Prevention of XSS Attack by IP Defending Mechanism

J.Ragavi[*1],R.Sumaya Jabeen*[2],T.Vijayalakshmi*[3] H.Jeya Mohan[#4]
[*123]*U.G Student, B.E CSE, Alpha College of Engg, Chennai, T.N, India.*
[#4]*Assistant Professor, Dept of CSE, Alpha College of Engg, Chennai, T.N, India.*

ragavijayakrishnan@gmail.com

Abstract-— Hacking has become a recent threat faced by many people and there are more script kiddies wanting to steal other's personal information just for fun and also to gain fame and recognition. One can easily hack into others mail or website using various techniques like cross site scripting (XSS), phishing, key loggers, social engineering, and SQL-injection attacks and so on. These attacks make use of vulnerabilities in the code of web applications, resulting in serious consequences such as theft of cookies, passwords and other personal credentials. Cross Site Scripting (XSS) make victims execute an arbitrary script and leak out personal information from victim's computer. An adversary can easily get victims cookie by the XSS attack in the existing system the verification process is weak and do not provide much efficiency in protecting our details of the account. Here we suggest a multilevel security to the users account, using the IP Defending Mechanism which gives you 24*7 protections to your account even when you are offline. When the hacker tries to breech the security, IP Defending Mechanism allows the user to block the intruder.

Index Terms— Cookies, Cross Site Scripting, HTTP, IP Defending Mechanism, Web Application

I. INTRODUCTION

Today the Internet is widely used all over the world. More the Internet is used more the security of computer is demanded. Web Applications have become one of the most important means of communication between various kinds of users and service providers. In World Wide Web, web browsers and web applications communicate to each other through HTTP. The HTTP is a stateless protocol [1] which the web browsers send requests for resources and the web applications supply those resources, no session states are retained. The web applications generally use cookies to provide a mechanism for creating state full HTTP sessions. The cookies are supported by nearly all modern browsers and therefore allow for a greater flexibility in how user sessions are managed by the web applications. For web applications that require authentications, they often use the cookies to store session IDs, and then pass the cookies to user after they have been authenticated. The cookies are stored in the user's web browser. The web browser returns the cookies every time it needs to reconnect as a part of an active session and then the web application associates the cookies with the user. As the cookies can both identify and authenticate the user, this makes the cookies a very interesting target for attackers. In Many cases, the attackers who can obtain valid cookies of the user session can use them to directly enter that session.XSS attack is one of the popular attacks which is often used to steal the cookies using malicious script. The malicious script on executing steals the cookies of the user from a browser's database and sends them to the attacker who can then use them for malicious script. The malicious script on executing steals the cookies of the user from a browser's database and sends them to the attacker who can then use them for malicious purposes. With the cookies of the user in and, the attacker can impersonate the user and then acts instead of that user and interact with the web application .The remainder of this paper is organized as follows. Section II discusses topics which are related to proposed approach: Cookie mechanism, XSS attack and its types, protection of cookies. Section III presents the approach in section IV. Finally, we conclude and also brief the future work in Section V.

BACKGROUND

A.COOKIES

The cookies are a mechanism to provide stateful communication over the HTTP. As mentioned earlier, the cookies are broadly used to store the session IDs or personal sensitive information in today's web applications. The cookies are sent by the web application as a part of a response message using Set-Cookie or Set-Cookie2 header.

The browser stores the cookies in its database, and includes the cookies in its database, and includes with every subsequent request to the web application. The browser uses Cookie header to return the cookies. In general the cookies can be classified into two types: session cookies and persistent cookies [1].

- Session cookies are temporarily used; they are discarded when the browser is closed.
- Persistent cookies can be kept longer until they expire, they are stored on a disk and survive across a computer restarts.

