SQL INJECTION

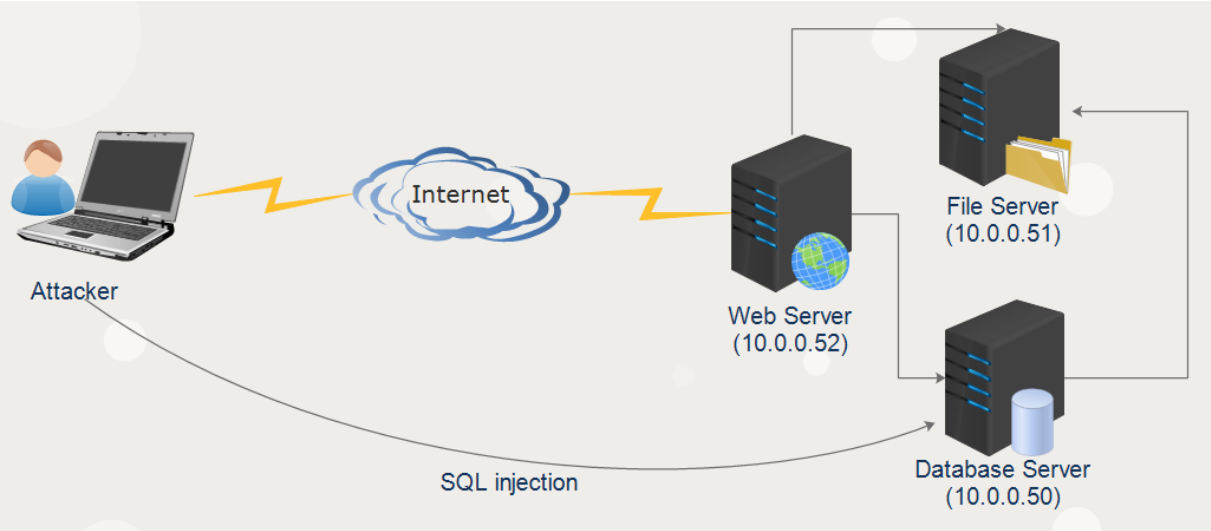
Mehmet TUNCER

https://www.linkedin.com/in/mehmetuncer/

İÇİNDEKİLER

1 SQL Injection Saldırılarına Giriş

SQL Injection saldırısı, web uygulamalarının SQL sorgularına SQL komutları enjekte edilmesi olarak tanımlanabilir. Başarılı bir SQL injection saldırısı kötü niyetli bir programcının bir web uygulamasının veri tabanına ulaşıp onu yönetebilmesine imkan verir.



Karmaşık web uygulamaları genellikle bilgileri, kullanıcı istatistiklerini ve kimlik bilgilerini depolamak için veri tabanı kullanır. CMS'ler basit kişisel web sayfaları ile birlikte MySQL, SQL Server, Oracle, PostgreSQL ve diğer veri tabanlarına bağlanabilir.

Sistem operatörleri, programcılar, uygulamalar ve web uygulamaları veri tabanlarıyla etkileşim kurmak için SQL kullanır.

1.1 SQL temelleri

Bir saldırıyı nasıl yapacağımızı öğrenmeden önce bilmemiz gereken bazı SQL temelleri aşağıdadır:

- SQL deyimleri sözdizimi
- Bir sorgu nasıl yapılır
- İki sorgunun sonuçları nasıl birleştirilir?
- DINSTINCT ve ALL operators
- Yorumlar nasıl çalışır?

Bir SQL sorgusu aşağıdaki gibi gözükmektedir;

```
Select  username, control FROM users WHERE id=1;
```

Yukarıdaki sorgumuz "users" tablosundaki "id" değeri 1 olan üyenin username ve control değerlerini çekmemize yardımcı olmaktadır.

UNION iki veya daha fazla sorgunun çıktısını tek seferde görmek için kullanılır.

```
Select  username, control FROM users UNION SELECT  log_name, log_date FROM userlog
```

Bir tabloda birden fazla tekrar eden kayıtları tek sonuç da görmek için kullanılır.

```
SELECT DISTINCT city FROM users
```

SQL komutlarında yazılan sorguya ait sorgunun istediğimiz yerine hatırlatıcı yazılar yazmak için kullanılır.

```
Select  username  FROM users;  #  this is a comment
Select  username  FROM users;  --  this is a comment
Select  username  FROM users;   /* this is  a comment */
```

Dokümanda ki örneklemeler aşağıda bulunan örnek tablolar üzerinden yapılacaktır.

Users			
id	username	password	control
1	Mehmet	pass123	1
2	Smith	contol123	0
3	Root	R00T	1

Userlog			
id	usersid	log_name	log_date
1	1	Login Successful	1.05.2018
2	3	Login incorrect	1.05.2018
3	1	Logout Successful	1.05.2018

Aşağıda bulunan iki sorgu aynı sonucu sağlamaktadır.

```
SELECT username  FROM users  WHERE id =’1’;
```

```
SELECT username  FROM users  WHERE username =’Mehmet’;
```

Yukarıdaki yazılmış olan SQL sorguların çıktısı “ Mehmet” olacaktır.

