2011 International Conference on Modeling, Simulation and Control
IPCSIT vol.10 (2011) © (2011) IACSIT Press, Singapore

Anti-replay API

Jatuphum Juanchaiyaphum[+]and Somnuk Puangpronpitag

(jatuphum.j@msu.ac.th, somnuk.p@msu.ac.th)

Faculty of Informatics Mahasarakham University Thailand

Abstract. A cookies replay attack is one of the most powerful web application attacking techniques. There have been several tools (such as Cain & Abel, Backtrack4, Ferret & Hamster) and variant techniques based on this attack (such as sidejacking) widely available for script-kiddies. From the literature, several solutions to this problem have been proposed. However, almost all of them rely on a full Secure Socket Layer (SSL). Unfortunately, full SSL with a CA-signed certificate can be rather costly for some organizations. Furthermore, there have been several attacking techniques (such as, SSL-strip, MitM with SSL decoding) targeting SSL. So, in this paper, we propose a new solution, called Anti-replay API to assuage the cookies replay attack problem. Also, we have run several experiments to evaluate the Antireplay API. The experimental results have shown its effectiveness and efficiency.

Keywords: MITM, Cookie Replay Attack, Sidejacking

1. Introduction

Data eavesdropping from web users is a serious problem towards websites over the internet. This problem has become a significant issue that attracts the attention of both service users and software developers. To steal the data while the other people browsing the website can be done via the process of Man In The Middle or Monkey In The Middle (MITM) [1]; meanwhile there have been several tools applied to solve this attack e.g. Cain & Abel [2] and Backtrack [3]. These tools can be used in which the user required no deep comprehension of IT network or security system. To be exact, by reading only a manual, the user can launch this attack to interrupt the system. Similarly, a huge number of sources of knowledge for using these tools are broadly available.

MITM not only detects the data while the website is functioning but also leads to Sidejacking [4], which is the process that the attacker exploits a cookies stored in the ID session by replaying it. After that, the attacker is able to use the system as the real user while the real user does not recognized this process. The tools used for Sidejacking includes Hamster's [5] and Ferret's [5].

Secure Socket Layer (SSL) [6] has been developed to solve the problematic MITM by encrypting the data during the communication between the entry and destination. So the sniffered data cannot be read. However, SSL still encounters some problems. By using SSL with its full function, the user needs to pay for SSL certificates legally issued by the Certificate Authority (CA) which involves a high expense. For instance, one of the reliable CA called Verisign [7] requests approximately \$399 - \$1499 per year for its SSL certificate [7], which considerably is highly expensive for small and medium enterprises.

A number of websites commonly use HTTPS for the user's authentication in order to prevent the data detection. Nevertheless, after passing the authentication, the system returns to HTTP as usual; for example, hotmail.com, yahoo.com, facebook.com, etc. Practically, the function of SSL undertakes the data encryption and decryption during the communication, so the functioning is slowed down. Thus, the websites chooses to

[+] Corresponding author. Tel.: +6643754359.
E-mail address: jatuphum.j@msu.ac.th

use HTTPS only for the user's authentication, and the possibility of being attacked by Sidejacking becomes wider.

According to the mentioned problem, this study proposes Anti-replay API to prevent Sidejacking by using Anti-replay Token, developed from the study of a Secure Cookie Protocol carried out by Kovacs et.al. [8], for authenticating the client who requests the data from the server.

Specially, the researchers have developed an Anti-replay API model via PHP to be used together with Moodle Version 1.9.2 [9] as well as to examine Sidejacking with Cain & Abel functioning along with Hamster's and Ferret's. Finally, the result indicated that the Moodle equipped with Anti-replay API gave more effective performance in preventing Sidejacking than an old-version Moodle and it showed no requirement of additional software, as well as the application was simple.

2. Related Studies and Theories

2.1. Cookie Replay Attack

Cookie Replay Attack is the process existing when an attacker detects a victim's cookies and exploits it to request the data from a server. After the server receives the data and mistakenly assumes that the request belongs to the real user, by viewing the data from the cookies sent along with the request, and responses with the requested data as illustrated in Figure 1.