There are two different version of cookie specifications in use [1]: Version 0 cookie (Netscape cookie), and Version 1 cookie (RFC 2965). The version 0 cookie is the most widely used version, it defines the Set-Cookie header, and the Cookie-header as follows:

```
Set-Cookie: name=value[;expires=date][;path=path]
[;domain=domain][;secure]
Cookie: name=value
For example:
Set-Cookie: SID=123abc;domain=.kmitl.ac.th
Cookie: SID=123abc
```

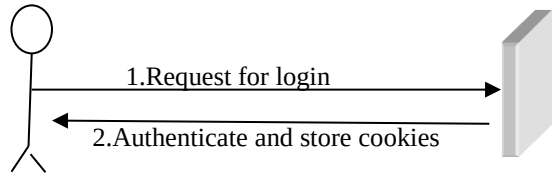
The Version 1 cookie is an extended version of Netscape cookie. In addition to identifying the cookies by name, domain and path attributes as in the Version 0, the Version 1 adds an ability to identify the cookie by the port attribute as well. The web server must set the cookie using the Set-Cookie2 header instead of the Set-Cookie header. The browser still returns the cookie using the Cookie header as the Version 0 but uses a different format [1].The web server must always specify the value of the name attribute and the cookie version in the cookie and the browser must

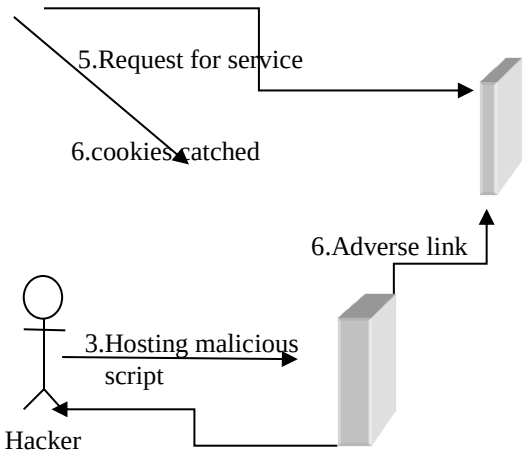
Return the same values. Almost all modern browser do not support the Version 1 cookie except Opera browser, so the Version 1 cookie is not widely used by web developers. The following example shows the Set-Cookie2 header and the Cookie header that are used in the Version 1 cookie:

```
Set-Cookie2:SID=”123abc”;versions=”1”
Cookie:$versions=”1”;
SID=”123abc”
```

B. XSS Attack

Figure 2. Simple Attack Model





7.Acessing legitimate session using cookies

XSS attack is used to hack websites online and it mostly works on those sites which use cookies for storing your username and password when you log in that site.XSS usually works on those sites which allow users to add any code in an open place like starting new thread in forums or send codes using messages to other members. It is actually a script/a code which attacker submit and whoever clicks or even see it got affected

The purpose of the attacker or hacker doing XSS is to steal the cookie of a user, which is currently log in on the site and viewing that code submitted by the hacker so that he can later use that cookie to get into his account. An example of how an attack is implemented is shown in the figure 2.

There is no single standardized classification of cross-site scripting flaws, but most experts distinguish between at least three primary flavour of XSS: non-peristent, persistent and DOM-based attacks.

II TYPES OF XSS ATTACKS:

1. NON-PERSISTENT XSS:

The non-persistent (or reflected) cross site scripting vulnerabilitiy is by far the most common type.It means the malicious code is not persistently stored in a vulnerable server back to a victim.

To consider Figure 3,if the victim is accessing www.bank.com in order to do an online transaction, in the same time the victim may also be accessing www.attacksite.com , and be persuaded into clicking on a below link:

```
<a href=”http://www.bank.com/
<SCRIPT>
Document.location=”http://www.bank.com/
Stealcookie.php?’+document.cookie;
</SCRIPT>”>
Click here to win a million dollars.
</a>
```

Figure 3: Example of Non-Persistent XSS attack

When victim clicks on the link, the malicious script will be sent to the web server cannot find the requested page. Once the web server cannot find the requested page, it will usually return an error page; The web server may also decide to include a name of the requested page in the error page which is actually the malicious script. When the malicious script is executed on the victim's browser, the cookies of the www.bank.com will then be sent to the www.attacksite.com. An owner of the www.attacksite.com can use those cookies to impersonate the victim with respect to the www.bank.com. The malicious script can read the cookies of the www.bank.com without being denied by the same origin policy because it was echoed by the www.bank.com, so it has the same origin as the cookies.