Bu, iki SELECT ifadesi arasındaki bir UNION örneğidir:

```
SELECT username , control FROM users  WHERE id =’1’ UNION SELECT log_name,
log_date FROM userlog;
```

Yukarıdaki yazılmış olan SQL sorguların çıktısı;

Sonuç	
Mehmet	1
Login Successful	1.05.2018
Login incorrect	1.05.2018
Logout Successful	1.05.2018

1.2 Web Uygulaması İçinde SQL Sorguları

Bir önceki örnek, veri tabanını direk belli bir konsoldan incelerken SQL sorgusunun nasıl kullanıldığını göstermektedir. Aynı işlemi web uygulaması içinde yapabilmek için, web uygulamasının:

- Veri tabanına bağlanması,
- Veri tabanını sorgulaması,
- Sonuçları alıp, daha sonra uygulamanın amacına göre kullanması gerekir.

Aşağıda bulunan kod PHP ile veri tabanına bağlanma ve SQL sorgusu çalıştırma örneğidir.

```
<?php
$host = “10.0.0.50”;
$user = “root”;
$pass = “toor”;
$database = “example”;

$mysqli = new mysqli($host,$user,$pass,$database);
$mysqli->select_db($database);

$Query=$mysqli->query(“Select username, control FROM users WHERE id=1  UNION
SELECT log_name, log_date FROM userlog”);
$result = $Query->fetch_object();

echo $result;
?>
```

new mysqli (); -> Belirttiğimiz veritabanına bağlantı sağlamak için kullanılmaktadır.
select_db(); -> Veritabanı sorguları için varsayılan veritabanını seçer.
query(); -> Belirttiğimiz veritabanında sorğu yapmamızı sağlamaktadır.
fetch_object(); -> Bir sonuç kümesinin geçerli sırasını bir nesne olarak döndürür.

1.3 Hassas Dinamik Sorgular (Vulnerable Dynamic Queries)

Çoğu zaman sorgular statik olmasada, kullanıcı girdileri tarafından dinamik olarak oluşturulurlar. Burada Hassas Dinamik Sorgu (Vulnerable Dynamic Queries) örnekleri bulabilirsiniz:

```
<?php
$host = “10.0.0.50”;
$user = “root”;
$pass = “toor”;
$database = “example”;

$id=$_GET[‘id’];

$mysqli = new mysqli($host,$user,$pass,$database);
$mysqli->select_db($database);

$Query=$mysqli->query(“Select username, control FROM users WHERE id=’$id’;”);
$result = $Query->fetch_object();

echo $result;
?>
```

Önceki örnek, bir sorgu oluşturmak için kullanıcı tarafından sağlanan girdiyi kullanan bazı kodları göstermektedir (GET request öğesinin id parametresi). Kod daha sonra sorguyu veri tabanına gönderir. Bu davranış çok tehlikelidir, çünkü kötü niyetli bir kullanıcı veri tabanı etkileşiminin kontrolünü ele almak için sorgu yapısını kullanabilir.

Nasıl olduğunu görelim!

Dinamik sorgu:

```
SELECT username, control FROM users WHERE id=’$id’;
```

Aşağıdaki gibi $id değerlerini bekler;

- **1→** `SELECT username, control FROM users WHERE id='1';`
- **Örnek →** `SELECT username, control FROM users WHERE id ='Example';`
- **İdd3 →** `SELECT username, control FROM users WHERE id ='idd3';`

Ancak, bir saldırgan aslında gerçekte bir $id değeri elde ederse sorguyu değiştir.

Aşağıdaki gibi:

```
‘OR ‘a’=’a
```

Sonra sorgu şöyle olur ;

```
SELECT username, control FROM users WHERE id=’’ OR ‘a’=’a’;;
```

Bu, veri tabanının iki koşulu kontrol ederek öğeleri seçmesini söyler:
• id boş olmalı (id='')
• OR her zaman doğru bir koşul ('a'='a')
İlk koşul karşılanmazken, SQL motoru OR'ın ikinci durumunu dikkate alacaktır. Bu ikinci koşul her zaman gerçek bir koşul olarak üretilmiştir. Başka bir deyişle, bu veri tabanı users tablosundaki tüm öğeleri seçmek için söyler!

Veri tabanı yönetimi sistemleri ile ilgili birkaç derin bilgiyle, saldırgan bütün veri tabanına web uygulaması kullanarak ulaşabilir.

1.4 SQL injection Ne Kadar Tehlikelidir?

SQL Injection güvenlik açıklarının “bulma ve kullanma sürecine” daha derinlemesine girmeden önce, bu güvenlik açıklarının başarılı bir şekilde kullanıldıklarında nereye yönlendirebileceklerini anlamanız gerekir.

İlk olarak, web uygulamasının kullandığı DBMS'ye (MySQL, SQL Server ...) göre, saldırganın veritabanının yalnızca manipülasyonundan çok daha ileriye giden bir dizi eylem gerçekleştirebileceğini anlamalıyız.