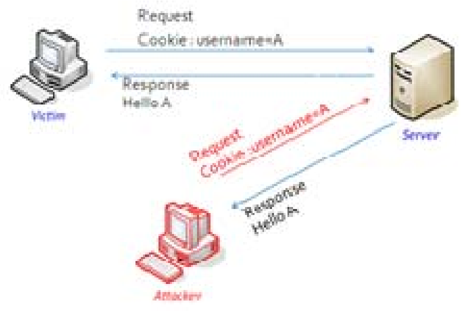


Figure 1: Functioning of Cookie Replay Attack

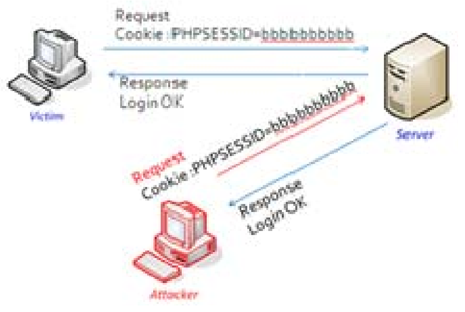


Figure 2: Functioning of Sidejacking

When the attacker is able to replay session id cookie it as presented in Figure2, this process is called Session Hijacking or Sidejacking [6].

2.2. Sidejacking

Sidejacking was formerly invented by Robert Graham, a chairman of Errata Security, a manufacturer in the field of computer network security, in USA. The attack of Sidejacking firstly spread out in 2007 at the Black Hat Conference. At the conference, Graham demonstrated Sidejacking by entering the victim's Gmail inbox with the developed tool, via the use of both Cookie Replay Attack and Session Hijacking.

The tools used for Sidejecking are Hamster's [5] and Ferret's [5]. Ferret's functions for capturing a session id cookie; meanwhile, Hamster's will send the captured session id cookie to the server to deceive the server that it is the real user's. This enables the attacker to hack in the user's system with an equivalent authority to the real user.

2.3. A Secure Cookie Protocol

In 2005, Kovacs et. al. [8] proposed the study of Secure Cookie Protocol as a solution toward Cookie Replay Attack by fulfilling the weak in Fu's Cookie Protocol [10] with the following features: 1) Authentication, 2) Confidentiality, 3) Integrity, and 4) Anti-Replay. For the prevention of Cookie Replay Attack, SSL session key was used as a component in creating a Digest for the data confirmation in cookies as illustrated in Figure 3. Since SSL session key was the data that the client sent to the server in SSL Handshake Protocol, the received Digest became a specific data that could authenticate the client for sending the cookies between the client and the server.

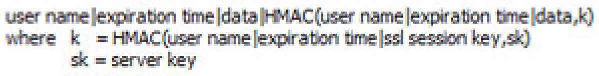

user name|expiration time|data|HMAC(user name|expiration time|data,k)
where k = HMAC(user name|expiration time|ssl session key,sk)
sk = server key

Figure 3: Components of Secure Cookie Protocol

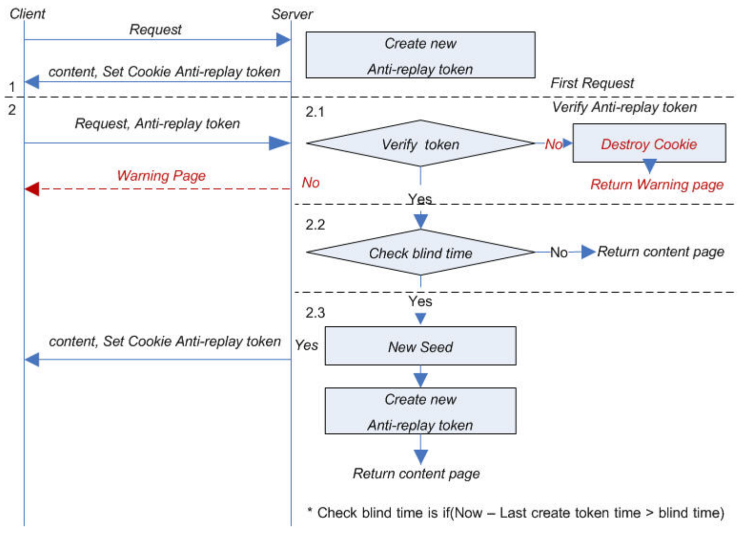
From our experiments, we found that Secure Cookie Protocol [8] could perform its functions perfectly with HTTPS. Nonetheless, it could not prevent Cookie Replay Attack when functioning with HTTP since SSL session key as an essential component of the digest creation for the cookies confirmation could not perform well with HTTP. So, this technique can work out properly only when the communication existed on HTTPS.

