Preventing Abuse of Cookies Stolen by XSS

Hiroya Takahashi
Kanazawa University
Kanazawa, Ishikawa, Japan

Kenji Yasunaga
Kanazawa University
Kanazawa, Ishikawa, Japan

Masahiro Mambo
Kanazawa University
Kanazawa, Ishikawa, Japan

Kwangjo Kim
KAIST
Korea

Heung Youl Youm
Soonchunhyang University
Korea

***Abstract*—Cross Site Scripting (XSS) makes victims execute an arbitrary script and leaks out personal information from victims' computers. An adversary can easily get victim's cookies by the XSS attack. If the adversary cannot use the stolen cookies to impersonate the victim, stealing cookie has no meaning. Therefore, we propose a method to prohibit the abuse of stolen cookies in order to make it ineffective to steal cookies through the XXS attack. The proposed method uses one-time password and challenge-response authentication to identify whether a person is a valid owner of the cookie or not.**

***Keywords*-Cookies, Cross Site Scripting, HTTP, 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. Especially protecting personal information is one of major concerns because of the advent of various kinds of Internet services such as SNS and online shopping. Such various kinds of services often use a function called ”cookie”.

Cookie is a small piece of data sent from a website and stored in a user's web browser while a user is browsing the website [1], [2]. It can be read out by web server whenever needed. Cookie provides websites with a reliable mechanism for remembering their state and user's activities. Cross Site Scripting (XSS) is a typical web application vulnerability[3] and an attacker of XSS can steal cookies by setting up the execution of malicious scripts after crossing several websites. This attack finally enables the adversary to do illegal access or session hijack. Nowadays, XSS dominates the largest percentage of all web application vulnerabilities [4]. Unfortunately, there is no effective way to prevent it [5], [6]. A secure cookie system and a method to prevent XSS attack are highly demanded.

In this paper, we propose a secure cookie protocol which prevents the abuse of cookies stolen by XSS. If an adversary cannot abuse the cookies, accounts of the victims remain safe. This paper is organized as follows. After the introduction, we explain background knowledge of HTTP, Cookies and XSS including a fundamental method to invalidate a malicious script contained in the HTTP request in section II. In addition to the fundamental method, previous methods are described in section III. Then we describe a new method to prevent abuse of stolen cookies with enough usability in section IV. After that, we discuss in comparison with previous methods in section V. Conclusion is given in section VI.

II. Background

A. Session Management of HTTP

Hyper Text Transfer Protocol (HTTP) is a protocol which is constructed for exchanging data between a web browser and a web server at World Wide Web (WWW). HTTP is a request-response type protocol such that a client, web browser, sends a server a HTTP request containing URL and method, and receives a HTTP response. The set of request and response is called session. When the session is completed, connection between the client and the server is disconnected. The connection is stateless. That is, the server treats each request as an independent connection: The server does not hold the information of the previous client and cannot reflect the state of the previous session. This kind of stateless protocol is not suitable for online

operations such as online shopping. In order to solve this problem, there is an idea of session management which provides statefull protocol that is able to hold the state of the connection. Session management means that the server recognizes the session with the clients and grasps the progress of processing. If session starts, messages between the server and the client share the same session ID. They recognize each other by checking the session ID [7].

B. Cookie

The technology which enables the session management over the HTTP protocol is called cookie. Cookie is widely used for storing the session ID and personal information handled in web applications. It is a small size of data stored in a text file of the user's computer and exchanged between the server and the client [1], [2]. There are six parameters in the cookie called attribute.

1) Name of the cookie
2) Value of the cookie
3) Deadline of the cookie
4) Path of the server which the browser sends the cookie
5) Domain of the server where the browser sends the cookie
6) The demand for a secure connection between the browser and the server

Cookie is given to a web browser from the server and is held at the browser until it expires. There are two types of cookies, a session cookie and a persistent cookie. The session cookie is used temporarily and discarded when the browser closes. The value of a session cookie is a random value and renewed every time a new session starts. On the other hand, a persistent cookie is stored in the browser for a definite period of time. Once a persistent cookie is given to the browser, it can be reused for any number of times, which improves the performance of web services.