2. Persistent XSS

The persistent(or stored) XSS vulnerability is a more devastating variant of a cross-site scripting flaw: it occurs when the data provided by the attacker is saved by the server , and then permanently displayed on" normal" page returned to other user in the course of regular browsing without proper HTML escaping. A classic example of this is with outline message boards where users are allowed to post HTML formatted messages for other users to read.

Consider a script shown in Figure 4 which it is posted in an online message board of the www.freebooks.com.

```
Click here to get free recharge!!!!!
<SCRIPT>
Document.images[0].src=http://www.attacksite.com/
Images.jpg/stealcookie+document.cookie;
</SCRIPT>
```

Figure 4 Example of persistent attack

The victim who reads a message will receive the malicious script as a part of the message. The victim's browser will then execute the malicious script which will later send the cookies of the www.freebooks.com. Again the malicious script will then execute the malicious script can read the cookies of the www.freebooks.com because it was loaded from the www.freebooks,com which has the origin of the cookies.

3. DOM-based XSS Attack:

DOM Based XSS(or as it is called in some texts,"type-0 XSS") is an XSS attack wherein the attack payload is executed as a result of modifying the DOM "environment" in the victim's browser used by the original client side script, so that the client side code runs in an "unexpected" manner. That is, the page itself does not change, but the client side code contained in the page executes differently due to the malicious modifications that have occurred in the DOM environment. This is in contrast to other XSS attacks, wherein the attack payload is placed in the response page.

Consider the code in the Figure 5, it is used to create a form to let the user choose his/her preferred language. A default language is also provided in the query string, as the parameter "default"

```
…….

Select your language:
<select><script>
Document.write("<OPTION
Value=1>"+document.location.href.substring(document.l
oc
Ation.href.indexOf(" default=")+8)+"<OPTION");
Document.write("<OPTION value=2>English
</OPTION>");
</script></select>
……
```

Figure 5.Example of DOM-based XSS attack

The page is invoked with a URL such as:
http://www.some.site/page.html?default=French
A DOM based XSS attack against the page is accomplished
By sending the following URL to a victim:
http//www.somesite/page.html?default=<script>alert
(document.cookie)</script>
When the victim clicks the above link then the browser sends the request for the cookie to www.somesite. The server responds with the page containing the above JavaScript code. The browser creates a DOM object for the page, in which the document.location object contains the string which is given above. The original JavaScript code in the page does not expect the default parameter to contain HTML markup, and as such it simply echoes it into the page (DOM) at runtime. The browser then renders the resulting page and executes the attacker script as a result the cookie got steeled.

III Protection of Cookies

It is possible to totally disable using the cookies, but it may cause the web servers denying to work without the cookies. So instead of disabling the cookies, in this session we will discuss about common solutions to protect the cookies from the attacks [2].

1. ***IP Mapping:*** The web server maps IP addresses of the users with the cookies and denies any access that come from invalid IP addresses. This helps to mitigate the problem but it does not work where the users access the Internet through the web proxy.

2. Http Only Attribute: Http Only Attribute is a Microsoft extension; it can also be included in cookies before being sent to the browser. With the Http Only Attribute, the browser will deny scripting languages to access those cookies. The Http Only Attribute, the browser will deny scripting languages to access those cookies. The Http Only Attribute will ignore it and will consequently remain vulnerable,

3. Secure Cookies: Secure Cookies means that the client and the web servers only send the cookies via the SSL connections. When using the SSL, all requests and responses are encrypted including the cookies. This can protect the cookies from sniffing whenever they are sent across the network; however this cannot protect the cookies on the browser itself.

No solutions mentioned above can guarantee that the cookies will be safe from the XSS attacks.

