden önce Ethernet, IP, TCP ve SSL gibi protokollerin kullanıldığı anlamına gelir.



HTTP iletişimi, sunucu ve istemci arasında gerçekleşir. İlk olarak, istemci sunucudan belirli bir kaynak ister. Sunucu, HTTP isteğini alır ve belirli kontroller ve işlemlerden geçirdikten sonra istemciye bir (HTTP Yanıtı) geri gönderir. İstemcinin cihazı yanıtı alır ve istenen kaynağı uygun bir biçimde görüntüler.

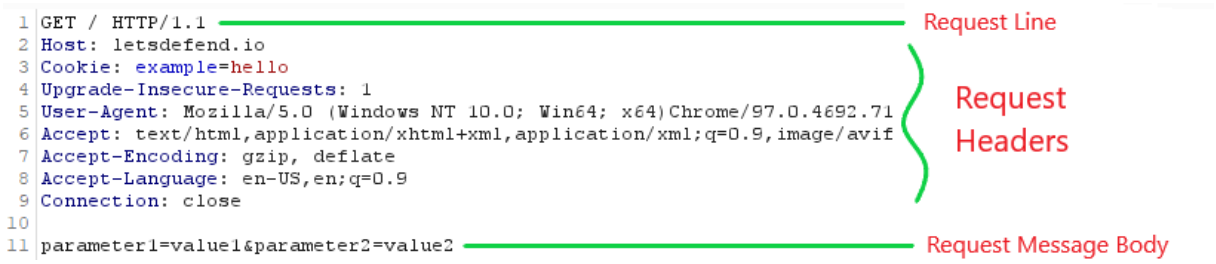


HTTP İsteklerini ve HTTP Yanıtlarını daha detaylı inceleyelim.

HTTP İstekleri

Bir web sunucusundan belirli bir kaynağı almak için bir HTTP İsteği kullanılır. Bu kaynak bir HTML dosyası, video veya json verisi vb. olabilir. Web sunucusunun görevi alınan yanıtı işlemek ve kullanıcıya sunmaktır.

Standart bir HTTP biçimi vardır ve web sunucularının isteği anlayabilmesi için tüm istekler bu biçime uymalıdır. Talep farklı bir formatta gönderilirse, web sunucusu bunu anlamaz ve kullanıcıya bir hata gönderir veya web sunucusu hizmet sağlayamayabilir (ki bu başka bir saldırı türüdür).



Bir HTTP İstek satırı, bir istek satırından, istek başlıklarından ve bir istek mesajı gövdesinden oluşur. Bir istek satırı, HTTP yönteminden ve web sunucusundan istenen kaynaktan oluşur. İstek başlığı, sunucunun işleyeceği belirli başlıkları içerir. İstek mesajı gövdesi, sunucuya gönderilmesi amaçlanan verileri içerir.

Yukarıdaki resimde bir HTTP İsteği örneği görüyorsunuz. Bu HTTP İsteğini satır satır inceleyelim.

1. GET yöntemi, “/” kaynağının sunucudan istendiğini belirtir. İsim olmadığı için “/” gibi bir sembol, web sunucusunun ana sayfasının istendiği anlamına gelir.
2. Günümüzde tek bir web sunucusunda bulunan birden fazla etki alanına ait web uygulamaları vardır, bu nedenle tarayıcılar, istenen kaynağın hangi etki alanına ait olduğunu açıklamak için “Host” başlığını kullanır.
3. Bir web uygulaması, müşterinin cihazında bilgi depolamak istediğinde, bunu bir “Çerez” başlığında saklar. Çerezler genellikle oturum bilgilerini saklamak için kullanılır. Bu nedenle, giriş gerektiren bir web uygulamasını ziyaret ettiğinizde kullanıcı adınızı ve şifrenizi yeniden girmeniz gerekmez.
4. "Upgrade-Insecure-Requests" başlığı, istemcinin şifreleme (SSL) ile iletişim kurmak istediğini belirtmek için kullanılır.
5. "User-Agent" başlığı altında istemcinin tarayıcısı ve işletim sistemi ile ilgili bilgiler bulunmaktadır. Web sunucuları, istemciye belirli HTTP Yanıtları göndermek için bu bilgileri kullanır. Bu başlığın altına bakarak bazı otomatik güvenlik açığı tarayıcılarını bulabilirsiniz.
6. İstenen veri türü “Kabul Et” başlığı altında bulunur.
7. İstemcinin anladığı kodlama türü “Kabul Et-Kodlama” başlığı altında bulunur. Genellikle bu başlık altında sıkıştırma algoritması adlarını bulabilirsiniz.

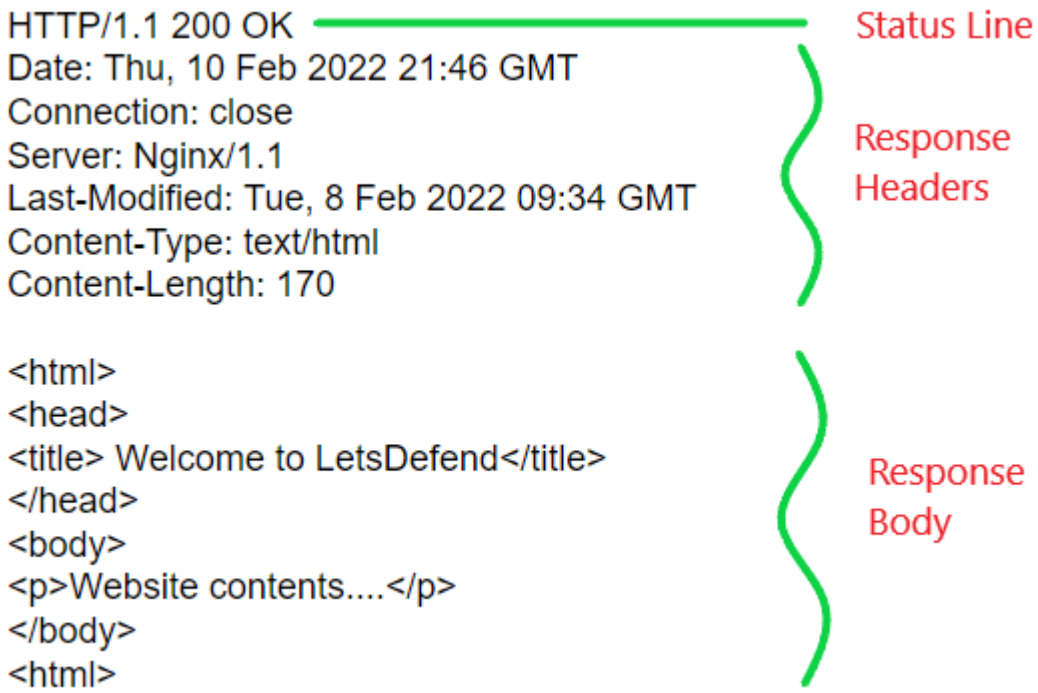
8. “Accept-Language” başlığının altında istemcilerin dil bilgilerini bulabilirsiniz. Web sunucusu, hazırlanan içeriği istemcinin dilinde görüntülemek için bu bilgileri kullanır.
9. “Bağlantı” başlığı, HTTP bağlantısının nasıl yapılacağını gösterir. Burada bulunan “kapat” gibi bir veri varsa, HTTP yanıtı alındıktan sonra TCP bağlantısının kapatılacağı anlamına gelir. “Canlı tut” ifadesini görürseniz bu, bağlantının devam edeceği anlamına gelir.
10. Bir bölüm oluşturmak için HTTP İstek Başlığı ile HTTP İstek İleti Gövdesi arasına boş bir satır konur.
11. Web uygulamasına gönderilmesi amaçlanan diğer veriler, İstek Mesajı Gövdesinde bulunur. HTTP POST yöntemi kullanılıyorsa, POST parametreleri burada bulunabilir.

HTTP Yanıtları

Web sunucusu bir HTTP İsteği aldığında, gerekli kontrolleri ve işlemleri gerçekleştirir ve ardından istenen kaynağı istemciye gönderir. Burada tek tip bir süreç yoktur çünkü çok sayıda teknoloji ve tasarım söz konusudur. Sunucu, talep edilen kaynağın ne olduğuna göre veritabanından veri çekebilir veya gelen verilere göre işlem yapabilir. Ancak HTTP Yanıt Mesajı, tüm işlemlerden sonra istemciye ulaşmalıdır.

Bir HTTP Yanıt İletisi, bir Durum Satırı, Yanıt Başlıkları ve bir Yanıt Gövdesi içerir. Durum Satırı, durum kodunu (örneğin, 200: OK) ve HTTP protokol bilgilerini içerir. Response Header içerisinde çok sayıda amaç için kullanılan başlıklar bulunmaktadır. Talep edilen kaynakla ilgili veriler, Müdahale Gövdesinde bulunur.

Bir web sayfası istendiyse, Yanıt Gövdesinde genellikle HTML kodları olacaktır. İstemci HTML kodunu aldığında, web tarayıcısı HTML kodunu işler ve web sayfasını görüntüler.



Yukarıdaki resimde bir HTTP Yanıt isteği görebilirsiniz. Bu görüntüye dayalı bir HTTP Yanıt isteğini inceleyelim.

Durum Satırı

Durum Satırında HTTP sürümü ve HTTP yanıtı durum kodu hakkında bilgi vardır. HTTP yanıt durum kodu, isteğin durumunu açıklamak için kullanılır. Birçok HTTP yanıt durum kodu vardır, ancak bunlar şu şekilde özetlenebilir:

- **100-199** : Bilgilendirici yanıtlar
- **200-299** : Başarılı yanıtlar
- **300-399** : Yönlendirme mesajları
- **400-499** : İstemci hata yanıtları
- **500-599** : Sunucu hatası yanıtları

Yanıt Başlıkları

İşte sık karşılaşabileceğiniz bazı HTTP Yanıt Başlıkları:

- **Tarih** : Sunucunun istemciye HTTP Yanıtı gönderdiği tam saat.
- **Bağlantı** : Tıpkı HTTP İsteği başlığında olduğu gibi bağlantının nasıl işleneceğini belirtir.
- **Sunucu** : Sunucunun işletim sistemi ve web sunucusunun sürümü hakkında bilgi.
- **Last-Modified** : İstenen kaynağın ne zaman değiştirildiği hakkında bilgi. Bu başlık, önbellek mekanizması için kullanılır.
- **İçerik Türü** : Gönderilen veri türü.
- **Content-Length** : Gönderilen verinin boyutu.