Bir saldırgan dosya sistemini okuyabilir, OS komutlarını çalıştırabilir, kabuk (Shell) kurabilir, uzak ağa erişebilir ve temel olarak tüm altyapıya sahip olabilir.

Ancak bu her zaman geçerli değildir, daha sonra DBMS'nin ne kadar güçlü olduğunu, SQL'in ne kadar ilerlemiş olduğunu ve bir saldırgandan sonra bir saldırganın yeteneklerinin ne kadar büyük olduğunu göreceğiz.

Gizli verileri (kullanıcı kimlik bilgileri, SSN'ler, kredi kartları ve bir şirkete ait hassas bilgilerin ne olduğu, bir şirket veya bir kişi veri tabanında saklanabilir) saklayan bir veri tabanına erişmenin bir web uygulamasına yapılabilecek en tehlikeli saldırı şekli olduğunu unutmayın.

Web uygulamalarını etkileyebilecek tüm güvenlik açıkları arasında SQL enjeksiyonları, en hızlı sonuçları elde etmeleri nedeniyle bilgisayar korsanları tarafından ilk kontrol edilen açıklardır.

1.5 SQLi Saldırı Sınıflandırması

SQLi hakkında çok fazla literatür var ve her biri farklı yönlere dayanan birçok farklı sınıflandırma türü vardır:

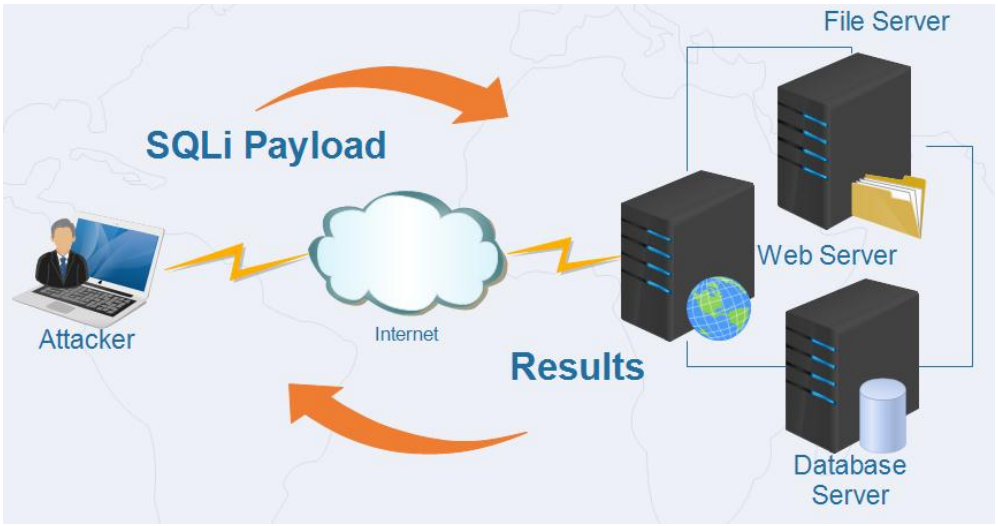
• Saldırının kapsamı,
• Açık bulma vektörü,
• Saldırının kaynağı.

Bu yazıda bu konulara deyineceğiz :
• In-band SQLi,
• Error-based SQLi,
• Blind SQLi.

Bu sınıflandırmalar, saldırıyı gerçekleştirmek için kullanılan açık bulma yöntemine dayanmaktadır. Tespit ve açık bulma aşamalarının açıklamasını daha iyi takip etmenize yardımcı olacağından emin olun. Onu ayrıntılı olarak görelim!

1.5.1 In-Band SQLi

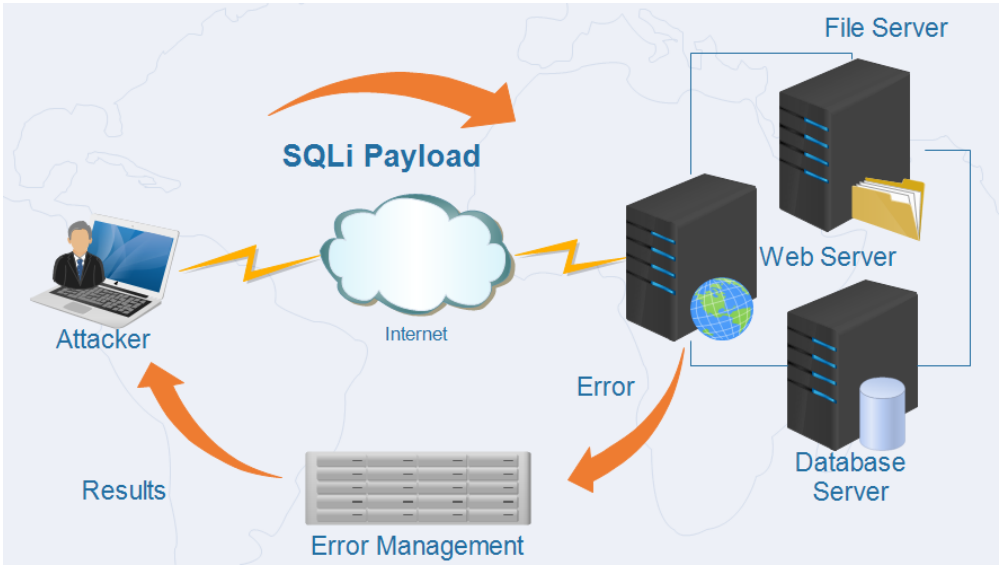
In-band SQLi SQL kodunu (yani, web uygulaması tarafından oluşturulan sayfalar) enjekte etmek için kullanılan aynı kanalı kullanır.