Indeed, using Secure Cookie Protocol with HTTP could not prevent Sidejacking because Secure Cookie Protocol was designed only for preventing the data changes in cookies. In contrast, if the attacker captured and replayed session id cookies without changing the cookies, this cookie replay could be possible before the cookies expired.

3. Development Information

3.1. Functions of Anti-Replay API

Functions of Anti-replay API employ Anti-Replay Token as a client's authentication. Each time that the client makes a request, the server will investigate the Anti-replay Token that the client has sent in a form of cookies. If the sent Anti-replay Token was acceptable, the server will create new Anti-replay Token and send it back in a form of cookies with the client's requested data. On the other hand, if the Anti-replay Token has been proved as faked or expired, the server will destroy it before sending the attacker to the warning page with a caution message to inform that the user is being under the attack. In details, the functioning procedure of Anti-replay API is shown in Figure 4 with the followings details.



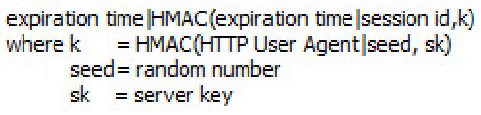
Client
Server
Request
Create new Anti-replay token
content, Set Cookie Anti-replay token
1
First Request
2
2.1
Verify Anti-replay token
Request, Anti-replay token
Verify token
No
Destroy Cookie
Warning Page
No
Return Warning page
Yes
2.2
Check blind time
No
Return content page
Yes
2.3
content, Set Cookie Anti-replay token
Yes
New Seed
Create new Anti-replay token
Return content page
* Check blind time is if(Now – Last create token time > blind time)

Figure 4: Functioning of Anti-replay Token

1. In case of the very first request, the system will create *Anti-replay Token* and send it back together with the client's requested data.
2. If it is not the first request:
    - 2.1. The system authenticates the sent *Anti-replay Token*. If unacceptable, it will be sent to the warning page and the cookie will finally be killed.
    - 2.2. In case that the sent *Anti-replay Token* is acceptable, the time-range between the last request and the recent point of time will be comparatively investigated in order to avoid the overlapping of the client's requests (caused by the user's double clicking). If the time-range is shorter than the determined *blind time*, the server will send back the data that the client requests without creating new Anti-replay Token.
    - 2.3. In case that the time-range was longer than the determined *blind time*, the server will create new *Anti-replay Token* and send it back with the client's requested data.

3.2. Anti-replay Token Creation

This study applied HMAC as mentioned in the RFC 2104 [11] document to create the Digest from the types of the client's browser (*HTTP User Agent*) combined with the randomized number (*seed*). The server's secret key (*sk*) was used as the key for creating the Digest and this key (*k*) would further be the key for creating the Digest, which was an essential component of Anti-replay Token as presented in Figure 5.