Yanıt Gövdesi

HTTP Yanıt Gövdesi, sunucu tarafından gönderilen ve istemci tarafından istenen kaynağı içerir.

```
1  <html>
2      <head>
3          <title> Welcome to LetsDefend</title>
4      </head>
5      <body>
6          <p>Website contents....</p>
7      </body>
8  <html>
```

SQL Enjeksiyonu (SQLi) nedir?

SQL Enjeksiyonları, bir web uygulamasının SQL sorgularında kullanıcı tarafından sağlanan temizlenmemiş verileri doğrudan içerdiği kritik saldırı yöntemleridir.

Bugünlerde web uygulamaları geliştirmek için kullandığımız çerçeveler, SQL Injection saldırılarına karşı koruma sağlayan önleyici mekanizmalara sahiptir. Ama yine de SQL Injection açıklarıyla karşılaşıyoruz çünkü bazen ham SQL sorguları kullanılıyor, bazen çerçevenin doğuştan gelen bir SQL Injection güvenlik açığı var ya da çerçeve doğru şekilde kullanılamıyor.

SQL Enjeksiyon Türleri

3 tip SQL Enjeksiyonu vardır. Bunlar:

1. **In-band SQLi (Klasik SQLi)** : Aynı kanal üzerinden bir SQL sorgusu gönderilmiş ve cevaplanmışsa bunlara In-band SQLi deriz. Diğer SQLi kategorilerine kıyasla saldırganların bunlardan yararlanmaları daha kolaydır.

2. **Çıkarımsal SQLi (Kör SQLi): Görünmeyen** bir yanıt alan SQL sorgularına Çıkarımsal SQLi denir. Cevap görülemediği için Blind SQLi olarak adlandırılırlar.

3. **Out-of-band SQLi** : Bir SQL sorgusuna verilen cevap farklı bir kanal üzerinden iletilirse, bu tip SQLi'ye Out-of-band SQLi denir. Örneğin, saldırgan SQL sorgularına DNS üzerinden yanıt alıyorsa buna bant dışı SQLi denir.

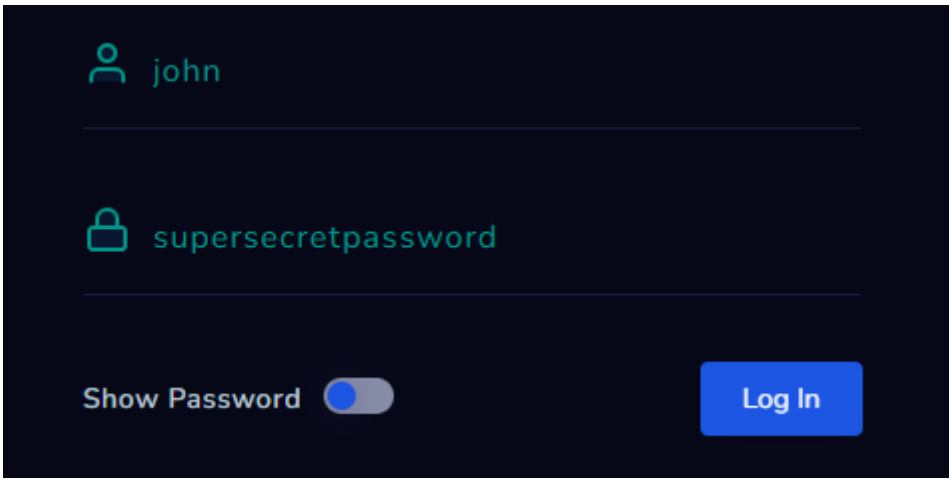
SQL Enjeksiyonu Nasıl Çalışır?

Günümüzde standart web uygulamaları en yaygın olarak bir kullanıcıdan veri alır ve bu verileri belirli içeriği görüntülemek için kullanır. Oturum açma sayfası, çoğu SQL Enjeksiyon saldırısının gerçekleştiği yerdir. SQL enjeksiyonlarının nasıl çalıştığını bir örnek üzerinden inceleyelim.

Bir kullanıcının genellikle giriş sayfasında kullanıcı adını ve şifresini girmesi beklenir. Diğer tarafta, web uygulaması aşağıdaki gibi bir SQL sorgusu oluşturmak için bu kullanıcı adı ve şifre bilgilerini kullanacaktır:

SELECT * FROM users WHERE username = ' **USERNAME** ' AND password = ' **USER_PASSWORD** '

Bu SQL sorgusunun anlamı “ **USERNAME** ve şifresi **USER_PASSWORD** olan users tablosundan kullanıcı ile ilgili tüm bilgileri bana getir ” dir. Web uygulaması eşleşen bir kullanıcı bulursa kullanıcının kimliğini doğrular, sorgu yapıldıktan sonra bir kullanıcı bulamazsa oturum açma başarısız olur.

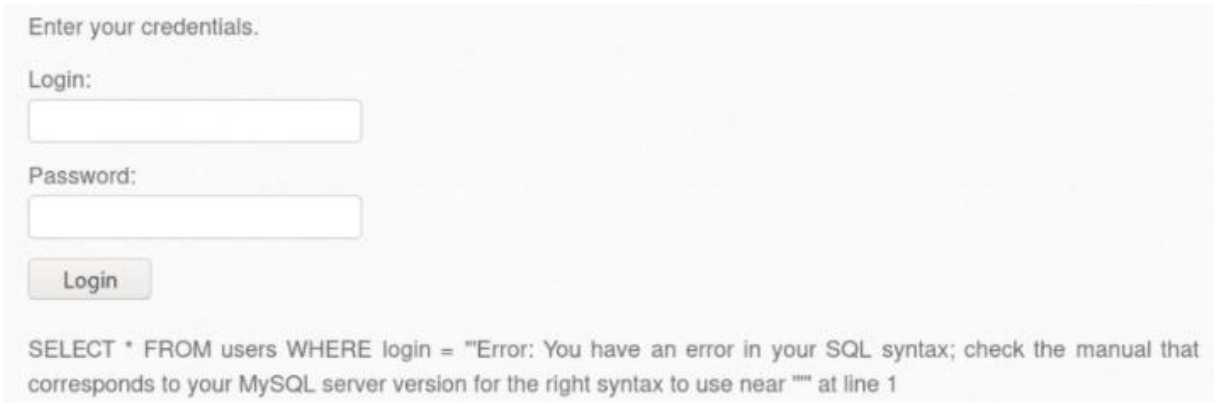


Diyelim ki kullanıcı adınız “ **john** ” ve şifreniz “ **süper gizli** şifre ”. Bu bilgileri girip login butonuna tıkladığınızda aşağıda gördüğünüz SQL sorgusu sorgulanacak ve SQL sorgusu sonrasında bir eşleşme bulunduğu için girebileceksiniz.

SELECT * FROM users WHERE username = ' **john** ' AND password = ' **supersecretpassword** '

Peki ya bu sistemi tasarlandığı gibi kullanmasaydık ve kullanıcı adı alanına kesme işareti (') koysaydık? SQL sorgusu aşağıdaki gibi olacak ve sorgu hatalı olduğu için hata veritabanından çıkarılacaktır.

SELECT * FROM users WHERE username = ' **john** ' AND password = ' **supersecretpassword** '



Saldırgan bir hata mesajı almaktan memnun olacaktır. Saldırgan hem hata mesajındaki bilgileri kendi avantajına değiştirebilir hem de doğru yolda

olduğunu gösterir. Saldırgan, kullanıcı adı alanına aşağıdaki gibi bir yük girerse ne olur?

' VEYA 1=1 – -

Saldırgan yükü gönderdiğinde, web uygulaması aşağıdaki SQL sorgusunu yürütecektir:

SEÇİN * WHERE kullanıcı adı = '' VEYA 1=1 – - AND password = ' **süper gizli** şifre ' olan kullanıcılardan

SQL'de "-- -" den sonra gelen karakterler yorum satırı olarak algılanacaktır. Yani yukarıdaki sorguya bakarsak "-- -" den sonra gelen sorgular bir şey ifade etmez. SQL sorgusunu incelemeye devam etmeden önce işleri basitleştirmek için bu kısmı kaldıralım.

SEÇİN * WHERE kullanıcı adı = '' VEYA 1=1 olan kullanıcılardan

Şimdi yukarıdaki sorgu şuna benziyor: " **kullanıcı adı boşsa veya 1=1** ". Kullanıcı adı alanının boş bırakılıp bırakılmaması çok önemli değil çünkü 1 her zaman 1'e eşittir. Bu nedenle bu sorgu her zaman doğru olacak ve büyük ihtimalle veritabanındaki ilk satırı çağıracaktır. Saldırgan, eşleşme olduğu için web uygulamasına başarılı bir şekilde girebilecektir.

Bu örnek tipik bir SQL enjeksiyon saldırısıdır. **Elbette SQL enjeksiyon saldırıları bu örnekle sınırlı değil, saldırgan xp_cmdshell** gibi SQL komutları yardımıyla sistemde komutları çalıştırmak için SQL'i kullanabilir .

Saldırganlar SQL Enjeksiyon Saldırılarından Nasıl Yararlanır?

SQL Injection saldırılarının neden bu kadar kritik öneme sahip olduğunu anlamak için, SQL Injection saldırısının neden olabileceğine bir göz atalım.

- Kimlik doğrulama atlama
- Komut yürütme

- Hassas verileri sızdırma
- Veritabanı girişlerini oluşturma/silme/güncelleme

SQL Enjeksiyonları Nasıl Önlenir

- **Bir çerçeve kullanın:** elbette sadece bir çerçeve kullanmak, bir SQL Enjeksiyon saldırısını önlemek için yeterli olmayacaktır. Çerçevenin dokümantasyona uygun olarak kullanılması son derece önemlidir.
- **Çerçevenizi güncel** tutun: Kullandığınız çerçeve ile ilgili güvenlik güncellemelerini takip ederek web uygulamanızın güvenliğini sağlayın.

- **Bir kullanıcıdan alınan verileri her zaman sterilize edin: Bir kullanıcıdan** alınan verilere asla güvenmeyin. Bunun da ötesinde, yalnızca form verilerini sterilize etmekle kalmaz, aynı şeyi diğer verilerle (Başlıklar, URL'ler vb.)
- **Ham SQL sorguları kullanmaktan kaçının: Ham SQL sorguları** yazma alışkanlığınız olabilir, ancak bir çerçevenin sağladığı avantajlardan yararlanmayı seçmeli ve sağladığı güvenlikten de yararlanmalısınız.

SQL Enjeksiyon Saldırılarını Tespit Etme

Saldırganların SQL Injection saldırısı ile neler yapabileceğini önceki bölümde tartışmıştık. Yukarıda belirtilen SQL Injection sonuçlarının her biri bir kurum için büyük kayıplara neden olabilir, bu nedenle SOC Analistleri olarak bu saldırıları tespit edebilmemiz ve bunlara karşı önlem alabilmemiz gerekir.