C. Cross Site Scripting (XSS)

An attack of XSS [3] inserts a malicious script while a user is browsing web pages. If the attack succeeds, the adversary makes a victim to execute an arbitrary script which is sent from the server to the browser without any translation. The adversary does not attack a vulnerable server directly, but uses the vulnerability to lead a target user to a phishing page. The figure 1

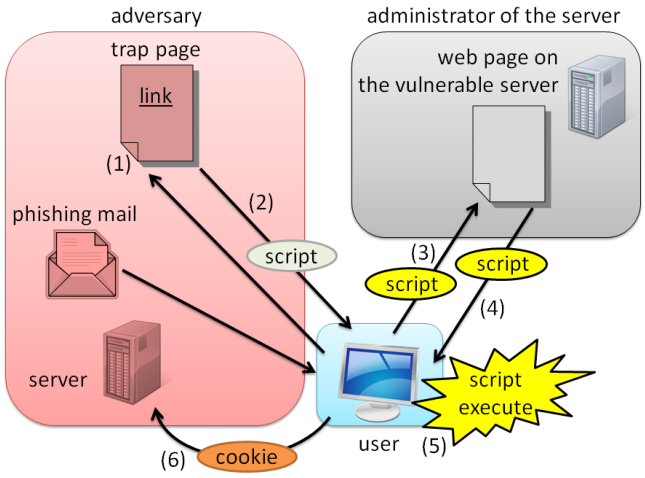

Figure 1. Procedures of XSS

shows the concept and procedures of XSS. An attacker follows the following procedures to launch the XSS attack.

1) An adversary prepares a web site (trap page) and puts a link to other web site which has vulnerability. The link contains a malicious script, that sends cookie to the adversary in step 6).
2) When a user comes to the trap page, he receives the HTTP response which contains malicious script. At this stage, the script has not been invoked.
3) When a user clicks the link, the malicious script is sent to the vulnerable server as a part of HTTP request.
4) The server returns the malicious script to the user's browser without any process because of the vulnerability.
5) The browser executes the malicious script because the browser falsely regards the script is required not from the adversary but from the server.
6) Because of the execution of the malicious script, the browser sends cookie or personal information to the adversary.

If the vulnerable server exists, it enables the adversary to commits various kinds of attack by crossing a trap page and embeds a malicious script. It means that after the success of the XSS attack the adversary can get personal information to impersonate a valid user or

session hijack.

XSS vulnerability often exists in the web page which has a text field and operates dynamically according to the input character. Therefore web pages which deal with the personal information such as resister page or login page are more likely to have XSS vulnerability. The cause of XSS vulnerability is that a web browser executes all scripts received from any server. Of course a browser has option to prohibit the execution of all the scripts that the browser receives. However, it spoils the convenience of cookies. One of the reasons why XSS is still a major issue is low level awareness of people who manage the web server. People who are not an expert of computing can make web pages easily today and get damage of XSS. Meanwhile, server administrators are less likely to take an active action against XSS because they do not get damage directly. Also, taking measures against XSS needs a lot of cost and XSS seems to be ignored in spite of its damages.

A fundamental method to invalidate a malicious script contained in the HTTP request is escaping special characters by using CGI script. In the method, servers regard a script in the HTTP request as a script embedded by an adversary. The CGI script replaces special characters such as ”&” and ”<” of the HTTP request with equivalent characters shch as ”&” and ”<”, respectively, before it is sent to the user's browser as a HTTP response. A malicious script is never executed after escaping special characters. Unfortunately, there are still many sites which do not take this counter measure. From these reasons, XSS is still a major attack in computer networks that we must prevent.

III. Previous Work

A. Using Session Cookie

Cookie is stored at a browser until it expires. If we are able to remove the cookie before the adversary successfully steals it, the leakage of cookie never happens. The valid period of cookie can be determined by setting the deadline of cookie described in section II-B. In the session cookies, the period is set at the past time so that these cookies will be removed every time the session terminates. This method [5] disables the adversary to abuse the stolen cookie, but it ruins the advantage of cookie which does not require the valid user to input his personal information every time he logs in to his account. Also it cannot prevent replay attack which reuse the password stored in the session cookie.

B. Dynamic Cookies Rewriting Technique

Rewriting the value of cookies is effective for precluding the adversary to use stolen cookies [8]. When the server sends a cookie to the browser, a web proxy changes the values of the cookie into randomized values before stored in the browser. As the browser's database does not have the original values of the cookies, the adversary does not get the true values even if the XSS attack succeeds. The proxy has a table which stores the attributes of the cookie and randomized values. When a server requests the browser to send a cookie, the randomized cookie is rewritten to the original value by the proxy and send to the server. A user can reuse the same cookie again. So the user can enjoy the benefit of the cookie. However there are two types of adversaries for which this method does not work; (1) If an adversary and valid users are in the same LAN and the adversary steals the randomized cookies, the adversary succeeds in impersonation by sending these cookies through the same proxy in the LAN. (2) When an adversary steals cookies after the proxy writes them back to the original values, these stolen cookies contain true values. This method is secure only if the adversary cannot use the same web proxy as a valid user uses. Concerning replay attack, both of adversaries described above can perform replay attack easily.