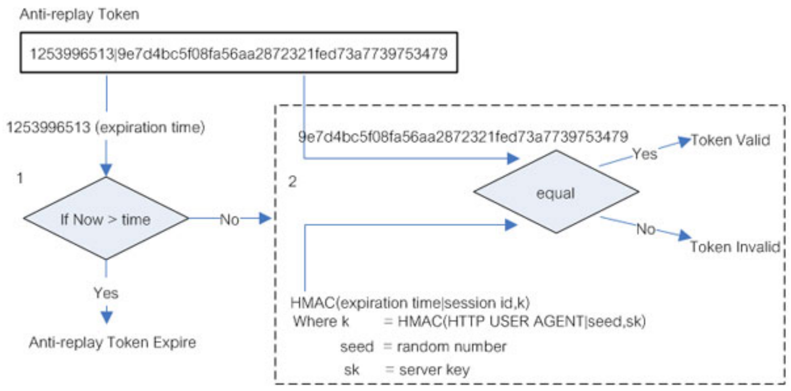


```
expiration time|HMAC(expiration time|session id,k)
where k      = HMAC(HTTP User Agent|seed, sk)
        seed= random number
        sk    = server key
```

Figure 5: Components of Anti-replay Token

Anti-replay Token was always changeable for each time that the client made a request, which resulted from the randomized number and an expiration time of Anti-replay Token, which was changeable as well. With this feature, the cookie replay was hardly possible.

3.3. Anti-replay Token Investigation



Anti-replay Token
1253996513|9e7d4bc5f08fa56aa2872321fed73a7739753479
1253996513 (expiration time)
1
If Now > time
No
Yes
Anti-replay Token Expire
2
9e7d4bc5f08fa56aa2872321fed73a7739753479
equal
Yes
Token Valid
No
Token Invalid
HMAC(expiration time|session id,k)
Where k = HMAC(HTTP USER AGENT|seed,sk)
seed = random number
sk = server key

Figure 6: Procedure of Anti-replay Token Investigation

The procedure of Anti-replay Token Investigation illustrated in Figure 6 was as the following details.

1. The expiration time of Anti-replay Token was investigated. If it was found expired, the system would send back the value as 1 (an expired cookie).
2. If the token was still activated, there would be a comparison between the Digest of *Anti-replay Token* and of HMAC (expiration time | sessionid, k). At this point, k = HMAC (HTTP User Agent | seed, sk) *seed* was a number in the variable session and *sk* was the server's secret key. If the data was found mismatched, the system would send back the value as 2 (cookie replay detected).

4. Experimental Results

In this study, experiments have been divided into 2 sections including an investigation of algorithm's functioning time and the experiments of the Sidejacking preventing performance. The experimental results are discussed as follows.

4.1. Algorithm's Functioning Time

The experimental results of the algorithm's functioning time after compared with Secure Cookie Protocol functioning on HTTS (Secure Cookie HTTPS) and Secure Cookie Protocol functioning on HTTP (Secure Cookie HTTP) have been shown in Table1.

Table 1 Comparison of Algorithm's Functioning Time

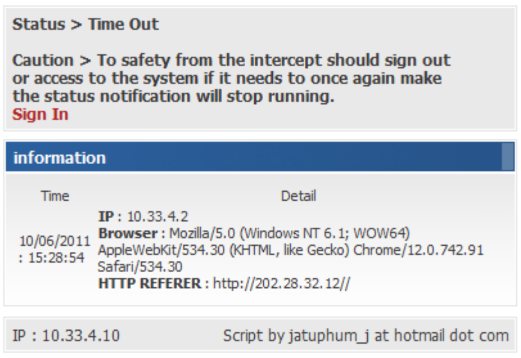
Source	$\bar{X}$ (ms)	S.D.
Secure Cookie HTTP	24.01±1.86	9.51
Secure Cookie HTTPS	69.51±1.04	5.33
Anti-replay	25.17±1.30	6.66

Table 2: The Result of the Sidejacking Prevention

Functioning Model	Prevention
Moodle HTTP	✗
Moodle HTTPS	✓
Moodle Anti-replay API	✓

4.2. Prevention of Sidejacking

After testing Sidejacking with Cain & Abel [2] functioning together with Hamster's and Ferret's in attacking the Moodle equipped with Anti-replay API, the results have been shown on the warning page informing that the cookie replay was found on the user's computer as in Figure 7; meanwhile, the computer would display a caution message warning that the user's data was being detected as in Figure 8.