Peki SQL Injection saldırılarını nasıl tespit edebiliriz?

Bu sorunun birden fazla cevabı var. Bunlar:

- **Bir web isteğini incelerken kullanıcıdan gelen tüm alanları kontrol edin:** SQL Injection saldırıları form alanlarıyla sınırlı olmadığı için User-Agent gibi HTTP İstek Başlıklarını da kontrol etmelisiniz.

- **SQL anahtar sözcüklerini** arayın: Kullanıcılardan alınan verilerde INSERT, SELECT, WHERE gibi sözcükleri arayın.

- **Özel karakterleri kontrol edin:** Kullanıcıdan alınan veriler içerisinde SQL'de kullanılan kesme işareti ('), tire (-) veya parantez veya SQL saldırılarında sıklıkla kullanılan özel karakterler olup olmadığına bakın.
- **Sık kullanılan SQL Injection yükleri hakkında bilgi edinin:** SQL yükleri web uygulamasına göre değişse de, saldırganlar SQL Injection güvenlik açıklarını kontrol etmek için hala bazı ortak yükler kullanır. Bu yüklere aşina iseniz, SQL Injection yüklerini kolayca tespit edebilirsiniz. Sık kullanılan bazı SQL Injection yüklerini burada görebilirsiniz .

Otomatik SQL Enjeksiyon Araçlarını Algılama

Saldırganlar, SQL Injection güvenlik açıklarını tespit etmek için birçok otomatikleştirilmiş cihaz kullanır. En iyi bilinenlerden biri Sqlmap'tir. Belirli bir araca odaklanmak yerine daha geniş resme bakalım.

SQL Injection cihazlarını tespit etmek için aşağıda listelenen yöntemleri kullanabilirsiniz:

1. **User-Agent'a bakın:** Otomatik tarayıcı cihazlarının adları ve sürümleri genellikle kaydedilir. Bu otomatik cihazları algılamak için Kullanıcı Aracısına bakabilirsiniz.

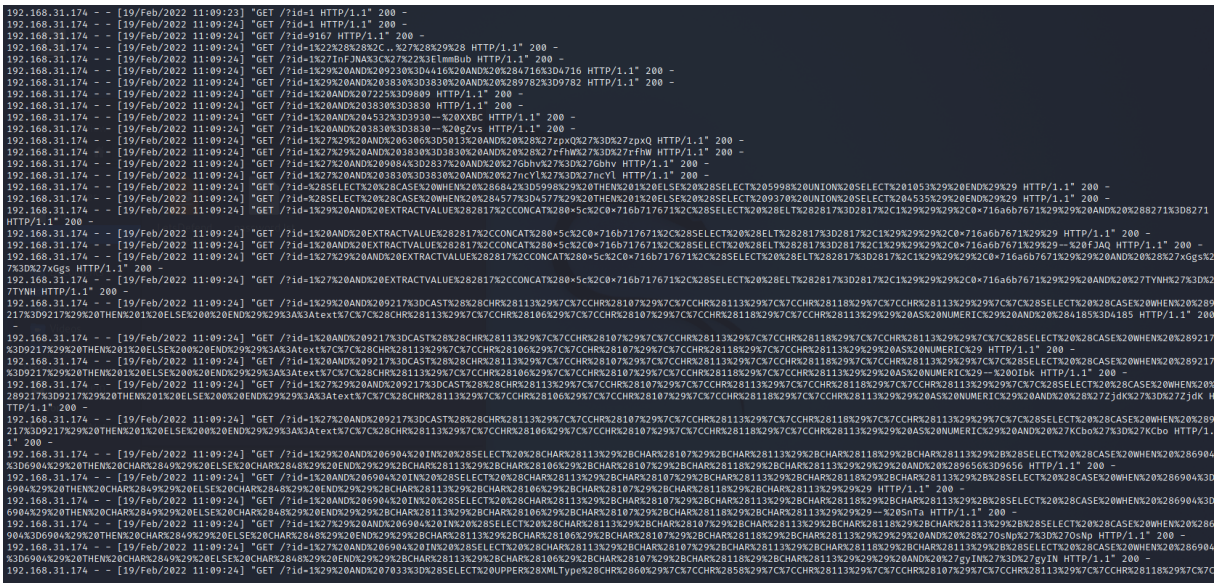
2. **İsteklerin sıklığını kontrol edin:** Otomatik cihazlar, yükleri mümkün olduğu kadar çabuk test edebilmek için, tahmini olarak saniyede çok sayıda istek gönderecek şekilde tasarlanmıştır. Normal bir kullanıcı saniyede 1 istek gönderebilir, böylece isteklerin otomatik bir cihaz tarafından yapılıp yapılmadığını saniyedeki istek sayısına bakarak anlayabilirsiniz.

3. **Yükün içeriğine bakın:** Otomatik cihazlar genellikle kendi adlarını yüklerine kaydeder. Örneğin, otomatikleştirilmiş bir cihaz tarafından gönderilen bir SQL Enjeksiyon yükü şöyle görünebilir: **sqlmap' VEYA 1=1**
4. **Yük karmaşık mı:** Bu algılama yöntemi her zaman işe yaramayabilir, ancak deneyimlerime dayanarak, otomatik cihazların daha karmaşık yükler gönderdiğini söyleyebilirim.

Algılama Örneği

SQL Injection saldırısının kurbanı olan bir web uygulamasının erişim günlüklerine sahibiz.

Erişim günlüğünün ne olduğunu daha önce duymamış olabilirsiniz. Kısacası bunlar web sunucusunun erişim günlükleridir. Bu günlükler genellikle kaynak IP adresini, tarihi, istenen URL'yi, HTTP yöntemini, kullanıcı aracısını ve HTTP Yanıt kodunu içerir. Bu günlükler araştırmalarda çok faydalıdır.



(SQL Enjeksiyon Erişim Günlükleri)

Elimizde bir erişim kaydı var. Şimdi ne yapıyoruz?

Öncelikle talep edilen sayfalara baktığımızda oldukça okunaklı olan “info.php” gibi sayfaların yanı sıra karmaşık ve % gibi sembolleri olan sayfalara da istekte bulunulduğunu görüyoruz. Bu tür sayfalara yapılan isteklerin kötü niyetli olduğunu söyleyemeyiz ama defalarca ve defalarca yapılması şüphelidir.

Öncelikle % sembollerinin ne anlama geldiğinden bahsedelim. Özel karakterler içeren bir sayfa talep ettiğimizde bu istekler doğrudan web sunucusuna aktarılmaz. Bunun yerine, tarayıcılarımız özel karakterlerin bir URL kodlamasını (Yüzde Kodlama) gerçekleştirir ve her özel karakteri % ile başlayan ve 2 onaltılık karakter içeren bir karakter dizisiyle değiştirir. Yani yukarıdaki % sembolünü içeren sayfalar özel karakterler içeren sayfalardır.

Character	From Windows-1252	From UTF-8
space	%20	%20
!	%21	%21
"	%22	%22
#	%23	%23
$	%24	%24
%	%25	%25
&	%26	%26
'	%27	%27
(	%28	%28
)	%29	%29

Artık % sembollerinin ne anlama geldiğini anladığımıza göre, erişim günlüklerini tekrar gözden geçirelim. Taleplere baktığımızda % sembollerinin yanında “UNION”, “SELECT”, “AND”, “CHR” gibi okunabilir kelimelerin olduğunu rahatlıkla görebiliriz. Bunlar SQL'e ait spesifik kelimeler olduğu için bir SQL Injection saldırısı ile karşı karşıya olduğumuzu belirleyebiliriz.

Gözümüze sokmamak için, incelemeyi biraz daha kolaylaştıralım :) "Online URL Decoder" anahtar kelimelerini kullanarak arama yaparak URL kodunu sizin yerinize otomatik olarak yapacak web uygulamalarını bulabilirsiniz. Bu erişim kayıtlarını daha kolay okuyabilmek için bu web uygulamalarından yardım alacağım, böylece ne gözlerimi ne de sizin gözlerinizi yormuş olacağım.

Küçük bir not ekleyeyim. Bir 3. taraf web uygulamasına kritik bilgiler içeren erişim günlükleri gibi bir şey yüklemek akıllıca değildir. Yüklediğim erişim logları bu eğitim için özel olarak hazırlandı, bu yüzden yapmamda bir sorun yok. Ancak profesyonel yaşamınızda bu tür hatalar yapmamalısınız.

URL Decoder/Encoder

URL kod çözme işlemini yaptığımızda bunun bir SQL Injection saldırısı olduğunu daha net görebiliriz. Öyleyse şimdi ne yapmalıyız? Evet, bunun bir SQL Injection saldırısı olduğunu doğruladık ama onu orada mı bırakacağız?

Tabii ki değil. Şimdi bu erişim günlüklerinden alabildiğimiz diğer bilgileri bulacağız.

URL Decoder/Encoder

Öncelikle talep tarihlerine bakalım. Tüm SQL Injection yükleri “19/Feb/2022 11:09:24” tarihinde gönderildi. 1 saniyede 50'den fazla istek yapıldığını görebiliyoruz. Bu kadar kısa sürede bu kadar çok talebin

yapılmış olması bize bunun otomatikleştirilmiş bir saldırı olduğunu gösteriyor. Ek olarak, daha önce de belirttiğimiz gibi, saldırganlar manuel testler yaparken önce kolay yükleri test etmeyi seçerler. Ancak erişim loglarına baktığımızda, yüklerin çok karmaşık olduğunu görüyoruz. Bu, saldırının çok iyi otomatikleştirilebileceğini gösteriyor.

Bir SQL Injection saldırısının gerçekleştirildiğini ve otomatik bir cihazla gerçekleştirildiğini doğruladık. Böylece analizimizi sonlandırabiliriz, değil mi?

Yapılacak bir adım daha kaldı. Saldırının başarılı olup olmadığını belirlememiz gerekiyor. Bir SQL Injection saldırısının başarılı olup olmadığını cevaba bakarak anlayabilirsiniz, ancak profesyonel kariyerinizde cevaba neredeyse hiçbir zaman erişemeyeceksiniz. Saldırı aynı sayfada ve “id” değişkeni üzerinden gerçekleştirildiği için tüm yanıtların aşağı yukarı aynı boyutta olacağını varsayabiliriz. Tepkinin boyutuna bakarak saldırının başarısını tahmin edebiliriz.