İn-band saldırı esnasında aldırı sırasında, penetrasyon tester, istenen uygulama için web uygulamasından bilgi almanın bir yolunu bulur.

1.5.2 Error-based SQLi

Error-Based SQL enjeksiyon saldırısı sırasında, penetrasyon tester, DBMS'yi bir hata mesajı vermesi için zorlamaya çalışır ve daha sonra veri sızdırılmasını gerçekleştirmek için bu bilgiyi kullanır.



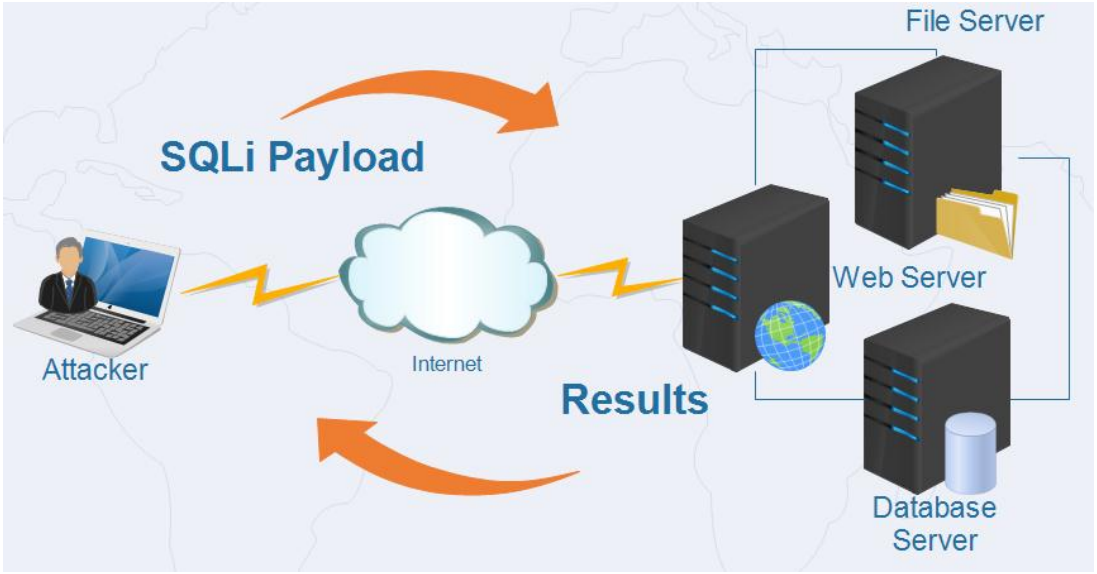
Error-based SQLi enjeksiyondan yararlanmak için, penetrasyon testerin gelişmiş DBMS özelliklerini kullanması gerekir. Hatalar ya web uygulaması çıktısı yoluyla ya da diğer yollarla otomatik raporlar ya da uyarı e-postaları yoluyla gönderilebilir.

1.5.2 Blind SQLi

Blind SQLi enjeksiyonuna karşı savunmasız bir web uygulaması, çıktıdaki enjeksiyonun sonuçlarını yansıtmaz. Bu durumda, penetrasyon tester, güvenlik açığından yararlanmak için bir interference yöntemi bulmalıdır.

İnterference çoğunlukla doğru / yanlış koşullar kullanılarak gerçekleştirilir.

Penetration tester, web uygulama davranışını inceleyerek bir koşulun doğru veya yanlış olup olmadığını anlayabilir.



Bu modülün sonraki bölümlerinde çeşitli SQLi güvenlik açıklarını nasıl algılayacağınızı ve kullanacağımızı göreceğiz.

2 SQLi Bulma

Bir SQL enjeksiyonundan yararlanmak için, önce enjeksiyon noktasının nerede olduğunu bulmanız gerekir, o zaman hedef dinamik sorgunuzun kontrolünü ele geçirmek için bir payload oluşturabilirsiniz.

Bir web uygulamasında SQL enjeksiyonları bulmanın en kolay yolu, SQL sorgusunun sözdizimsel olarak geçersiz olmasına neden olduğu ve bu nedenle web uygulamasını bir hata döndürmesi için zorladığı bilinen karakterlerle girişlerini araştırmaktır.

Not: Bir web uygulamasının tüm girdileri, SQL sorguları oluşturmak için kullanılmaz. Bilgi Toplama modülünde, farklı giriş parametrelerini sınıflandırmanızı ve veri tabanı veri alma ve manipülasyon için

kullanılanları kaydetmenizi tavsiye ederiz.

