

Protecting Cookies from Cross Site Script Attacks Using Dynamic Cookies Rewriting Technique

Rattipong Putthacharoen*, Pratheep Bunyatneparat *

**Department of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand*
s8060056@kmitl.ac.th, kbprathee@kmitl.ac.th

Abstract— Web applications often use cookies for maintaining an authentication state between users and web applications, these cookies are typically sent to the users by the web applications after the users have been successfully authenticated. Every subsequent request that contains the valid cookies will be automatically allowed by the web applications without any further authentication. The cookies are used to both identify and authenticate the users; therefore they are an interesting target for potential attackers. Cross Site Scripting attack (XSS for short) is one of popular attacks which is often used to steal the cookies from a browser's database. In this paper, we introduce a new technique called "Dynamic Cookies Rewriting", this technique aims to render the cookies useless for XSS attacks. Our technique is implemented in a web proxy where it will automatically rewrite the cookies that are sent back and forth between the users and the web applications. With our technique in place, the cookies at the browser's database now are not valid for the web applications; therefore the XSS attack will not be able to impersonate the users using stolen cookies.

Keywords— Cookies, Cross Site Script Attacks, Web Proxy, HTTP and HTTPS.

I. INTRODUCTION

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 stateful HTTP sessions. The cookies are supported by nearly all modern browsers and therefore allow for a great flexibility in how user sessions are managed by the web applications. For web applications that require authentication, they often use the cookies to store session IDs [2], and then pass the cookies to users 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 users [3], this makes the cookies a very interesting target for attackers. In many cases, the attacker who can obtain valid cookies of the user session can use them to directly enter that session. Cross Site Script (XSS) attack is one of popular attacks which is often used to steal the cookies

using a 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 hand, the attacker can impersonate the user and then acts instead of that user, and interacts with the web application.

The remainder of this paper is organized as follows. Section II discusses topics which are related to a proposed approach: XSS attack types, cookie mechanism, a concept of a web proxy, and basic protections of the cookies. Section III presents the proposed approach. We then evaluate the proposed approach in section IV, and discuss about challenges in section V. Finally, we conclude and also brief the future work in Section VI.

II. BACKGROUND

A. XSS Attack Types

There is no standardized classification of the XSS attacks, obviously there are two well known types of the XSS attacks [4][5]: Non-persistent, and Persistent.

1) *Non-persistent (or reflected) XSS*: means that malicious code is not persistently stored in a vulnerable server, but it is immediately echoed by the vulnerable server back to a victim.

To consider Figure 1, 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.attacksite.com/  
    stealcookie.php?' + document.cookie;  
</SCRIPT>'>  
Click here to win a million dollars.  
</a>
```

Figure 1. Example of Non-persistent XSS attack

When the victim clicks on the link, the malicious script will be sent to the web server (www.bank.com) as a 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 [13] because it was echoed by the www.bank.com, so it has the same origin as the cookies.

2) *Persistent (or stored) XSS*: means that the malicious code is persistently stored in a server's storage, and may later be embedded in an HTML page sent to the victim.

To consider a script shown in Figure 2 which it is posted on an online message board of the www.bank.com.

Click here to see a new promotion.

```
<SCRIPT>
document.images[0].src=http://www.attacksite.com/
images.jpg?stealcookie+ document.cookie;
</SCRIPT>
```

Figure 2. Example of Persistent XSS 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.bank.com to the www.attacksite.com. Again the malicious script can read the cookies of the www.bank.com because it was loaded from the www.bank.com which has the same origin as the cookies.

Some authors have also presented the other advance XSS attacks such as DOM-based XSS [5]-[7] attack, Induced XSS [7] and so on but they are out of scope of this paper.

B. COOKIES

The cookies are a mechanism to provide stateful communications 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 with every subsequent request to the web application. The browser uses Cookie header to return the cookies, as shown in Figure 3.

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
```

In version 0, the cookies are identified by the combination of the following attributes: name, domain, and path. The web server can use an arbitrary string as the value of the name attribute. The domain and path attributes inform the browser that the cookie must be sent back to the server when requesting URL of a given domain and path. If the domain and path attributes are not specified, then they default to the domain and path of the requested object. The expires and secure attributes are possibly not used.