Örnek olması için geliştirilen temel web sunucusu ne yazık ki güvenilir bir yanıt boyutu sağlayamıyor. Bu nedenle, bu örneğe bakarak saldırının başarılı olup olmadığını tahmin edemeyiz. Ancak doğru yapılandırılmış web sunucuları ile erişim günlüklerinde yanıt boyutunu bulabiliriz. Yanıt boyutlarında kayda değer bir fark olup olmadığını belirlemek için bu alanı inceleyebilirsiniz. Belirgin bir fark varsa, saldırının başarılı olduğunu tahmin edebilirsiniz. Ancak bu durumda, bu uyarıyı daha üst düzey bir analiste iletmek en iyisi olacaktır.

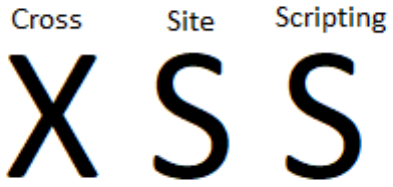
Ne biliyoruz:

1. Web uygulamasının ana sayfasındaki “id” parametresine SQL Injection saldırısı yapılmıştır.
2. İstekler şu IP adresinden geldi: 192.168.31.174.
3. Saniyede 50'den fazla istek olduğundan, bu saldırı otomatik bir güvenlik açığı tarama aracı tarafından gerçekleştirildi.
4. Yüklerin karmaşık yapısı 3. maddedeki iddiayı desteklemektedir.

5. Yanıt boyutu hakkında herhangi bir bilgimiz olmadığı için yanıtın başarılı olup olmadığını belirleyemiyoruz.

Siteler Arası Komut Dosyası Çalıştırma (XSS) nedir?

Siteler Arası Komut Dosyası Çalıştırma (XSS), meşru web uygulamalarına dahil olan ve kötü amaçlı kodların çalıştırılmasına olanak tanıyan, enjeksiyon tabanlı bir web güvenlik açığı türüdür.



Günümüzde web uygulamaları geliştirmek için kullanılan çerçevelerin çoğu, siteler arası komut dosyası çalıştırma saldırılarına karşı önleyici tedbirler almıştır. Ancak, çerçeveler bazen kullanılmadığından veya çerçevenin kendisinde bir XSS güvenlik açığı bulunduğundan ve kullanıcıdan gelen veriler temizlenmediğinden bugün hala sık sık XSS güvenlik açıkları görüyoruz.

XSS Türleri

3 farklı XSS türü vardır. Bunlar:

1. **Reflected XSS (Non-Persistent)** : XSS yükünün istekte içermesi gereken kalıcı olmayan bir XSS türüdür. En yaygın XSS türüdür.

2. **Stored XSS (Persistent)** : Saldırganın XSS yükünü web uygulamasına kalıcı olarak yükleyebildiği bir XSS türüdür. Diğer türlere kıyasla en tehlikeli XSS türü Stored XSS'dir.

3. **DOM Tabanlı XSS** : DOM Tabanlı XSS, saldırı yükünün, kurbanın orijinal istemci tarafı komut dosyası tarafından kullanılan tarayıcısındaki DOM "ortamının" değiştirilmesinin bir sonucu olarak yürütüldüğü bir XSS saldırısıdır, böylece istemci tarafı kodu "beklenmeyen bir şekilde çalışır". " tavır. (OWASP)

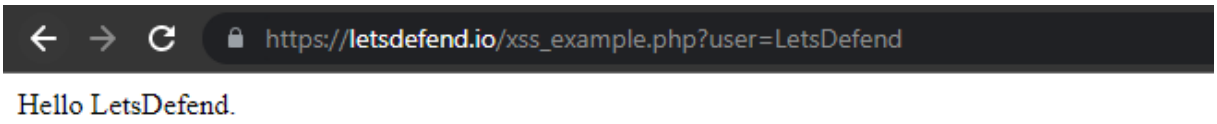
XSS Nasıl Çalışır?

Diğer web saldırı yöntemleri gibi, XSS de veri temizliği eksikliğinden kaynaklanan bir güvenlik açığıdır. XSS güvenlik açığı, kullanıcıdan alınan veriler sterilize edilmeden yanıt olarak gönderildiğinde ortaya çıkar.

XSS saldırılarını daha iyi anlamak için bir örnek izleyelim.

```
<p>Hello <?php echo $_GET['user']; ?>.</p>
```

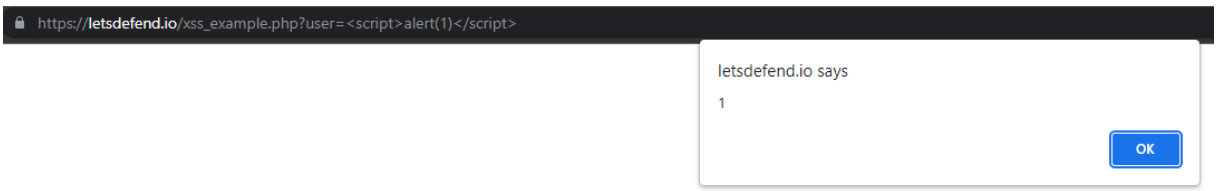
Yukarıdaki kod parçasına bakalım. Yaptığı şey aslında çok basit. Yalnızca 'user' parametresine girilenleri görüntüler. Kullanıcı parametresi olarak “LetsDefend” girersek “Hello LetsDefend” yazısını göreceğiz.



Şu ana kadar herhangi bir sorun yok. Kullanıcı parametresine uygun verileri girersek sıcak bir selam ile karşılanırız. Ancak yukarıda gördüğümüz gibi kullanıcı parametresi için bir kontrol mekanizması yoktur. Bu, “user” parametresine girdiğimiz her şeyin, geri aldığımız HTTP Yanıtına dahil edileceği anlamına gelir.

Peki, normal bir değer girmeyip bunun yerine bir pop-up çağıracak bir payload girersek ne olur?

Yük: **<script>uyarı(1)</script>**

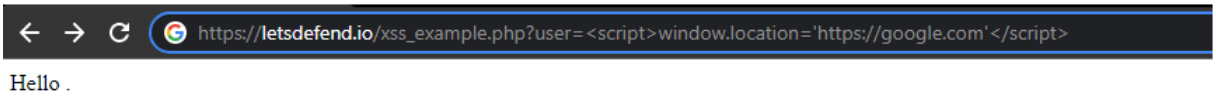


“user” parametresine ne girdiysek direk HTTP Response içerisinde yer aldığı için yazdığımız javascript kodu çalıştı ve ekrana bir pop-up penceresi çıktı.

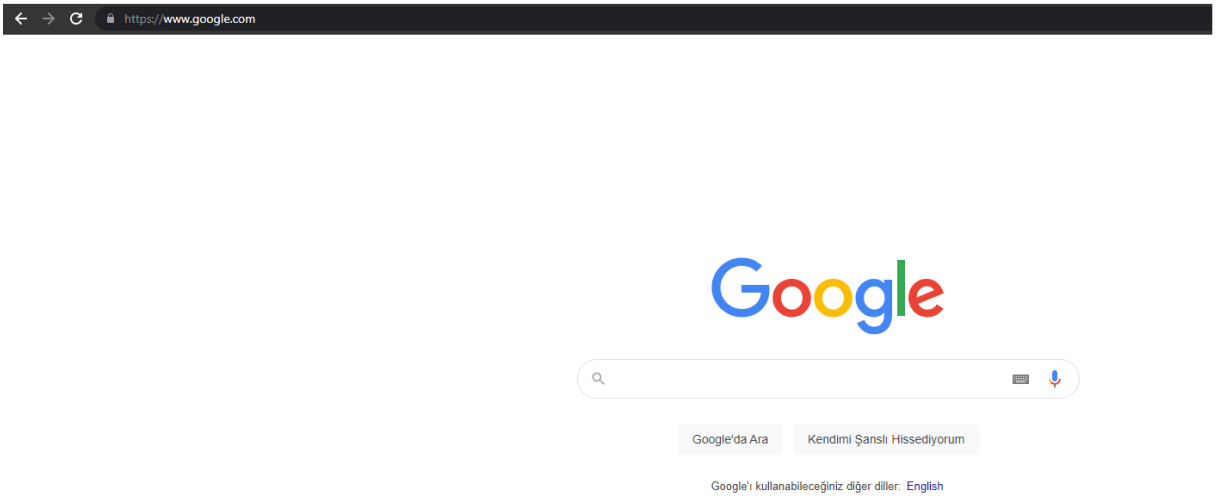
Yani, XSS tam olarak böyle çalışır. Kullanıcı tarafından girilen değer onaylanmadığı için saldırgan istediği javascript kodunu girerek istediği sonucu alabilir. Saldırgan kullanıcıyı kötü niyetli bir siteye yönlendirmek isterse ne olur?

Yük: **<script>window.location='https://google.com'</script>**

https://letsdefend.io/xss_example.php?user=%3Cscript%3Ewindow.location=%27https://google.com%27%3C/script%3E



Elbette sizi bir web uygulamasına yönlendirmeyeceğiz. Örnek olarak sizi Google'a yönlendirmek yeterli olacaktır. Kullanıcı URL'ye tıkladığında, mükemmel LetsDefend web uygulaması yerine Google'a yönlendirilecektir.



Saldırganlar XSS Saldırılarından Nasıl Yararlanır?

XSS istemci tabanlı bir saldırı yöntemi olduğundan, diğer saldırı yöntemlerinden daha az önemli görünebilir ancak XSS saldırıları ve etkileri hafife alınmamalıdır.

Saldırganlar bir XSS saldırısıyla şunları yapabilir:

- Bir kullanıcının oturum bilgilerini çalma
- Bir kullanıcının yapabileceği işlemleri başlatma

- Kimlik bilgilerini yakala

…ve diğer çeşitli işlevler.

XSS Güvenlik Açığı Nasıl Önlenir

- **Bir kullanıcıdan gelen verileri dezenfekte edin: Bir kullanıcıdan** gelen verilere asla güvenmeyin. Kullanıcı verilerinin işlenmesi ve kaydedilmesi gerekiyorsa, özel karakterler kullanılarak html kodlaması ile kodlanmalı ve ancak o zaman kaydedilmelidir.

- **Bir çerçeve kullanın:** Çoğu çerçeve, XSS saldırılarına karşı önleyici tedbirlerle birlikte gelir.

- **Çerçeveyi doğru kullanın:** Web uygulamaları geliştirmek için kullanılan hemen hemen tüm çerçeveler bir sanitasyon özelliğine sahiptir, ancak bu düzgün kullanılmazsa, yine de XSS güvenlik açıklarının oluşma olasılığı vardır.