Aşağıdaki sayfalarda, SQLi açıklarını tanımlamak ve kullanmak için toplanan bilgileri nasıl kullanacağımızı göreceğiz.

Input parametreleri aşağıdakiler aracılığıyla gerçekleştirilir:

GET ve POST istekleri, HEADERS ve COOKIES (Yani, verilerin müşteriden alındığı tüm kanalları kontrol etmeliyiz).

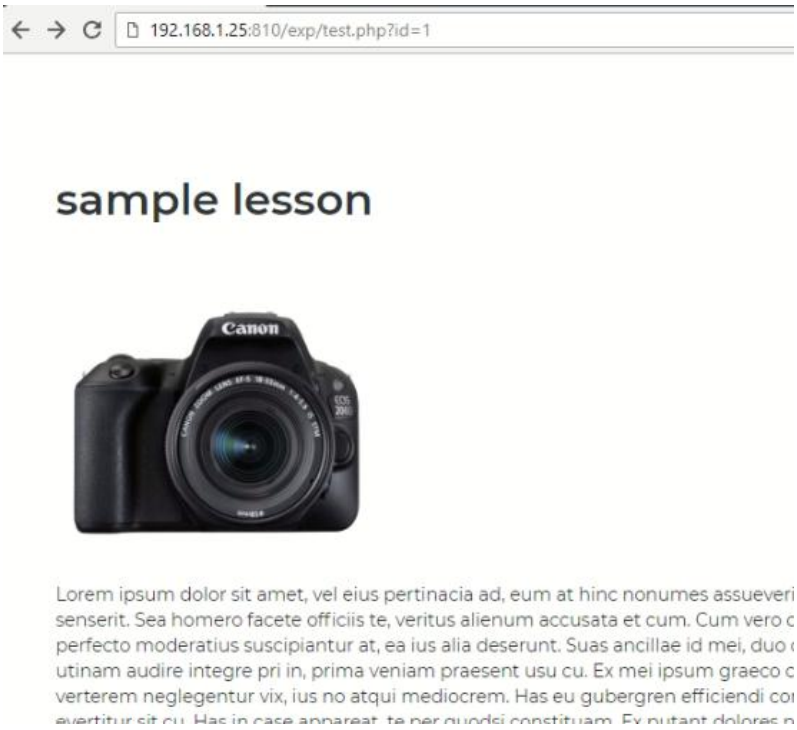
Aşağıdaki örnekler, basit bir şekilde, girdilerin doğrudan URL'den alındığı senaryoları (GET yöntemi ile) inceleyecektir. Aynı teknikler diğer kanallar için de geçerlidir.

2.1 Basit SQLi Senaryoları

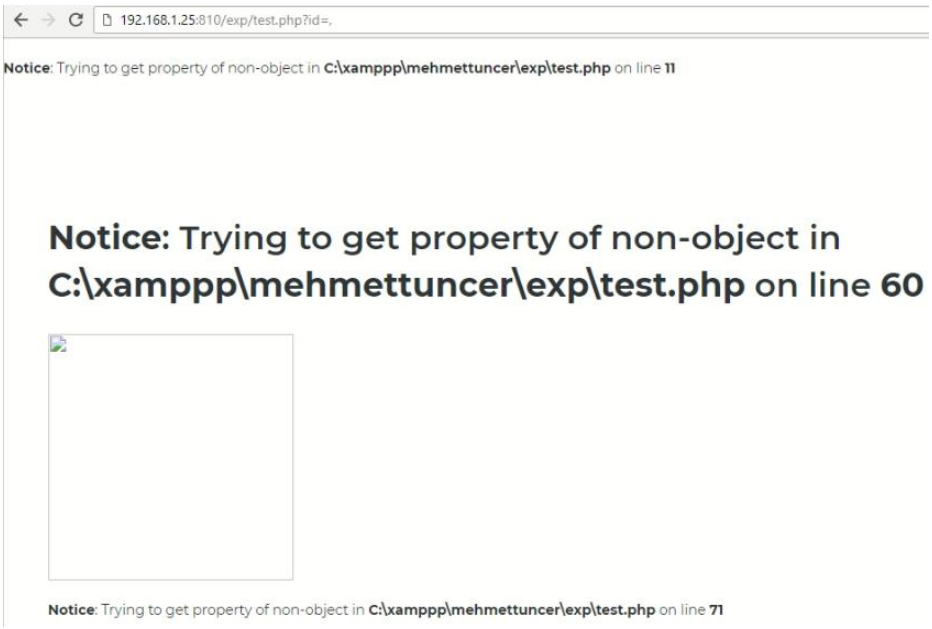
SQL enjeksiyon sürecini açıklamak amacıyla küçük bir savunmasız ürün tanıtım scripti oluşturduk.

test.php, ürün özelliklerini veri tabanından okuyan ve bunları sayfa üzerinde basan bir id parametresidir.