IV. Preventing Abuse of Stolen Cookies

A. Outline

As explained in section II-C, direct anti-XSS methods such as escaping special characters is not secure enough. In this section, we discuss indirect anti-XSS methods. In our approach, we pay attention to decreasing the effect of stealing cookies by XSS. That is, an adversary cannot abuse stolen cookies even if they have leaked out. Stolen cookies are often used for session hijacking and prohibiting abuse of the stolen cookies allows us to prevent session hijacking. When someone accesses to a login page with a stolen cookie, the system regards him as a valid user and impersonation succeeds. From this point of view, distinguishing a regular user from an adversary possessing stolen cookies is effective for invalidating the XSS. In the

next section, we propose a method for indirect anti-XSS using the challenge-response authentication.

B. Synchronized State Cookie Protocol

We propose a method that uses one-time password and challenge-response authentication. In our method, a server and a user have the same password which is renewed every fixed time. The user keeps the password in the persistent cookie and uses it when he needs to log in to his account. The password the user uses and the password the server keeps need to be synchronized at all times. There are various kinds of one-time password schemes [9]. If a new password is generated by an algorithm using a secret value, the value should not be stored in any cookie.

We also use challenge-response authentication in our method. In response to an authentication request, the server sends a challenge value to the user. We use a PHP session ID as a challenge value. A user calculates the response value by hashing the concatenation of his password and the challenge value. Then the user sends the response value to the server, and the server checks whether the value matches with the value calculated by the server or not. When it matches, the user is able to access his account.

V. CONSIDERATION

In this section, we discuss properties of the proposed method. There are some advantages in our method. Even if an adversary succeeds in stealing a password contained cookie by XSS, the adversary cannot abuse the cookie after its expiry. In other words, the adversary cannot impersonate a valid user when a fixed time has passed. However, before the cookie expires, the adversary can succeed in impersonation. From this reason, we have to set an appropriate short interval to renew the password.

Challenge-response authentication avoids the reuse of intercepted values. Even if an adversary gets the response value sent from a valid user to the server, the adversary cannot use it later because a different challenge is sent to the adversary in the succeeding authentication. Our method can prevent the replay attack.

We compare our method with previous work. The table I shows the summary of the comparison. As described in section III-B, the Dynamic Cookies Rewriting Technique is vulnerable to several types of adversaries, and the abuse of cookies is prevented depending on the type of adversaries. Using proxy to rewrite the cookie causes latency of communication between the server and the user.

Table I
COMPARISON OF THREE PROTECTIVE METHODS

	Section III-A Session Cookie	Section III-B Cookie Rewriting Technique	Our Method Synchronized State Cookie
Cookie Abuse Prevention Before Expiry	×	∗	×
Cookie Abuse Prevention After Expiry	✓	∗	✓
Anti-Replay Attack	×	×	✓
Low Latency	✓	×	×
Possibility of No Error	✓	×	×
Usability	×	✓	✓

✓ : satisfied , × : not satisfied ,
∗ : Not satisfied against adversaries described in section III-B

Our method also makes latency because of a challenge-response authentication. The Dynamic Cookies Rewriting Technique rewrites the cookies and keeps the original values of the cookies at proxy. So when it fails to write them back to original values, even the valid user cannot properly access to the server. Our method needs to synchronize a password between the user and the server. When the synchronization fails, the authentication fails. Both methods have a possibility to induce errors such that cookies do not properly operate.

As for usability, using session cookie in section III-A requires valid users to input their login information every time they login to their account. In contrast, users are required to input their login information only once in our method.

VI. CONCLUSION

When a server has XSS vulnerability, an adversary can obtain user's cookies. To minimize the influence of XSS indirectly, we introduced a method for preventing abuse of stolen cookies. It uses one-time password and challenge-response authentication to judge whether an accessing user is valid or not. With keeping usability, it can prevent the abuse of stolen cookies after their expiry and offers anti-replay attack property. We plan to implement our proposed method and evaluate its feasibility including latency.

ACKNOWLEDGMENT

A part of this paper is supported by JSPS A3 Foresight Program. We would like to appreciate the support.

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, Computer 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, *Protecting 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 Oorschot, Scott A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, ISBN: 0-8493-8523-7, 1997.