- **Çerçevenizi güncel tutun:** Çerçeveler insanlar tarafından geliştirildiğinden, onlar da XSS güvenlik açıkları içerebilir. Ancak bu tür güvenlik açıkları genellikle güvenlik güncellemeleriyle kapatılır. Bu yüzden çerçevenizin güvenlik güncellemelerini tamamladığınızdan emin olmalısınız.

XSS Saldırılarını Tespit Etme

Bir önceki yazıda bahsettiğimiz gibi Acunetix tarafından yapılan bir araştırmaya göre siber saldırıların %75'i web uygulamaları üzerinden gerçekleştiriliyor. XSS, en sık test edilen güvenlik açıklarından biri

olduğundan, bir SOC analisti olarak kariyeriniz boyunca bunlardan çokça göreceksiniz.

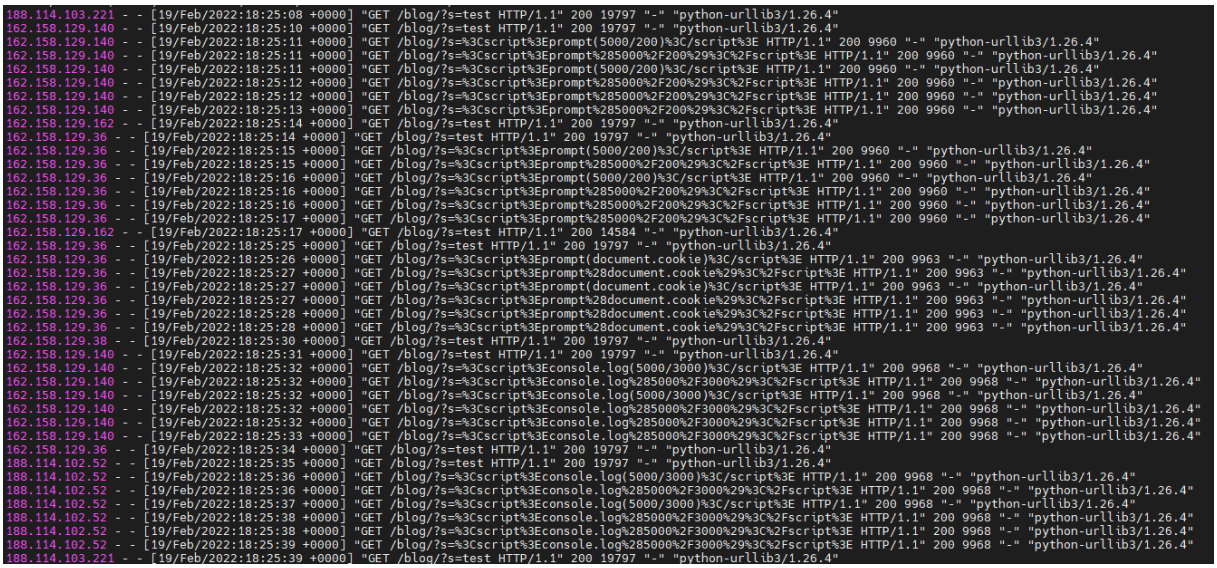
- **Anahtar kelimeleri arayın:** XSS saldırılarını yakalamanın en kolay yolu, XSS yüklerinde yaygın olarak kullanılan "alert" ve "script" gibi anahtar kelimeleri aramaktır.

- **Sık kullanılan XSS yükleri hakkında bilgi edinin:** Saldırganlar, bir XSS güvenlik açığından yararlanmadan önce güvenlik açıklarını aramak için öncelikle aynı yükleri kullanır. Bu nedenle, sık kullanılan XSS yüklerini tanımanız, XSS güvenlik açıklarını tespit etmenizi kolaylaştıracaktır. Sık kullanılan bazı faydalı yükleri buradan inceleyebilirsiniz .

- **Herhangi bir özel karakterin kullanılıp kullanılmadığını** kontrol edin: Bir kullanıcıdan gelen verileri kontrol ederek XSS veri yüklerinde (>) veya daha küçük (<) gibi sık kullanılan özel karakterlerin olup olmadığını kontrol edin.

Algılama Örneği

Bu örnekte, Wordpress ile bir Apache sunucusundan erişim günlüklerini görüyoruz. Erişim günlükleri hakkında daha fazla bilgi için “SQL Enjeksiyon Saldırılarını Tespit Etme” makalemizi tekrar ziyaret etmeyi unutmayın.



Şimdi, sağlanan erişim günlüklerini inceleyelim.

Öncelikle yapılan taleplere genel bir göz atalım ve bunları anlamaya çalışalım. “/blog/” sayfası için tüm isteklerin yapıldığını ve sadece “s” parametre değerlerinin değiştirildiğini görüyoruz. Ziyaret ettiğiniz web sayfalarının URL'lerine dikkat ederseniz, Wordpress'te arama yaptığınızda girdiğiniz kelimelerin “?s=" parametresi kullanılarak gönderildiğini fark etmişsinizdir. Baktığımız örnek, bunların Wordpress'te yapılan aramalar olduğunu gösteriyor.

“SQL Enjeksiyon Saldırılarını Tespit Etme” makalesindeki örnek gibi kolay okunabilir örnekler bulmak zor. Bunun yerine, URL kodlaması sonucunda %XX'e dönüşen karakterler buluyoruz. Sırada URL kod çözme yapacağız ama önce URL'lere bir göz atalım ve herhangi bir kelimeyi tanıyıp tanıyamayacağımızı görmeye çalışalım.

Loglara baktığımızda “script”, “prompt”, “console.log” gibi javascript ile ilgili kelimeler görüyoruz. Javascript'i gördüğümüzde hemen XSS'yi akla getiriyor. Bir URL kod çözme yaparsak, yapılan istekleri kolayca anlayabileceğiz.

URL Decoder/Encoder

```
188.114.103.221 - - [19/Feb/2022:18:25:08  0000] "GET /blog/?s=test HTTP/1.1" 200 19797 "-" "python-urllib3/1.26.4"
162.158.129.140 - - [19/Feb/2022:18:25:10  0000] "GET /blog/?s=test HTTP/1.1" 200 19797 "-" "python-urllib3/1.26.4"
162.158.129.140 - - [19/Feb/2022:18:25:11  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.140 - - [19/Feb/2022:18:25:11  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.140 - - [19/Feb/2022:18:25:11  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.140 - - [19/Feb/2022:18:25:12  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.140 - - [19/Feb/2022:18:25:12  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.140 - - [19/Feb/2022:18:25:13  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.162 - - [19/Feb/2022:18:25:14  0000] "GET /blog/?s=test HTTP/1.1" 200 19797 "-" "python-urllib3/1.26.4"
162.158.129.36 - - [19/Feb/2022:18:25:14  0000] "GET /blog/?s=test HTTP/1.1" 200 19797 "-" "python-urllib3/1.26.4"
162.158.129.36 - - [19/Feb/2022:18:25:15  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.36 - - [19/Feb/2022:18:25:15  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.36 - - [19/Feb/2022:18:25:16  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.36 - - [19/Feb/2022:18:25:16  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.36 - - [19/Feb/2022:18:25:16  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.36 - - [19/Feb/2022:18:25:17  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.162 - - [19/Feb/2022:18:25:17  0000] "GET /blog/?s=test HTTP/1.1" 200 14584 "-" "python-urllib3/1.26.4"
162.158.129.36 - - [19/Feb/2022:18:25:25  0000] "GET /blog/?s=test HTTP/1.1" 200 19797 "-" "python-urllib3/1.26.4"
162.158.129.36 - - [19/Feb/2022:18:25:26  0000] "GET /blog/?s=<script>prompt(document.cookie)</script> HTTP/1.1" 200 9963 "-" "python-urllib3/1.26.4"
162.158.129.36 - - [19/Feb/2022:18:25:27  0000] "GET /blog/?s=<script>prompt(document.cookie)</script> HTTP/1.1" 200 9963 "-" "python-urllib3/1.26.4"
162.158.129.36 - - [19/Feb/2022:18:25:27  0000] "GET /blog/?s=<script>prompt(document.cookie)</script> HTTP/1.1" 200 9963 "-" "python-urllib3/1.26.4"
162.158.129.36 - - [19/Feb/2022:18:25:27  0000] "GET /blog/?s=<script>prompt(document.cookie)</script> HTTP/1.1" 200 9963 "-" "python-urllib3/1.26.4"
162.158.129.36 - - [19/Feb/2022:18:25:28  0000] "GET /blog/?s=<script>prompt(document.cookie)</script> HTTP/1.1" 200 9963 "-" "python-urllib3/1.26.4"
162.158.129.36 - - [19/Feb/2022:18:25:28  0000] "GET /blog/?s=<script>prompt(document.cookie)</script> HTTP/1.1" 200 9963 "-" "python-urllib3/1.26.4"
162.158.129.38 - - [19/Feb/2022:18:25:30  0000] "GET /blog/?s=test HTTP/1.1" 200 19797 "-" "python-urllib3/1.26.4"
162.158.129.140 - - [19/Feb/2022:18:25:31  0000] "GET /blog/?s=test HTTP/1.1" 200 19797 "-" "python-urllib3/1.26.4"
162.158.129.140 - - [19/Feb/2022:18:25:32  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
162.158.129.140 - - [19/Feb/2022:18:25:32  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
162.158.129.140 - - [19/Feb/2022:18:25:32  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
162.158.129.140 - - [19/Feb/2022:18:25:32  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
162.158.129.140 - - [19/Feb/2022:18:25:32  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
162.158.129.140 - - [19/Feb/2022:18:25:33  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
162.158.129.36 - - [19/Feb/2022:18:25:34  0000] "GET /blog/?s=test HTTP/1.1" 200 19797 "-" "python-urllib3/1.26.4"
188.114.102.52 - - [19/Feb/2022:18:25:35  0000] "GET /blog/?s=test HTTP/1.1" 200 19797 "-" "python-urllib3/1.26.4"
188.114.102.52 - - [19/Feb/2022:18:25:36  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
188.114.102.52 - - [19/Feb/2022:18:25:36  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
188.114.102.52 - - [19/Feb/2022:18:25:37  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
188.114.102.52 - - [19/Feb/2022:18:25:38  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
188.114.102.52 - - [19/Feb/2022:18:25:38  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
188.114.102.52 - - [19/Feb/2022:18:25:39  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
188.114.103.221 - - [19/Feb/2022:18:25:39  0000] "GET /blog/?s=test HTTP/1.1" 200 19797 "-" "python-urllib3/1.26.4"
```

Bir URL kod çözme işlemi gerçekleştirdikten sonra erişim günlüklerine tekrar baktığımızda XSS yüklerini açıkça görüyoruz. Bu erişim kayıtlarını aldığımız Wordpress uygulamasının bir XSS saldırısının kurbanı olduğunu kesinlikle söyleyebiliriz.