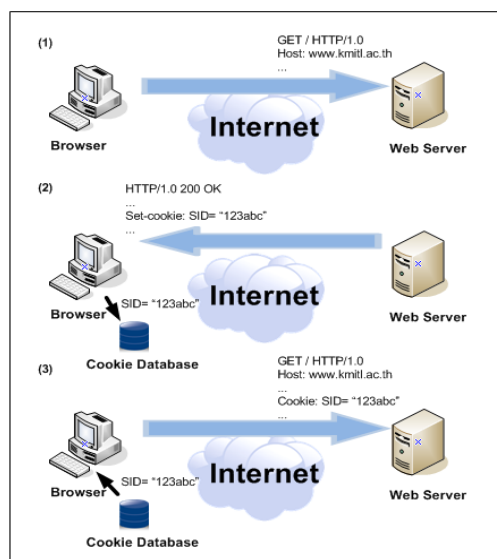


Figure 3. The web server and client exchange the cookies

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 browsers (latest versions at the time of this writing) do not support the version 1 cookie except Opera browser [14][15], 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: $version= "1";
SID= "123abc"
```

The domain and path attributes default to the domain and path of the requested object as in the version 0. The port attribute can be as follows:

- The web server explicitly specifies the port attribute along with its portlist in the cookie, e.g. Set-Cookie2: SID= "123abc"; versions= "1"; port= "80,8000".
- If the web server includes the port attribute but without specifying the portlist, e.g. Set-Cookie2: SID= "123abc"; versions= "1"; port, then the port attribute defaults to the port that the object was requested.
- If the web server does not include the port attribute in the cookie, then it allows "Any" port

In version 1, the web server is possible to include the following attributes in the version 1 cookie as well [1]: comment, commentURL, discard, max-age, and secure.

The browser discards the cookies, if those cookies are under any of the following conditions:

- The browser is closed, if the cookie is not persistent.
- An expiration date has been passed or changed to a date in the past.
- The user forces to delete the cookies.

C. WEB PROXY

The web proxy is an intermediary or a middleman [1] that fulfills transactions on behalf of clients, many organizations allow the users only to access the internet via the web proxy. It is an important control point for web surfing which is commonly built in with various security capabilities. The web proxy can be a separate device or a part of a firewall. It must sit between the clients and the web servers, and acts as both the client to the web server and the web server to the client. All web connections from the clients are intercepted at the web proxy, and then the web proxy will initiate new web connections to the web servers on behalf of the clients, see Figure 4.

Today web proxy can intercept both HTTP and HTTPS connections [8]-[10] so it can have a full control on both protocols. A main difference between the HTTP and the HTTPS is that the HTTPS sends HTTP requests and responses over SSL connections that encrypt all data. In order to see web contents inside the SSL connections, the web proxy must be able to terminate and also initiate the SSL connections [1].

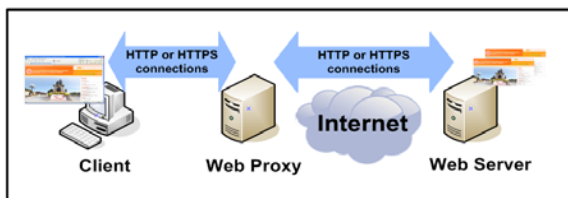


Figure 4. Web Proxy intercepts both HTTP and HTTPS

D. 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 comes 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) *HttpOnly Attribute*: HttpOnly attribute is a Microsoft extension, it can also be included in the cookies before being sent to the browser. With the HttpOnly attribute, the browser will deny scripting languages to access those cookies. The HttpOnly attribute is originally not a part of the HTTP; the browsers that are not aware of this attribute will ignore it and will consequently remain vulnerable.

3) *Secure Cookies*: Secure cookies mean that the clients 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 cookies not reusable for the attackers.

III. OUR APPROACH

In this section, we present the new approach that significantly protects the cookies from the XSS attacks. Our approach can be implemented simply in the web proxy without any change required on both web browser and web server. We propose a new technique called "Dynamic Cookie Rewriting" which is then implemented as a part of our web proxy. With this technique in place, the web proxy will automatically rewrite the value of the name attribute in the cookie with the randomized value before sending the cookie to the browser, so the browser will keep the randomized value in its database instead of the original value sent by the web server. The returned cookie from the browser will also be rewritten back to the original value at the web proxy before being forwarded to the web server. As the browser's database does not store the original values of the cookies, so even the XSS attacks can steal the cookies from the browser's database, the cookies cannot be used later to impersonate the users

Our approach supports both version 0 and version 1 cookies. We design a table for storing the cookies of both versions as follows.

TABLE I. TABLE FOR STORING THE ATTRIBUTE VALUES

Name	Domain	Path	Port	Original Value	Randomized Value
SID	kmitl.ac.th	/	Any	123abc	&^@#\$5