Id parametresinin bir integer olması beklenir. Id = 1 GET parametresini göndermek, uygulamanın doğru davrandığını gösterir.



Ancak bir virgül gönderilirse, uygulama bir hata vermeye zorlanmaktadır.



SQL enjeksiyonu için bir input test etmek, aşağıdakileri enjekte etmeye çalışmak demektir:
• String terminators: ' ve "
• SQL komutları: SELECT, UNION ve diğerleri,
• SQL yorumları: # or --
Web uygulamasının garip davranmaya başlayıp başlamadığını kontrol edin. Her seferinde bir enjeksiyon yapın! Aksi halde hangi enjeksiyon vektörünün başarılı olduğunu anlayamayacaksınız.

2.2 Boolean Tabanlı Tespit

Gizlilik yoluyla güvenlik, güvenliği sağlamak için tasarım, uygulama veya yapılandırma gizliliğinin kullanılmasıdır. Aşağıdaki sayfalarda, bu yaklaşımın SQLi saldırılarına karşı savunmasız bir uygulamayı nasıl savunamadığını göreceksiniz.

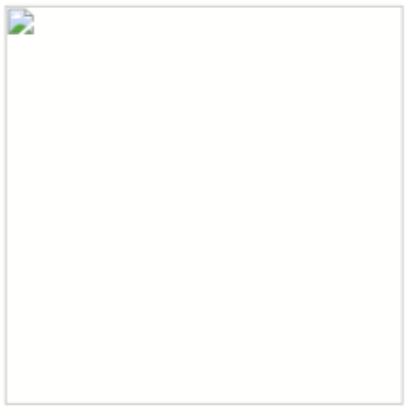
Bir web uygulaması, output hataları göstermiyorsa, Boolen tabanlı tespit tekniği kullanarak SQL enjeksiyonu test etmek hala mümkündür. Bu sürecin ardındaki fikir basit ancak zekice: Web uygulama sorgularını True / False koşullarına dönüştüren faydalı payloadlar oluşturmak. Daha sonra, penetration tester, uygulama davranışlarının farklı Doğru / Yanlış koşullarıyla nasıl değiştiğine bakarak sorgularından sonuçlar çıkarabilir.

2.2.1 Boolean Tabanlı Tespit Örneği

Boolean tabanlı SQL injection tespit etmek için kendi oluşturduğumuz ürün tanıtım script inde ki zafiyeti keşfetip exploit etmeye çalışacağız.

Her zamanki gibi, web sitemizi SQL ile ayrılmış karakterlere göndererek enjeksiyon noktasını tespit etmeye çalışıyoruz. Bu örnekte bir string sonlandırma karakteri kullanılıyor. Web uygulaması herhangi bir görüntü göstermiyor.

Ne yazık ki, uygulama, mevcut olmayan bir görüntüyü istediğimizde aynı şekilde davranır. Örneğin, id = 999999'u GET parametresi olarak iletirsek:



Sayfanın arkasındaki sorguda bir şey olduğundan şüpheleniyoruz.

*SELECT * FROM <Tablo> WHERE id='GETID';*

Yani, sorguyu dönüştürmek için 999999'u veya '1' = '1'i enjekte etmeyi deneyebiliriz:

*SELECT * FROM <Tablo> WHERE id='999999 or '1'='1';;*

Temel olarak her zaman doğru olan bir durumdur!

Web uygulamasında payloadı test etmek bize bir output verir! Her zaman sahte bir durumu da test etmemiz gerekir.

Diğer “always”, “true" ve “always false” koşullarını da test etmek mümkündür:
• 999' and 'exp'='exp
• 999' and 'exp'='expexp
• 999' and 'merhaba'='merhaba
• 999' and 'merhaba'='exit
• exp' or '1'='1
• exp' or '1'='2
Test etmek için ya SQLMap yada manuel olarak yapabilirsiniz.

2.3 In-Band SQLi Sızma

In-Band SQL enjeksiyon teknikleri, UNION SQL komutunun kullanımı sayesinde veri tabanından verilerin çok güçlü bir şekilde alınmasını sağlar. Bu nedenle, In-Band enjeksiyonlar da UNION tabanlı SQL enjeksiyonları olarak bilinir.

Bu tür bir saldırı, bir penetration tester’ın, veri tabanı içeriğini, tablo şemalarını ve gerçek verileri biçiminde, veri tabanı içeriğini çıkarmasını sağlar.

Bu konunun ilk bölümünde gördüğümüz gibi, UNION ifadesi iki veya daha fazla SELECT ifadesinin sonuç kümesini birleştirir.

2.3.1 In-Band Tabanlı Tespit Örneği

Bazı senaryoları inceleyerek In-Band SQL enjeksiyonundan nasıl yararlanacağımızı göreceğiz. İlk senaryoda veri tabanı iki tablo içerir: Kredi Kartları ve Kullanıcılar.

Users			
id	username	password	control
1	Mehmet	pass123	1
2	Root	R00T	1

Card		
usersid	c_num	csv
1	0000 2222 3333 4444	321
2	5555 6666 7777 8888	251

Usersid sütunu, Kullanıcılar tablosunun yabancı anahtarıdır (Foreign Key).