İstenen IP adreslerine baktığımızda birden fazla olduğunu görüyoruz. Birden fazla saldırgan aynı anda XSS saldırısı mı gerçekleştirmeye çalışıyor? Yoksa saldırgan, güvenlik duvarları ve IPS gibi güvenlik ürünleri tarafından engellenmemek için IP adresini sürekli olarak mı değiştiriyor? IP adresini kontrol ederseniz, bunun Cloudflare'a ait olduğunu göreceksiniz. Wordpress uygulaması Cloudflare'in arkasına konulduğu için Cloudflare'in istekte bulunması oldukça normal.

URL Decoder/Encoder

```
188.114.103.221 -  [19/Feb/2022:18:25:08  0000] "GET /blog/?s=test HTTP/1.1" 200 19797 "-" "python-urllib3/1.26.4"
162.158.129.140 -  [19/Feb/2022:18:25:10  0000] "GET /blog/?s=test HTTP/1.1" 200 19797 "-" "python-urllib3/1.26.4"
162.158.129.140 -  [19/Feb/2022:18:25:11  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.140 -  [19/Feb/2022:18:25:11  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.140 -  [19/Feb/2022:18:25:11  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.140 -  [19/Feb/2022:18:25:12  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.140 -  [19/Feb/2022:18:25:12  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.140 -  [19/Feb/2022:18:25:13  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.162 -  [19/Feb/2022:18:25:14  0000] "GET /blog/?s=test HTTP/1.1" 200 19797 "-" "python-urllib3/1.26.4"
162.158.129.36 - -[19/Feb/2022:18:25:14  0000] "GET /blog/?s=test HTTP/1.1" 200 19797 "-" "python-urllib3/1.26.4"
162.158.129.36 - -[19/Feb/2022:18:25:15  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.36 - -[19/Feb/2022:18:25:15  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.36 - -[19/Feb/2022:18:25:16  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.36 - -[19/Feb/2022:18:25:16  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.36 - -[19/Feb/2022:18:25:16  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.36 - -[19/Feb/2022:18:25:17  0000] "GET /blog/?s=<script>prompt(5000/200)</script> HTTP/1.1" 200 9960 "-" "python-urllib3/1.26.4"
162.158.129.162 -  [19/Feb/2022:18:25:17  0000] "GET /blog/?s=test HTTP/1.1" 200 14584 "-" "python-urllib3/1.26.4"
162.158.129.36 - -[19/Feb/2022:18:25:25  0000] "GET /blog/?s=test HTTP/1.1" 200 19797 "-" "python-urllib3/1.26.4"
162.158.129.36 - -[19/Feb/2022:18:25:26  0000] "GET /blog/?s=<script>prompt(document.cookie)</script> HTTP/1.1" 200 9963 "-" "python-urllib3/1.26.4"
162.158.129.36 - -[19/Feb/2022:18:25:27  0000] "GET /blog/?s=<script>prompt(document.cookie)</script> HTTP/1.1" 200 9963 "-" "python-urllib3/1.26.4"
162.158.129.36 - -[19/Feb/2022:18:25:27  0000] "GET /blog/?s=<script>prompt(document.cookie)</script> HTTP/1.1" 200 9963 "-" "python-urllib3/1.26.4"
162.158.129.36 - -[19/Feb/2022:18:25:27  0000] "GET /blog/?s=<script>prompt(document.cookie)</script> HTTP/1.1" 200 9963 "-" "python-urllib3/1.26.4"
162.158.129.36 - -[19/Feb/2022:18:25:28  0000] "GET /blog/?s=<script>prompt(document.cookie)</script> HTTP/1.1" 200 9963 "-" "python-urllib3/1.26.4"
162.158.129.36 - -[19/Feb/2022:18:25:28  0000] "GET /blog/?s=<script>prompt(document.cookie)</script> HTTP/1.1" 200 9963 "-" "python-urllib3/1.26.4"
162.158.129.38 - -[19/Feb/2022:18:25:30  0000] "GET /blog/?s=test HTTP/1.1" 200 19797 "-" "python-urllib3/1.26.4"
162.158.129.140 -  [19/Feb/2022:18:25:31  0000] "GET /blog/?s=test HTTP/1.1" 200 19797 "-" "python-urllib3/1.26.4"
162.158.129.140 -  [19/Feb/2022:18:25:32  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
162.158.129.140 -  [19/Feb/2022:18:25:32  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
162.158.129.140 -  [19/Feb/2022:18:25:32  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
162.158.129.140 -  [19/Feb/2022:18:25:32  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
162.158.129.140 -  [19/Feb/2022:18:25:32  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
162.158.129.140 -  [19/Feb/2022:18:25:33  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
162.158.129.36 - -[19/Feb/2022:18:25:34  0000] "GET /blog/?s=test HTTP/1.1" 200 19797 "-" "python-urllib3/1.26.4"
188.114.102.52 - -[19/Feb/2022:18:25:35  0000] "GET /blog/?s=test HTTP/1.1" 200 19797 "-" "python-urllib3/1.26.4"
188.114.102.52 - -[19/Feb/2022:18:25:36  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
188.114.102.52 - -[19/Feb/2022:18:25:36  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
188.114.102.52 - -[19/Feb/2022:18:25:37  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
188.114.102.52 - -[19/Feb/2022:18:25:38  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
188.114.102.52 - -[19/Feb/2022:18:25:38  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
188.114.102.52 - -[19/Feb/2022:18:25:39  0000] "GET /blog/?s=<script>console.log(5000/3000)</script> HTTP/1.1" 200 9968 "-" "python-urllib3/1.26.4"
188.114.103.221 -  [19/Feb/2022:18:25:39  0000] "GET /blog/?s=test HTTP/1.1" 200 19797 "-" "python-urllib3/1.26.4"
```

Taleplerin tarihlerini incelediğimizde 3-4 saniyede bir istek yapıldığını görüyoruz. Bir insanın bu kadar çok XSS yükünü bu kadar kısa sürede girmeye çalışması gerçekten mümkün değil ama saniyede yapılan istek sayısının aşırı olduğundan emin olamayabilirsiniz. Şanslıyız çünkü bu örnekte User-Agent bilgisine sahibiz. Bu bilgiyi incelersek bir urllib kütüphanesine ait olduğunu görürüz. Bu bize bu isteklerin otomatik bir güvenlik açığı tarayıcı aracı aracılığıyla yapıldığını gösteriyor.

Peki saldırı başarılı oldu mu?

Cevaplara ulaşamadığımız için kesin bir şey söyleyemeyiz.

İncelemelerimiz sonucunda:

1. Saldırının, erişim günlüklerinin geldiği web uygulamasını hedef aldığı belirlendi.
2. Talep miktarına ve User-Agent bilgilerine baktıktan sonra saldırının otomatik bir güvenlik açığı tarayıcısı tarafından gerçekleştirildiğini belirledik.

3. Uygulama Cloudflare'nin arkasında olduğu için kaynak IP adresleri bulunamadı.
4. Saldırının başarılı olup olmadığını bilmiyoruz.

Komut Enjeksiyon Saldırıları nedir?

Komut Enjeksiyon Saldırıları, bir kullanıcıdan alınan veriler temizlenmediğinde ve doğrudan işletim sistemi kabuğuna iletildiğinde gerçekleşen saldırılardır.

Saldırganlar, komutları doğrudan işletim sisteminde yürütmek için komut enjeksiyon güvenlik açıklarından yararlanır. Saldırganın önceliğinin sistemin kontrolünü ele geçirmek olması bu zafiyetleri diğer zafiyetlere göre daha kritik hale getirmektedir.

Saldırganın gönderdiği komut, web uygulaması kullanıcısının haklarını kullanacağından, yanlış yapılandırılmış bir web uygulaması, saldırgana yönetici haklarıyla erişim sağlar.

Komut Enjeksiyonu Nasıl Çalışır?

Komut enjeksiyon güvenlik açıkları, kullanıcıdan alınan veriler temizlenmediğinde meydana gelir. Bir örnekle komut enjeksiyon güvenlik açıklarını inceleyelim.

Diyelim ki kullanıcının dosyasını “/tmp” klasörüne kopyalayan basit bir web uygulamamız var. Web uygulamasının kodu aşağıdadır.

```php
<?php

$file_name = $_POST['file_name']
$output = shell_exec('cp $file_name /tmp/');

?>
```

Normal koşullar altında, doğru kullanıldığında uygulama normal şekilde çalışacaktır. Örneğin “letsdefend.txt” isimli bir dosya yüklersek, dosyayı başarıyla “/tmp” klasörüne kopyalayacaktır.

Peki “letsdefend;ls;.txt” isimli bir dosya yüklersek ne olur? Komut şöyle olur:

Komut: **cp allowdefend;ls;.txt**

“;” komutun bittiğini belirtir. Yani yukarıdaki payload'a baktığımızda işletim sisteminin yürüttüğü üç farklı komut var. Bunlar:

1. cp savunmaya izin ver
2. ls
3. .Txt

```
└─$ cp letsdefend;ls;.txt
cp: missing destination file operand after 'letsdefend'
Try 'cp --help' for more information.
crowbar.log  Desktop    Downloads  Pictures  Templates  words.txt
crowbar.out  Documents  Music      Public    Videos
.txt: command not found
```

İlk komut kopyalama işlemi içindir ancak parametreler doğru girilmezse düzgün çalışmayacaktır.

Komut #2, saldırganın yürütmek istediği dizin listeleme komutudur. Kullanıcı komut çıktısını almaz, bu nedenle saldırgan dizindeki dosyaları göremez ancak işletim sistemi komutu başarıyla yürütür.

İşletim sistemi 3 numaralı komutu çalıştırmak istediğinde ".txt" komutu olmadığından hata mesajı çıkacaktır.

Gördüğünüz gibi, kod web sunucusunun işletim sisteminde yürütülmüştür. Peki ya saldırgan "letsdefend;shutdown;.txt" adlı bir dosya yüklerse? İşletim sistemi kendini kapatır ve web uygulaması çalışamaz.

Saldırgan, doğru payload yardımıyla işletim sisteminde bir ters kabuk oluşturabilir.

Saldırganlar Komut Enjeksiyonu Saldırılarından Nasıl Yararlanır?

Saldırganlar, komut enjeksiyon güvenlik açıklarından yararlanarak bir işletim sisteminde komut çalıştırabilir. Bu, web uygulamasının ve sunucudaki diğer tüm bileşenlerin risk altında olduğu anlamına gelir.