We propose a new approach in the next section which aims not to protect the cookies but instead renders the cookie not reusable for the attackers.

Proposed Solution

Cookie Stealing and Session Hijacking has become a ubiquitous threat to the web applications in recent times. To avoid XSS and also various other attacks like phishing, tab nabbing, brute force to name few, we propose an ultimate tool called IP Defending Mechanism. It performs cookie detection and IP tracking mechanism. It is used for continuous monitoring and protection of real authorized users account from the attacker. Here the cookies once created are processed in the Web Server. The unique cookie is stored in one of the secure Web Server as long as that cookie expires. Once the attacker steals the cookie of the victim without his/her through XSS, he then injects the stolen cookie into his browser using any cookie injecting tool or add-on over his web browser. Now once he tries to execute his attack and gain access into the victims account, IP Defending Mechanism automatically detects the change in IP which access the same cookie and thus it expires the cookie that is used by the current user which is ofcourse the attacker. Then it sends an alert to the victims account. And here comes the ultimate use of IP Defending Mechanism. Even attacks other than this technique(cookie stealing through XSS) doesn't work because once the cookie is forced to expire in this fashion, the person trying to access the accounts need to perform the 2-step verification process where the dynamic code is generated and sent to the authorized user account. Thus by using IP Defending Mechanism technique we obtain a much needed security from some of the major hacking techniques that exists.

IV IMPLEMENTATION AND EXPERIMENTAL RESULTS

We created a simple web forum and implemented our proposed IP Defending Mechanism using PHP. We ran our forum on the Windows 7 Ultimate.

We conducted experiments to evaluate a compatibility of our approach on the web browsers like Google Chrome v6, FireFox v3, IE v8, Opera v10.The forum server code is written in PHP v5.3 and ran it on XAMPP. We then used two scenarios to evaluate our approach as shown below.

Scenario 1-evaluated our approach with the version 0 cookies using all five browsers on the following test cases.

- Set-Cookie:SID=abcdefg 1
- Set-Cookie:SID=abcdefg 2;domain=.test0.com Set-Cookie;
- SID=abcdefg 3;domain=.test0.com;path=/lab1
- Set-Cookie:SID=abcdefg4;domain=.test0.com
- Path=/lab2
- Set-Cookie:SID=abcdefg5;domain=.test0.com
- Path=/lab1

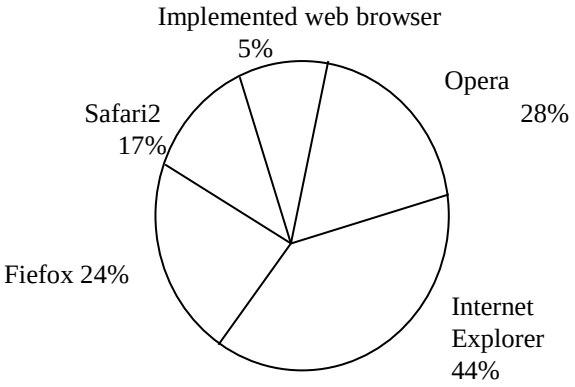
Scenario 2-evaluated our approach with the version 1 cookies using the Opera browser on the following test cases.

- Set-Cookie2:SID=”abcdefg 6”;version=”1”;
- Set-Cookie2:SID=”abcdefg 7”;version=”1”;
- domain=”.test1.com”
- Set-Cookie2:SID=”abcdefg 8”;version=”1”;
- domain=”.test1.com”;path=”/lab1”
- Set-Cookie2:SID=”abcdefg 9”;version=”1”;
- domain=”.test1.com”;path=”/lab2”
- Set-Cookie2:SID=”abcdefg 10”;version=”1”;
- domain=”.test1.com”;path=”/lab1”;port
- Set-Cookie2:SID=”abcdefg 11”;version=”1”;
- domain=”.test1.com”;path=”/lab2”port=” 80, 8000”

- Set-Cookie2:SID=”abcdefg 12”;version=”1”;
- domain=”.test1.com”;path=”/lab1”

V Security Evaluation

The proposed IP Defending Mechanism has been tested with hundreds of malicious inputs, non vulnerable input with white listed tags and vulnerable websites. Figure 6 shows comparison of the proposed browser with Firefox without security implemented, Microsoft’s Internet Explorer, Apple’s Safari Web Browsers on same platform and environment. It has been observed that there are more than hundred variants of XSS attacks exist and the approach is tested with the data collected from various research sites, white hat and black hat sites.