Bu örnekte Mehmet'in kredi kartı numarası 0000 2222 3333 4444, “Root” ise 5555 6666 7777 8888 numaralı kredi kartına sahiptir.

Web uygulaması kullanıcı isimlerini görüntülemek için aşağıdaki kodu kullanır:

```
<?php
$exp=$mysqli->query(“Select username FROM users WHERE id=”.$_GET[‘id’].”;”);
$result = mysql_fetch_assoc(exp);

echo $result[‘username’];
?>
```

Yukarıdaki kodu incelediğimiz zaman SQL injection maruz kalan kod kısmını görmüş oluyoruz.

Artık bir kullanıcı adıyla ilişkili kredi kartını almak için SQLi güvenlik açığından yararlanabiliriz.

Bizim payload’umuz:

```
9999 UNION ALL SELECT c_num FROM card WHERE usersid=1
```

Payload, web uygulamasında sorguyu aşağıdakilere dönüştürür:

```
SELECT username FROM users WHERE id=9999 UNION ALL SELECT c_num FROM card WHERE
usersid=1
```

Id = 9999 olan hiçbir kullanıcı olmadığından web uygulaması, çıkışında ilk kullanıcının c_num'unu gösterecektir!

Artık, tarayıcıyı ya da farklı bir araç kullanarak bir GET isteği göndererek payload’u web uygulamasına gönderebiliriz:

Web uygulaması kullanıcı isimlerini görüntülemek için aşağıdaki kodu kullanır:

ALL operatörün kullanımını not edin. Orijinal web uygulama sorgusunda bir nihai DISTINCT yan tümcesinin etkisini önlemek için kullanılabilir.

SQL Injection güvenlik açığından yararlanırken kullanacağınız bir başka iyi hile de yorumları kullanmaktır. Aşağıdaki gibi:

```
9999 UNION ALL SELECT c_num FROM card WHERE usersid=1; -- -
```

Önceki saldırıda not edilmesi gereken pek çok şey var:

• İkinci SELECT ifadesinin alan türleri, ilk ifadedeki ile eşleşmelidir.
• İkinci SELECT ifadesindeki alanların sayısı, ilk ifadedeki alanların sayısıyla eşleşmelidir.
• Saldırıyı başarılı bir şekilde gerçekleştirmek için, veri tabanının yapısını tablo ve sütun isimlerini bilmeliyiz.

İlk iki sorunu çözmek için, SELECT ifadesinde hangi sütunların kullanıldığını bulmak için gelişmiş bir teknik kullanabiliriz. Sütun sayısını ve türlerini arıyoruz. Veri tabanı yapısını daha sonra nasıl tersine çevireceğimizi göreceğiz

2.4 Error-Based SQLi Sızma

Error-Based SQL enjeksiyonları, veri tabanından veri almanın başka bir yoludur. Doğrudan veri istemedikleri halde, bir hata tetiklemek için bazı gelişmiş DBMS işlevlerini kullanırlar. Hata mesajı, penetration tester'ın hedeflediği bilgileri içermektedir.

Hata mesajının çoğu, web uygulama çıktısına yansıtılır, ancak bir e-posta mesajına gömülebilir veya bir log dosyasına da eklenebilir. Bu web uygulamasının nasıl yapılandırıldığına bağlıdır.

Error-Based SQL enjeksiyonu, bir veri tabanından veri almanın en hızlı yollarından biridir. Oracle, PostgreSQL ve MYSQL Server gibi DBMS'lerde kullanılabilir.

Bazı DBMS'ler hata mesajlarında verilen bilgiler açısından çok cömerttir. Önceki örneklerin bazılarında, olumlu veya olumsuz koşulları eşleştirmek için hatalar kullandık.

Bu kez, veri tabanı isimlerini, şemalarını ve verilerini hatalardan alırız. Bazı MS SQL Server'a özgü payload'ları göreceğiz ve daha sonra diğer DBMS'ler için bazı saldırı vektörlerini tanıtacağız. Temel ilke, herhangi bir DBMS'de aynıdır.

2.4.1 Error-Based Tabanlı Tespit Örneği

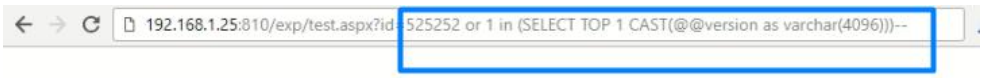
MS SQL Server, hata iletilerindeki veri tabanı nesnelerinin adını gösterir.

Pratikte görelim!

Konuyla ilgili olarak güvenlik sebebiyle sadece CAST tekniğini kullanarak versiyon öğrenme kullanıcı adı ve database adlarını öğrenmeyi göstereceğim.

Oluşturduğumuz test.aspx sayfasında denemelerimizi yapiyoruz.