Komut Enjeksiyonu Nasıl Önlenir

- **Bir kullanıcıdan alınan verileri her zaman sterilize edin: Bir kullanıcıdan** alınan verilere asla güvenmeyin. Dosya adı bile yok!

- **Kullanıcı haklarını sınırlayın:** Mümkün olduğunda web uygulaması kullanıcı haklarını daha düşük bir düzeye ayarlayın. Neredeyse hiçbir web uygulaması, kullanıcının yönetici haklarına sahip olmasını gerektirmez.

- **Liman işçileri gibi sanallaştırma teknolojilerinden yararlanın**

Komut Enjeksiyon Saldırılarını Tespit Etme

Sanırım hepimiz Command Injection güvenlik açığının kritiklik düzeyini çok iyi anlıyoruz. Böyle kritik bir güvenlik açığından yararlanılır ve tespit edilmezse, ilgili şirket büyük miktarda para ve itibar kaybedebilir.

Peki Komut Enjeksiyon Saldırılarını nasıl tespit edebiliriz?

Birden fazla yol var. Bunlar:

- **Bir web isteğini incelerken tüm alanlara bakın:** Komut enjeksiyon güvenlik açığı, web uygulamasının çalışmasına bağlı olarak çeşitli alanlarda bulunabilir. Bu nedenle, web isteğinin tüm alanlarını kontrol etmelisiniz.

- **Terminal diliyle ilgili anahtar kelimeleri arayın** : dir, ls, cp, cat, type, vb. gibi terminal komutlarıyla ilgili anahtar kelimeler için kullanıcıdan alınan verileri kontrol edin.

- **Sık kullanılan Command Injection yükleri hakkında bilgi edinin:** Saldırganlar bir komut ekleme güvenlik açığı tespit ettiğinde, daha kolay çalışmak için genellikle bir ters kabuk oluştururlar. Bu nedenle, sık kullanılan Komut Ekleme yüklerini bilmek, bir komut enjeksiyon saldırısını tespit etmeyi kolaylaştıracaktır.

Algılama Örneği

Bu örnekte erişim günlüklerine bakmayacağız, bunun yerine bir HTTP İsteğini inceleyeceğiz.

GET / HTTP/1.1

Ev sahibi: sirketiniz.com

Kullanıcı Aracısı: () { :;}; echo "NS:" $(</etc/passwd)

Kabul et: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3; q=0.9

Kabul-Kodlama: gzip, deflate

Kabul Et-Dil: tr-US,tr;q=0.9

Bağlantı: kapat

Yukarıdaki HTTP İsteğine bakarsak, yourcompany[.]com web uygulamasının ana sayfasının talep edildiğini görüyoruz.

Ancak HTTP İstek Başlıklarına baktığımızda User-Agent başlığında şüpheli bir durum görüyoruz. User-Agent başlığında bir bash komutu varken burada tarayıcı/işletim sistemi bilgisi bulunmalıdır.

Aslında bu istek, Shellshock adlı bir güvenlik açığından yararlanılırken yakalandı. Shellshock, 2014 yılında yayınlanan ve büyük etkileri olan bir güvenlik zafiyetidir.

Shellshock, bash'ın bir şekilde Ortam Değişkenlerini istemeden yürütmesinden kaynaklanan bir güvenlik açığıdır. Shellshock, komut enjeksiyon saldırısının harika bir örneğidir.

User-Agent içerisinde yer alan bash komutu çalıştırıldığında "/etc/passwd" dosyasının içeriği HTTP Response başlığında saldırgana "NS" olarak döndürülecektir.

Güvenli Olmayan Doğrudan Nesne Referansı (IDOR) Saldırılarını Tespit Etme

İDOR nedir?

Güvensiz **Doğrudan** Nesne **Referansı (IDOR),**
bir yetkilendirme mekanizmasının olmaması veya doğru kullanılmamasından kaynaklanan bir güvenlik açığıdır . Bir kişinin bir başkasına ait olan bir nesneye erişmesini sağlar.

Insecure Direct Object Reference

2021 OWASP'da yayınlanan en yüksek web uygulaması güvenlik açığı güvenlik riskleri arasında IDOR veya “Broken Access Control” ilk sırada yer alıyor.

IDOR Nasıl Çalışır?

IDOR, diğer web uygulaması tabanlı güvenlik açıkları gibi sağlıksız koşullardan kaynaklanan bir güvenlik açığı değildir. Saldırgan, web uygulamasına gönderilen parametreleri manipüle eder, kendisine ait olmayan ve içeriğini okuyabilen, değiştirebilen veya silebilen bir nesneye erişim sağlar.

İşte IDOR güvenlik açığından nasıl yararlanıldığını daha iyi anlamak için bir örnek.

Basit bir web uygulaması düşünelim. Kullanıcıdan “ **id”** değişkenini alır, ardından istekte bulunan kullanıcıya ait verileri görüntüler.

URL: **https://letsdefend.io/get_user_information?id=1**

Web uygulamamızda yukarıdaki gibi bir istek yapıldığında, id değeri 1 olan kullanıcının bilgilerini görüntüler.

İsteği yapan kullanıcı bensem ve id değerim 1 ise her şey normal şekilde çalışacaktır. Talepte bulunduğumda kişisel bilgilerimi göreceğim.

Ancak “id” parametresi olarak 2 ile bir istek yaparsak ne olur? veya 3?

Web uygulaması şunları kontrol etmiyorsa: “Talepteki “id” değeri, talebi yapan kişiye mi ait?” o zaman herkes bu isteği yapabilir ve kişisel bilgilerimi görebilir. Bu web güvenlik açığına IDOR denir.

Saldırganlar “id” gibi parametreleri değiştirerek kendilerine ait olmayan nesnelere ulaşabilirler. Ne tür bilgilere erişebilecekleri web uygulamasına

göre değişebilir ama her iki durumda da kimsenin kişisel bilgilerinize erişmesini istemezsiniz değil mi?

Saldırganlar IDOR Saldırılarından Nasıl Yararlanır?

Bir saldırganın yapabilecekleri, bir IDOR güvenlik açığı alanıyla sınırlıdır. Ancak en yaygın olarak görüldükleri alanlar, genellikle bir kullanıcının bilgilerinin alındığı sayfalardır. Bir saldırgan bir IDOR güvenlik açığından yararlanırsa şunları yapabilir:

- Kişisel bilgileri çalmak
- Yetkisiz belgelere erişin
- Yetkisiz işlemleri yürütmek (Örneğin: silme, değiştirme)

IDOR Nasıl Önlenir

IDOR zafiyeti olmadan güvenli bir ortam oluşturmak için her zaman talepte bulunan kişinin herhangi bir yetkisi olup olmadığını kontrol etmelisiniz.

Bunun üzerine gereksiz parametreler kaldırılmalı ve kullanıcıdan en az miktarda parametre alınmalıdır. Bir önceki örneği düşünürsek “id” parametresini almamıza gerek yok. Kullanıcıdan “id” parametresini almak yerine, oturum bilgisini kullanarak talepte bulunan kişiyi belirleyebiliriz.

IDOR Saldırılarını Tespit Etme

IDOR saldırılarını tespit etmek diğer saldırılara göre daha zordur. Çünkü SQL Injection ve XSS gibi belirli payload'ları yoktur.

HTTP Yanıtının elinizin altında olması, IDOR saldırılarını tanımlamaya yardımcı olur. Ancak HTTP Yanıtları çeşitli nedenlerle günlüğe kaydedilmez ve bu nedenle IDOR saldırılarını tespit etmek daha zordur.

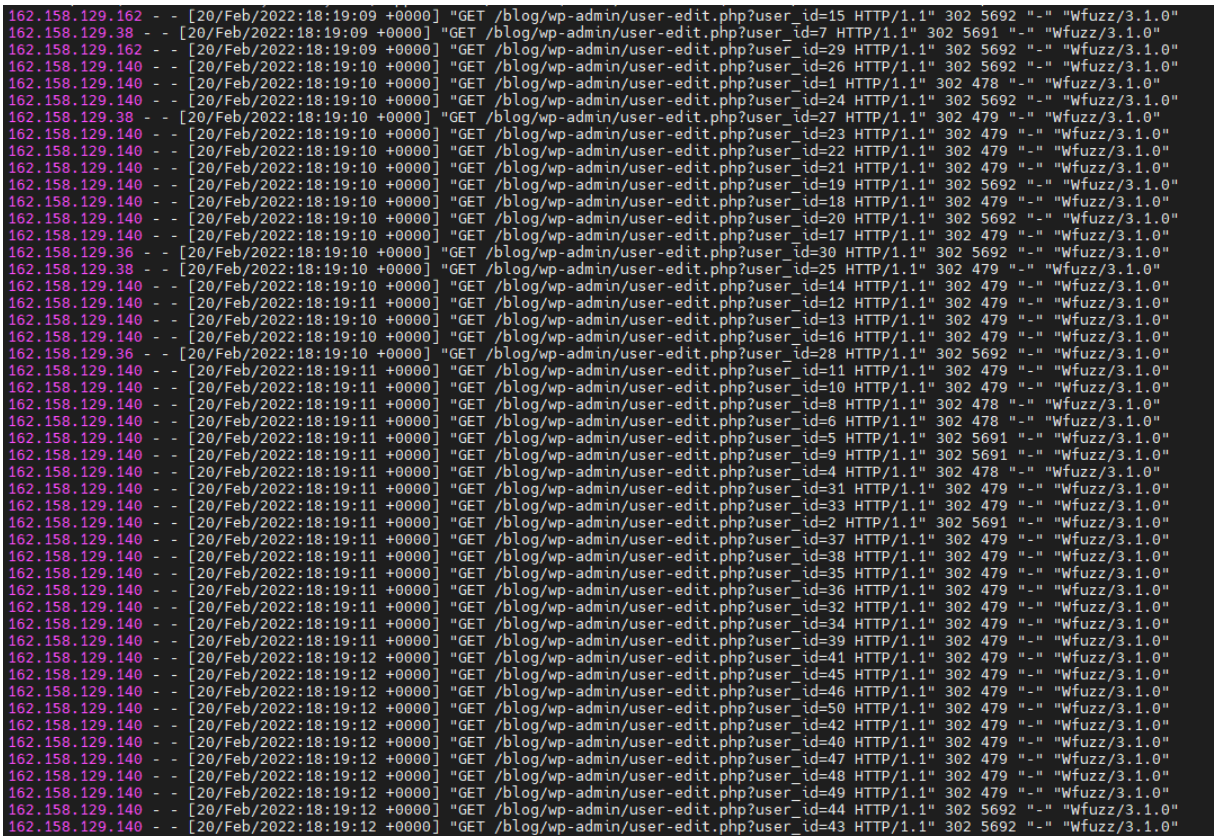
IDOR saldırılarını belirlemede kullanılan birkaç yöntem vardır. Bunlar:

- **Tüm parametreleri kontrol edin:** herhangi bir parametrede bir IDOR güvenlik açığı oluşabilir. Bu nedenle tüm parametreleri kontrol etmeyi unutmamalısınız.
-
- **Aynı sayfa için yapılan isteklerin miktarına bakın:** Saldırganlar bir IDOR güvenlik açığı tespit ettiğinde, diğer tüm kullanıcılarla ilgili bilgilere de erişmek isterler, bu nedenle genellikle kaba kuvvet saldırısı gerçekleştirirler. Bu nedenle aynı sayfa için tek bir kaynaktan çok sayıda istek yapıldığını görebilirsiniz.