VI PERFORMANCE EVALUATION

It is important that after the application of security by the proposed IP Defending Mechanism model, the user’s web browsing experience is not affected seriously. The proposed browser’s performance was compared with several available browsers. The performance has been observed by logging the time of processing. The approach is tested on 2.0 GHZ Intel Core2duo machine, with 1GB RAM. Each browser’s speed response was logged by putting them through a number of tests. To get a unbiased results, it is important that the internet connection speed should be uniform during the experimentation. The page load time can be calculated by writing a small script on a locally hosted webpage or freely available website load time and speed checker. The IP Defending Mechanism is simulated with existing tools and found to be more efficient and secured.

VII CONCLUSION AND FURTER RESEARCH

Large amount of websites are vulnerable to XSS attacks. The proposed solution is found to be very effective by the experimental results. The solution is platform independent and has been implemented on a platform independent browser, so it can be used with other operating systems With a few changes. Cross Site Scripting vulnerabilities exist on all the platforms, so it is a big advantage over other solutions. It uses a step by step approach instead of performing the entire test at the same time. So if the website is found clean, further tests are performed, thus giving an optimized web browsing experience to the users. The solution is still being tested against attack vectors and vulnerable websites. The solution can be further extended to cover other pernicious vulnerabilities and attacks. It can be implemented as a common solution to be used in web browsers.

REFERENCES

[1] Joon S. Park, Ravi Sandhu, *Secure* Cookies on the Web, 3rd ed. IEEE INTERNET COMPUTING, pp.36-44, JULY - AUGUST 2000.

[2] Vorapranee Khu-smith, Chris Mitchell, Enhancing the Security of Cookies, ICICS 2001, LNCS 288, pp.132-145, 2002.

[3] JNV (Japan Vulnerability Notes) ipedia, CWE-79, Cross Site Scripting, http://jvndb.jvn.jp/ja/ cwe/CWE-79.html.

[4] IPA Security Center, Report on Vulnerability-related Information of Software, http://www.ipa.go.jp/files/000009160.pdf.

[5] Hiromitsu Takagi, Satoshi Sekiguchi, Kazuhito Omaki, A Case Study in How E-commerce Sites Are Vulnerable To the ”Cross-Site Scripting” Attack, IPSJ, Com- puter Security Symposium 2001 (CSS2001), pp.247- 252, 2001.

[6] Hiroki Takahashi, Omar Ismail, Youki Kadobayashi, Suguru Yamaguchi, A Proposal and Implementation of automatic Detection/ Collection System for Cross- Site Scripting Vulnerabilities, IPSJ, IEICE Technical Research Report, Vol.103, No.62, IA2003-6, pp.31-36, 2003.

[7] D. Kristol, L. Montulli, HTTP State Management Mechanism, IETF Documents IETF Tools, http://tools.ietf.org/html/rfc2965.

[8] Rattipong Putthacharoen, Pratheep Bunyatnoparat, Pro- tecting Cookies from Cross Site Script Attacks Using Dynamic Cookies Rewriting Technique, ICACT 2011, ISBN 978-89-5519-155-4, pp.1090-1094, Feb 2011.

[9] Alfred J. Menezes, Paul C. van Oorischot, Scott A. Vanstone, Handbook of Applied Cryptography, CRC Press, ISBN: 0-8493-8523-7, 1997.

[10] Alcorna, W. Cross-site scripting viruses and worms– a new attack vector. Journal of Network Security, 2006(7):7–8, Elsevier, July 2006. [2]