2011 International Conference on Network and Electronics Engineering
IPCSIT vol.11 (2011) © (2011) IACSIT Press, Singapore

Designing of Non-Static Web Session

Preecha Noiumkar, Thawatchai Chomsiri, Krittikorn Kunkang, Sarawut Matloeng
Faculty of Informatics, Mahasarakham University, Thailand
preecha.n@msu.ac.th, thawatchai@msu.ac.th, zarahayo@hotmail.com, sonova_sarawut@hotmail.com

Abstract. A large number of web sites, including e-commerce and famous websites such as Hotmail, Gmail, Yahoo mail, Hi5, and Face-book, are vulnerable to Session Hijacking, whereby a hacker impersonates a user. Such attacks can be launched by capturing Cookie/Session IDs within an LAN, or by using XSS (Cross Site Scripting), which allows hackers to steal cookies from across the world and then use the captured Cookie/Session ID to access the system on a victim's identity. This problem is a result of using a Static Session ID. This research proposes a model to protect against Session hijacking by using a Non-Static Session ID instead of a Static Session ID. With this model, a victim's session ID captured by hacker will not be able to be used for replay attacks. We will demonstrate our model's effectiveness and ensure its high level of security by describing each step.

Keywords: Web Security, Web Session, Dynamic Session, Session Hijacking, Hack

1. Introduction

Current web technology is extremely important in our daily lives. We use e-mail, e-commerce, and social networking web sites. Security is the first issue that we must consider, because hackers have many ways to attack web sites, including Hijacking, XSS (Cross Site Scripting), SQL Injection, CSRF (Cross-Site Request Forgery), and so on. Hackers today are focused on attacking users instead of web sites. Several hackers use a popular technique called Session Hijacking. They can impersonate a victim by capturing a Cookie/Session ID on an LAN, or use XSS for stealing Cookies across a network, which can then be used to access the victim's account.

A session ID is usually sent via a URL, Hidden Fields, or most commonly, cookies. Currently, hackers use Static Session IDs, which make stealing a Session ID an easy job. Capturing (sniffing) data on an LAN can be done by exploiting the vulnerability of the ARP protocol. One popular technique, ARP Spoof, has many built-in support tools, such as Switch Sniffer [1], WinARP Spoofer, and 'arpspoof' for Linux. In addition, we discovered that there are unusual techniques for capturing cookies that do not involve the ARP spoof. We start by setting an IP address and an MAC address that match those of the gateway router, and then send a few packets to a switch [2]. We experimented with the impersonation of a test victim with popular free web-mail services, including Hotmail, Gmail, and Yahoo mail [3], and we found that these web sites are vulnerable. To impersonate a victim, we can easily use a tool such as Side jacking [4]. We tested Side jacking with Hotmail and Face-book, and we found both to be susceptible to attack. We continued the study and found that there are strong relationships between ARP Spoof, Sniffing packet, XSS, and Session Hijacking attacks.

A Session Hijacking attack cannot be resolved by merely improving firewall rules. Although web sites use HTTPS for all requests, hackers are still able to steal cookies by using XSS. Moreover, HTTPS can be decoded by using Cain and BackTrack. Users have to be aware of a web site's certificate for mitigating HTTPS decoding, but hackers can fake genuine certificates with SSL Stripping [5]. Some web sites prevent Session Hijacking by checking source IP addresses. However, IP addresses are easy to fake, whether by setting it directly or by using tools [6][7]. Session Hijacking is the seventh most popular web site attack method (ranked by OWASP) [8][9], and this problem becomes more critical as the frequency of financial transactions on the web increases.

We found that problems are caused by using Static Session IDs and that Layer2 protocols, such as ARP, are vulnerable [10]. Although web sites can check source IP addresses to detect user addresses, hackers are still able to fake them. Our idea is to use Non-Static (Dynamic) Session IDs instead of Static. The idea is that each HTTP request must use a different Session ID to provide protection from replay attacks. We have designed a method which generates Dynamic Session ID from Secret, Static Session ID, and the client's session time by using equation below:

$$\text{DynamicID} = \text{Hash}(\text{ Secret} + \text{ClientTime }) + \text{StaticID} + \text{ClientTime}$$

Servers are able to use the StaticID and ClientTime (in DynamicID) to find the Secret, and then they compute the results of Hash(Secret + ClientTime). If the results match Hash(Secret + ClientTime), which is derived from the DynamicID, the session is allowed.

Note: - The notation "+" means appending a string.

- The notation "–" means subtracting a string.

Both client and server have to use standard timing, such as NTP (Network Time Protocol, which adjusts their time correctly), because our model uses the difference between the client's time and the server's time.

2. Backgrounds and Related works

Timothy D. Morgan [11] suggests that it is better to use HTTP Digest Authentication [12] than cookies, because cookies are more vulnerable to attack than HTTP Digest Authentication. He shows that HTTP Digest Authentication lacks a customizable login form, an application-driven logout, or a choice in hash storage format, and the potential exists for exposure of weak passwords. So, he proposes minor changes in browser behaviour and an HTTP standard for a higher level of security. The RFC 2617 - Digest Access Authentication Scheme [12] is an authentication between a browser and a web server, such as Apache or IIS. It does not authenticate across Web Applications, such as PHP and ASP. In this scheme, the username and password must be created on a server's OS by using the "htdigest" command on Linux, for example, which is not a username/password in the SQL Server / Oracle / MySQL Database accessible by web applications. This scheme was rated at a lower level than web applications level. Furthermore, it does not use dynamic sessions since session IDs can be reused after authentication. This standard has some problems with proxy forms, and it is less secure than the Kerberos and client-side private-key scheme.