- **Bir model bulmaya çalışın:** Saldırganlar, tüm nesnelere ulaşmak için bir kaba kuvvet saldırısı planlayacaktır. Tamsayı değerler gibi ardışık ve öngörülebilir değerlere saldırı yapacakları için bu isteklerde bir kalıp bulmaya çalışabilirsiniz. Örneğin: id=1, id=2, id=3 gibi istekler görürseniz bir şeyden şüphelenebilirsiniz.

Algılama Örneği

Aşağıda, Wordpress çalıştıran bir web sunucusunda bulunan günlüklerin ekran görüntüsünü görebilirsiniz.



Diğer örneklerimizde olduğu gibi, genel, geniş tabanlı bir inceleme ile başlayalım. Yapılan isteklerde özel karakterler olmadığı için logları rahatlıkla okuyabiliyoruz.

Wordpress uygulamasını daha önce kullandıysanız, “wp-admin/user-edit.php?user_id=" sayfasının kayıtlı Wordpress kullanıcıları hakkında bilgiler içerdiğini biliyor olabilirsiniz. Bu sayfaya erişebilmek normal karşılanabilir, aslında birden fazla kullanıcınız varsa birden fazla “user_id: parametresi ile erişim sağlıyor olabilirsiniz. Ancak bu kadar farklı "user_id" parametresine sahip olmak normal değildir.

Elimizde bir IDOR saldırısı var gibi görünüyor.

Kaynak IP'nin ne olduğuna baktığımızda Cloudflare'a ait olduğunu görüyoruz. Bu, erişim günlüğünü aldığımız web uygulamasının bir Cloudflare hizmeti kullandığı anlamına gelir. Bu nedenle istekler Cloudflare üzerinden web uygulamasına iletildi.

Erişim loglarının kaydedildiği kısa zaman dilimi içerisinde 15-16 istek görüyoruz ve bu bize saldırının otomatik bir cihazla yapıldığını

gösteriyor. User-Agent başlığına bakarsak “wfuzz/3.1.0” yazdığını görebiliriz. Wfuzz, saldırganlar tarafından sıklıkla kullanılan bir cihazdır. Bu saldırının sadece otomatik bir tarayıcı aracı tarafından gerçekleştirildiğini tespit etmedik, ayrıca Wfuzz adlı bir araç tarafından gerçekleştirildiğini de belirledik.

Ama hala en önemli soruyu cevaplamış değiliz. Saldırı başarılı oldu mu?

Saldırgan, kullanıcıların bilgilerine erişebildi mi?

HTTP Yanıtlarımız olsaydı işimiz daha kolay olurdu. HTTP Yanıtlarımız olmadığı için Erişim Günlüklerindeki yanıt boyutuna bakalım ve bir çıkarım yapalım.

Daha önce de belirttiğimiz gibi, istenen sayfa kullanıcı bilgilerini gösteriyordu. Kullanıcı adları, soyadları ve kullanıcı adlarının toplam boyutu gibi bilgiler aynı olmayacaktır. Bu nedenle, yanıt boyutu 479 bayt olan istekleri yok sayabiliriz.

Yanıt boyutu 5691 ve 5692 olan isteklere bakarsak yanıt kodunun 302 (redirect) olacağını görüyoruz. Başarılı web istekleri genellikle 200 cevap kodu ile cevaplanacaktır. Dolayısıyla saldırının başarılı olmadığını söyleyebiliriz. Ancak bu bilgi tek başına saldırının başarısız olduğunu belirlemek için yeterli olmayabilir.

Yanıt boyutu 5692 olan 10 istek ve yanıt boyutu 5691 olan 4 istek var.

Daha önce de belirttiğimiz gibi, kullanıcı adı, soyadı, kullanıcı adı gibi tüm bilgilerin toplamının eşit olma olasılığı çok düşüktür. Bu, saldırının başarılı olmama olasılığını güçlendiriyor.

RFI ve LFI Saldırılarını Tespit Etme

Yerel Dosya Ekleme (LFI) nedir?

Yerel Dosya Dahil Etme (LFI), bir dosyanın kullanıcıdan alınan veriler sterilize edilmeden dahil edilmesi durumunda oluşan güvenlik açığıdır. RFI'den farklıdır çünkü eklenmesi amaçlanan dosya, web uygulamasının barındırıldığı web sunucusundadır.

Saldırganlar web sunucusundaki hassas dosyaları okuyabilir, sunucuya uzaktan erişmelerini sağlayacak şifreleri içeren dosyaları görebilirler.

Uzaktan Dosya Ekleme (RFI) nedir?

Remote File Inclusion (RFI), bir dosyanın kullanıcıdan elde edilen veriler temizlenmeden dahil edildiğinde oluşan güvenlik açığıdır. Dahil edilmesi amaçlanan dosyanın farklı bir sunucuda barındırılması nedeniyle LFI'den farklıdır.

Saldırganlar, hazırladıkları sunucuda kötü amaçlı kodlar barındırır ve uzaktaki sunucu üzerinden kurban web sitesini davet ederek çalıştırmaya çalışırlar.

LFI ve RFI Nasıl Çalışır?

Çoğu web uygulaması tabanlı güvenlik açığı gibi, LFI ve RFI'de de bir kullanıcıdan alınan verilerin sterilize edilmemesinden kaynaklanan güvenlik açıkları vardır.

SQL Enjeksiyon güvenlik açıkları, bir kullanıcıdan alınan veriler SQL sorgularına girildiğinde oluşur; Komut Enjeksiyonu güvenlik açıkları, bir kullanıcıdan alınan veriler doğrudan sistem kabuğunda yürütüldüğünde meydana gelir; IDOR güvenlik açıkları, bir kullanıcıdan alınan veriler nesnelere doğrudan erişmek için kullanıldığında ortaya çıkar. RFI ve LFI güvenlik açıkları, bir kullanıcıdan alınan verilerin doğrudan sistemde kullanılmasından veya uzak bir sunucuya bir dosya eklenmesinden kaynaklanır.

Bir kullanıcıdan alınan veriler neden bir dosya eklemek için kullanılır? Web uygulamaları oldukça karmaşık hale geldi ve maalesef geliştirilen her özellik kötü amaçlar için kullanılıyor. Web uygulamalarında bulunan dil seçeneği, bir kullanıcıdan alınan verilere dayalı dosyaları dahil etmek için kullanılır.

```php
<?php
    $language = $_GET["language"];
    include("website/$language/home.php");
?>
```

Yukarıdaki görseldeki kod parçasını incelersek kullanıcıdan alınan “dil” parametresi kullanılarak istenilen web sitesi dilinin seçildiğini görürüz.

Normal bir durumda web uygulaması planlandığı gibi çalışacaktır. Örneğin, "dil" parametresi olarak "en" girilirse, aşağıda görülen dosyayı alırız.

“web sitesi/ **tr** /home.php”

Ancak bir saldırgan aşağıda görülen yükü “language” parametresine girerse, ne yazık ki web uygulaması “/etc/passwd” dosyasını kullanıcıya gösterecektir.

Yük: **/../../../../../../../../../etc/passwd%00**

“web sitesi/ **/../../../../../../../../../etc/passwd%00** /home.php

“../” üst dizine gitmek için kullanılır. Saldırgan web uygulamasının hangi dizinde olduğunu bilmediği için “../” kullanarak “kök” dizine ulaşmaya çalışır. Daha sonra “/etc/passwd” dosyasını isimlendirir ve dosyanın web uygulamasına dahil edilmesini sağlar. Dizeyi sonlandırmak için “%0” kullanılır. Bu şekilde, kalan “/home.php” dizisi web uygulaması tarafından okunmaz.

Saldırganlar RFI ve LFI ile Nasıl Fayda Sağlar?

- Kod yürütme
- Hassas bilgi ifşası
- hizmet reddi

LFI ve RFI Nasıl Önlenir

RFI ve LFI saldırılarını önlemenin en etkili yolu, bir kullanıcıdan alınan verileri kullanmadan önce sterilize etmektir. İstemci tabanlı kontrollerin kolayca atlandığını unutmayın. Bu nedenle kontrollerinizi her zaman hem istemci tarafında hem de sunucu tarafında yapmalısınız.

LFI ve RFI Saldırılarını Tespit Etme

Saldırganların RFI ve LFI saldırıları ile neler başarabileceklerinden daha önce bahsetmiştik. Bir şirket bu tür güvenlik açıklarından yararlanma nedeniyle büyük kayıplar yaşayabileceğinden, bu tür saldırıları tespit edip önlem alabilmeliyiz.

LFI ve RFI saldırılarını nasıl tespit edebilir ve önleyebiliriz?

- **Bir kullanıcıdan gelen bir web isteğini incelerken tüm alanları inceleyin.**
- **Herhangi bir özel karakter olup olmadığını kontrol edin:** Kullanıcılardan alınan verilerde özellikle '/', `.`, `\` gibi notasyonları arayın.
- **LFI saldırılarında sıklıkla kullanılan dosyalar hakkında bilgi edinin:** Bir LFI saldırısında, saldırgan sunucudaki dosyaları okur. Sunucudaki kritik dosya adlarına aşina olursanız, LFI saldırılarını daha kolay tespit edebilirsiniz.
- **HTTP ve HTTPS gibi kısaltmaları arayın:** RFI saldırılarında saldırgan dosyayı kendi cihazına dahil eder ve dosyanın yürütülmesini sağlar.
- Saldırganlar bir dosya eklemek için genellikle kendi cihazlarına küçük bir web sunucusu kurar ve dosyayı bir HTTP protokolü üzerinden görüntüler. Bu nedenle RFI saldırılarını daha kolay tespit edebilmek için “http” ve “https” gibi notasyonları aratmalısınız.