We use four attributes to identify the cookies: name, domain, path and port. The port attribute will only be used with the version 1 cookie. After the web proxy reads the cookies from the response message sent by the web server, it will store all required attribute values into the table as shown in the Table I, and rewrites the values of the name attributes with the randomized values before sending to the browser. The randomized values will also be kept on the same table. The web proxy will only change the name attribute of the cookie, the other attributes will remain the same.

The domain, path or port attributes may not be included in the cookies sent by the web server, if this happens, the following rules will be applied:

- The domain attribute defaults to the same as the requested object, so the web proxy will store the domain of the requested object into the table.
- The path attribute also defaults to the same as the requested object, the web proxy will store the path of the requested object into the table too.
- If the web server includes the port attribute but without specifying the portlist, then the port attribute defaults to the port that the object was requested. Therefore the port that the object was requested will be stored into the table.
- If the web server does not include the port attribute in the cookie, then the web proxy will store “Any” port into the table.
- If it is the version 0 cookie, then the web proxy will store “Any” port into the table.

The web proxy will use the information in the table to rewrite back the values of the name attributes in the cookies (sent by the browsers) back to the original values, as shown in Figure 5.

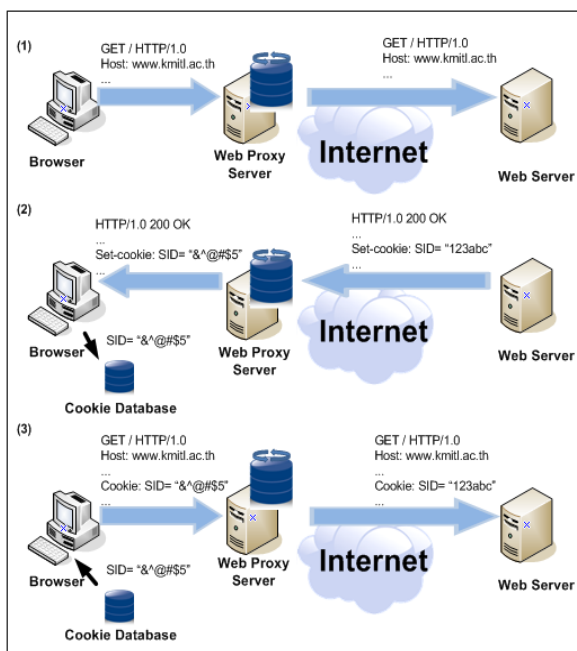


Figure 5. How the Dynamic Rewriting Cookies technique work

If the web proxy receives the request message but without the cookies, and also the domain, path and port of the requesting URL of that request message match an entry in the table, then that entry will be deleted from the table prior to the web proxy forwards that request message to the web server.

And finally if the value of the name attribute in the cookie sent by the web server is changed from that exists in the table, then the web proxy will update the table accordingly.

IV. EVALUATION

We wrote a simple web proxy and implemented our proposed approach using JAVA programming language. We ran our web proxy on the Windows 2008. The current version of our code does not support the HTTPs.

We conducted experiments to evaluate a compatibility of our approach on five popular web browsers: Google Chrome v6, Firefox v3, IE v8, Opera v10 and Safari v4. We wrote the server's code using Perl and ran it on Apache 2. We then used three 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= abcdefg1
- Set-Cookie: SID= abcdefg2 ;domain= .test0.com
- Set-Cookie: SID=abcdefg3;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="abcdefg6";version= "1";
- Set-Cookie2: SID= "abcdefg7";version= "1"; domain= ".test1.com"
- Set-Cookie2: SID= "abcdefg8"; version= "1"; domain= ".test1.com";path= "/lab1"
- Set-Cookie2: SID="abcdefg9"; version= "1"; domain= ".test1.com";path= "/lab2"
- Set-Cookie2: SID= "abcdefg10"; version= "1"; domain= ".test1.com";path= "/lab1";port
- Set-Cookie2: SID= "abcdefg11"; version="1"; domain= ".test1.com";path="/lab2";port= "80,8000"
- Set-Cookie2: SID= "abcdefg12";version= "1"; domain= ".test1.com";path= "/lab1"

Scenario 3 – evaluated our approach with real websites on the internet. Currently we have experimented with 120 websites.

The results showed that our approach worked well with all three scenarios. We saw in the experiments that the browsers stored and returned the randomized values properly and the web server accepted the cookies sent from our web proxy.

V. DISCUSSION

To implement our proposed approach in the web proxy and deploy it in the live environment, we have to consider at least the following three challenges:

- Compatibility - our approach changes the values of the cookies in HTTP header, so we have to make sure that this will not break the HTTP and be able to work well and transparently with all websites on the internet. Although almost all websites on the internet use only the version 0 cookies but our approach follows both version 0 and version 1 cookie specifications in order to be sure that it is able to manage properly all cookies sent by all websites. We conducted the experiments as shown in the scenario 1 and scenario 2 of the previous section to prove that our approach is able to handle both the version 0 and version 1 cookies, and also verify in the scenario 3 that it does not have any side effects on the HTTP protocol and be able to work well and transparently with the real websites on the internet.