Test.aspx sayfamızda POST veya GET yaptığımız zaman. Örnek olarak ;



Yukarıdaki örnek payload daki gibi payload ımızı genişletip devam edebiliriz. Örnek olarak veritabanlarını bulabiliriz.

db_name() kullanarak sadece tek veri tabanını görebiliriz fakat db_name(1) db_name(2) olarak arttırma yapıldığında tüm sistemdeki veri tabanlarını görebilirsiniz.

```
52525252 or 1 in (SELECT TOP1 CAST(db_name() as varchar(4096)))–
```

Aynı şekilde sistem üzerinde bulunan kullanıcıları da bulabiliriz.

```
52525252 or 1 in (SELECT TOP1 CAST(user_name() as varchar(4096)))--
```

CAST tekniği özet olarak anlatılmak istenirse bu şekilde yapılmaktadır. Çok fazla kullanılma yönetimi mevcut olup çok derin bir konudur.

2.5 Blind SQLi Sızma

Blind SQLi sızma, veri tabanı şemalarını ve verilerini elde etmek için kullanabileceğiniz bir sızma yöntemidir.

Web uygulaması in-band veya error-based SQL enjeksiyonları yoluyla kullanılamazsa, ancak yine de savunmasız durumda ise, Blind SQLi'ye güvenebilirsiniz.

Bu, blind web enjeksiyonlarının yalnızca web uygulamasının çıktısında hataları yazdırmaması durumunda kullanılabilecekleri anlamına gelmez.

Basitçe, bir Boole tabanlı SQLi payload işlerken, bir sorguyu, durumunu web uygulaması çıktısına yansıtan bir True/False durumunda dönüştürmek istediğiniz anlamına gelir.

Aşağıdaki sayfalarda, hem bir uygulamada hem de hataları çıktı olarak alınmayan farklı bir uygulamada blind SQLi örneğini göreceğiz.

2.5.1 Blind SQLi Tabanlı Tespit Örneği

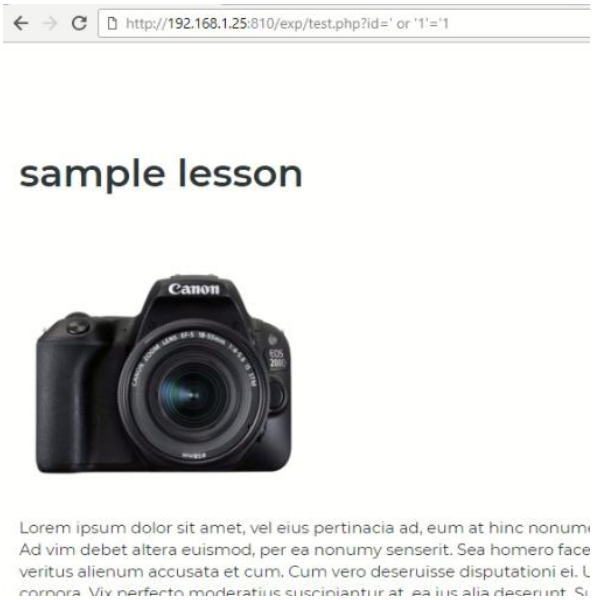
Bu örnekte, id savunmasız bir parametredir.

Dinamik sorgu yapısını tahmin edebiliriz:
SELECT * FROM <Table> WHERE id='<id parameter>';

Sorgu muhtemelen aşağıdaki gibi görünüyor:
SELECT * FROM images WHERE id='<id parameter>';

Bu yüzden her zaman True koşulunu tetiklemeye ve neler olduğunu görmeye çalışabiliriz.

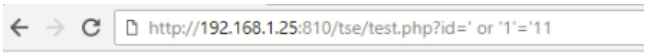
'OR' 1 '=' 1'i kullanabiliriz ve uygulamanın bir resim gösterdiğini görürüz.



Öte yandan, her zaman false koşulu: 'OR' 1 '=' 11

Veri tabanında hiçbir şey bulamıyor: görüntü yok ve görüntü sayacı yok.

Yani bu açıkça bir istismar edilebilir bir SQL enjeksiyon noktasıdır.



Bir kez penetration tester'lar, bir koşulun True ya da False olduğunu söyleyebilmenin bir yolunu bulduklarında, veri tabanına aşağıdaki gibi basit True / False soruları sorabilirler:

• 'a' kullanıcı adının ilk harfi mi?
• Bu veri tabanı üç tablo içeriyor mu?
• Buna basit başka sorular eklenebilir.
Bu yöntemi kullanarak, bir penetration tester veri tabanını serbestçe sorgulayabilir.

Yararlanılan Kaynaklar;

- www.securityidiots.com
- www.pentestmonkey.net
- https://websec.ca/kb/sql_injection
- https://github.com/infoslack/awesome-web-hacking

Online Çalışabileceğiniz Vuln Web Siteleri

- http://testaspnet.vulnweb.com/ - Acunetix ASP.Net
- http://testphp.vulnweb.com/ - Acunetix PHP
- http://crackme.cenzic.com/kelev/view/home.php - Crack Me Bank
- http://zero.webappsecurity.com/ - Zero Bank
- http://demo.testfire.net/ - Altoro Mutual