Status > Time Out

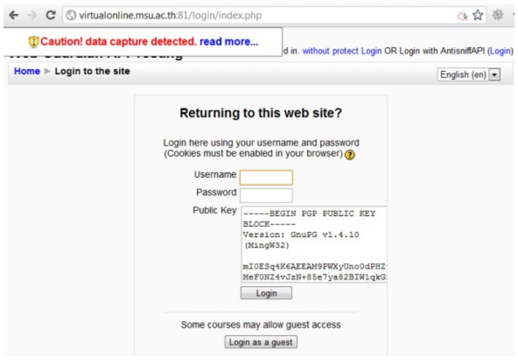
Caution > To safety from the intercept should sign out or access to the system if it needs to once again make the status notification will stop running.
Sign In

information

Time	Detail
10/06/2011 : 15:28:54	**IP :** 10.33.4.2 **Browser :** Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/534.30 (KHTML, like Gecko) Chrome/12.0.742.91 Safari/534.30 **HTTP REFERER :** http://202.28.32.12//

IP : 10.33.4.10 Script by jatuphum_j at hotmail dot com

Figure 7: Warning Page of Sidejacking



virtualonline.msu.ac.th:81/login/index.php

Caution! data capture detected. read more...

d in. without protect Login OR Login with AntisniffAPI (Login)

Home ▶ Login to the site

Returning to this web site?

Login here using your username and password
(Cookies must be enabled in your browser)

Username
Password
Public Key

```
-----BEGIN PGP PUBLIC KEY
BLOCK-----
Version: GnuPG v1.4.10
(MingW32)

mI0ESq4K6AEEAM9PWXyUno0dPHZ
MeF0NZ4vJzN+85e7ya82BIW1qkG
```

Some courses may allow guest access

Figure 8: Caution Message for the User

Comparing the results of the Sidejacking prevention by using the Moodle, equipped with Anti-replay API, with Moodle functioning on HTTP, as well as with Moodle functioning on HTTP, it was revealed as in Table 2.

5. Conclusions

Based on the experimental results, we summarize that the Anti-replay API significantly gives the effective performance in preventing Sidejacking and takes a shorter functioning period than Secure Cookie functioning on Protocol HTTPS. It also takes similar functioning period of Secure Cookie Protocol functioning on HTTP.

Using the notion of Anti-replay API, the software developers are able to develop Anti-replay API via the old systematical language without any requirement of additional software, and it can be simply applied into practical working as well.

6. REFERENCES

[1] Burkholder, P. *SSL Man-in-the-Middle Attacks*. [Online] 1 February 2002 cited 15 July 2008]; Available from: Available from: http://www.sans.org/reading_room/whitepapers/threats/480.php.

[2] *Cain & Abel*. 2009, Massimiliano Montoro.

[3] *Backtrack*. 2009, remote-exploit.org.

[4] Graham, R. *SideJacking with Hamster*. [Online] 5 August 2007 cited 25 October 2009]; Available from: Available from: http://erratasec.blogspot.com/2007/08/sidejacking-with-hamster_05.html.

[5] *Hamster Sidejacking Tool*. 2009, Errata Security.

[6] Dierks, T. and P. Karlton, *The TLS Protocol*. December 1999, IETF.

[7] VeriSign, I. *Compare All SSL Certificates*. [Online] 2008 cited 15 July 2008]; Available from: Available from: http://www.verisign.com/ssl/buy-ssl-certificates/secure-site-services/index.html.

[8] Kovacs, J., C. Huang, and M. Gouda. *A Secure Cookie Protocol*. in *Proceedings of 14th International Conference on Computer Communications and Networks*. 2005. San Diego, California, USA.

[9] *Moodle*. 2009, Moodle.

[10] Fu, K., et al. *Dos and Don'ts of Client Authentication on the Web*. in *Proceedings of the 10th USENIX Security Symposium*. 2001. Washington, D.C.

[11] Krawczyk, H., M. Bellare, and R. Canetti, *HMAC: Keyed-Hashing for Message Authentication*. February 1997, IETF.