- Performance - deploying the web proxy into the network definitely adds more latency and increases the response time. However, the web proxy is usually built in with a caching capability which is able to cache web contents and serve the cached contents locally to the users [11]. So with the web proxy in place, instead of degrading the system performance, it helps to leverage overall performance, improve user experience and also reduce internet bandwidth usage [11]. Hardware platforms have been developing dramatically over the past few years [12]. With the advanced hardware platform itself, the latency on the web proxy will be reduced accordingly.
- Single point of failure - as our web proxy must rewrite the cookies, and keep the original values of the cookies in its database. So when it fails, those original (and valid) values will be gone. Fortunately, there are several technologies in place which are able to detect and bypass the web proxy once it fails such as PAC file, WCCP Protocol, Policy Based Routing (PBR) and so on. With one of those technologies, even the web proxy fails, the clients are still able to reach the web servers. While invalid cookies in hand, the users must be re-authenticated by the web servers. After successful authentication, the authenticated users will be able to access web services as usual without the web proxy involved.

Although deploying the web proxy has some challenges as mentioned above, the web proxy still has huge advantages [8]-[10] such as Web Content Caching, Network Address Translation (NAT), Authentication and authorization, Content filtering, URL filtering, Malware prevention, Data leak protection, Logging/Reporting and so on which organizations

can gain benefits from those advantages along with our approach.

VI. CONCLUSIONS

This paper has presented the Dynamic Rewriting Cookie technique which aims to disarm the attackers from impersonating the users. This technique was implemented as a part of our web proxy and then was evaluated with three scenarios. The results showed that our new technique worked well with both versions 0 and version 1 cookies, and has showed no errors with any real world websites.

Currently we are doing more researches on how to program our web proxy to intercept the HTTPs. Nearly all e-commerce web sites today send the cookies over the SSL connections. Based on our current study and current evaluation, we believe that our technique will also work well with the HTTPs. In order to prove that, our web proxy must be able to intercept the SSL connections which this is a big part of our future work.

ACKNOWLEDGMENT

The authors would like to thank Rene Spallek for providing a review and valuable comments.

REFERENCES

- [1] D. Gourley, B. Totty, M. Sayer, S. Reddy, and A. Aggarwal, *HTTP The Definitive Guide*, 1st ed., O'Reilly Media, US, 2002.
- [2] D. Kristol, "HTTP State Management Mechanism," in *Internet Society*, 2000. Available: <http://www.ietf.org/rfc/rfc2965.txt>
- [3] "Cross Site Scripting Techniques and mitigation," GovCertUK, revision 1.0, October 2009. Available: www.govcertuk.gov.uk
- [4] J. Garcia-Alfaro and G. Navarro-Arribas, "Prevention of Cross-Site Scripting Attacks on Current Web Applications," Available: <http://hacks-galore.org/guille/pubs/is-otm-07.pdf>
- [5] S. Saha, "Consideration Points: Detecting Cross-Site Scripting," (*IJCSIS*) *International Journal of Computer Science and Information Security*, Vol. 4, No. 1 & 2, 2009.
- [6] A. Klein, "DOM Based Cross Site Scripting or XSS of the Third Kind," July 2005. Available: <http://www.webappsec.org/projects/articles/071105.shtml>
- [7] A. Wiegenstein, M. Schumacher, X. Jia, and F. Weidemann "Whitepaper: The Cross Site Scripting Threat," 2007. Available: <http://www.virtualforge.de>
- [8] "White paper: How to Gain Visibility and Control of Encrypted SSL Web Sessions," Available: <http://www.bluecoat.com>
- [9] "Technology Overview: Cisco IronPort Web Usage Controls," Available: <http://www.ironport.com>
- [10] "Solution Brief: McAfee Web Gateway," Available: <http://www.mcafee.com>
- [11] S. Patarin and M. Makpangou, "On-line Measurement of Web Proxy Cache Efficiency," Available : <http://hal.inria.fr/inria-00071804/PDF/RR-4782.pdf>
- [12] "White paper: Platform 2015: Intel® Processor and Platform Evolution for the Next Decade," Available : http://epic.hpi.unipotsdam.de/pub/Home/TrendsAndConceptsII2010/HW_Trends_borkar_2015.pdf
- [13] S. Shah, *WEB 2.0 SECURITY: Defending Ajax, RIA, and SOA*, Charles River Media – Thomson, US, 2008.
- [14] UW Staff Web server [Online]. Available: <http://staff.washington.edu/fmf/2009/06/19/setting-cookies/>
- [15] Wikipedia website [Online]. Available: http://en.wikipedia.org/wiki/Talk%3AHTTP_cookie