Chris Masone, Kwang-Hyun Baek, and Sean Smith [13] suggested a new model for preventing Web Spoof and DNS Spoof attacks, which they implemented on Firefox. Their Firefox extension will remember a server's public key and verify it upon each HTTP request. Even if the server's public key is changed (and the client is redirected to a fake web site), the Firefox extension will remove the cookie immediately. Their model is limited, however, to Web Spoof and DNS Spoof attacks. Anderson A. L. Queiroz and Ruy J. G. B. de Queiroz [14] show how a cookie can be stolen with XSS and then used to impersonate a victim. Their solution includes using cryptographic schemes in the cookies' values and substituting HTTP protocol with HTTPS. However, this research does not include detailed methodology.

Noiumkar and Chomsiri [3] have shown how to impersonate a victim by using special browsers, such as Opera or Firefox, with a cookie editor add-on. Cookies can be stolen with ARP Spoof, and then they are captured on an LAN with ethereal. Side Jacking [4] is the tool that makes this process easy. They tested Hotmail, Gmail, and Yahoo mail and found that all three are vulnerable to user impersonation. The focus of the research is on the measurement of the security level of a web site.

Chin-Ling Chen and I-Hsien Lin [15] propose a key management algorithm on sensor networks by using x-y position for generating a dynamic session key. This research focuses on sensor networks, not web sites. Ruopeng Ye, Agnes Chan, and Feng Zhu [16] have shown that their schemes can effectively stop replay-attacks from expired cookies. Their solution consists of a simple scheme and an MK scheme. A simple scheme is very easy to use, and while the MK scheme is more complex, it is more secure. Their methodology utilizes multiple fields to generate a session ID, such as a username and time stamp. A. Miklas and S. Jain [17] propose cookie/session management by using key pairs. Their procedure is to make a nonce and then to encrypt it with a private key. Robert Bobek [18] proposes cookie security improvement by using a "CookieCard," which is stored in a smart card. Users must have the card and a PIN. This method depends on hardware (CookieCard) and a PIN. Markus Huber, Martin Mulazzani, and Edgar R. Weippl [19] demonstrate how to hack users on a social network, such as Facebook. They propose a "Friend injection attack," such as session hijacking (for cloning an HTTP header, including session cookies). Their procedure begins with the use of Sniffing packets, then an HTTP header (session cookie) to impersonate a victim, and lastly invitation requests are sent to other e-mail addresses (a phishing web-page for other hacking purposes).

3. Designed Model

We have designed a model which consists of two main phases, shown in Figures 1 and 3 respectively: the "Authentication phase" and the "Normal phase."

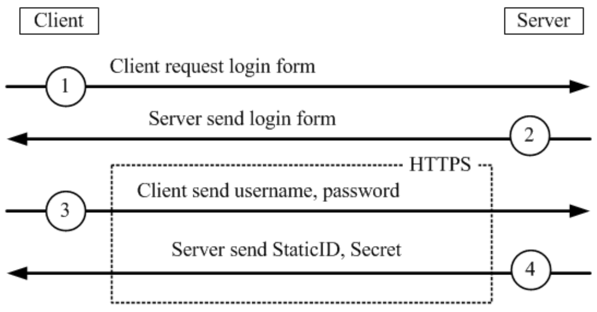
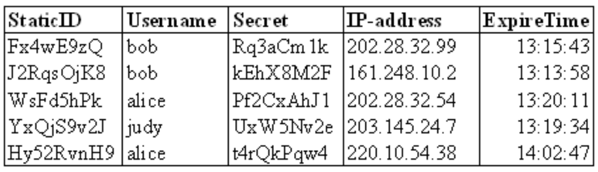

Client
Server
1
Client request login form
Server send login form
2
HTTPS
Client send username, password
3
Server send StaticID, Secret
4

Fig. 1. The processes inside authentication phase.

The process of the authentication phase, begin with the client request for a server's web page (step 1), after which the server sends a login page to the client (step 2). After the user has received the login page, such as https://www.msu.ac.th/login.php, he enters his username and password, and clicks the login button. After that, the process will reach step 3, in which the browser sends the username and password to server. After the server validates the username and password, it will generate a StaticID (Static Session ID) and a Secret for a user (step 4). Then it updates the table that located in server's memory (see Figure 2).



StaticID	Username	Secret	IP-address	ExpireTime
Fx4wE9zQ	bob	Rq3aCm1k	202.28.32.99	13:15:43
J2RqsOjK8	bob	kEhX8M2F	161.248.10.2	13:13:58
WsFd5hPk	alice	Pf2CxAhJ1	202.28.32.54	13:20:11
YxQjS9v2J	judy	UxW5Nv2e	203.145.24.7	13:19:34
Hy52RvnH9	alice	t4rQkPqw4	220.10.54.38	14:02:47

Fig. 2. The table that store matching between StaticID, Secret and Username.

The StaticID is an identifier for the user session. It is generated from random strings. StaticIDs may have expiring times similar to the expiring times of static session IDs in the traditional scheme. The Secret (Fig 2.) is generated randomly by the server in the same way the StaticID is. No one knows the Secret except the server and the client. Moreover, this table also records the client's IP address. If the client is behind a proxy or NAT, the recorded IP address belongs to the proxy/NAT.

The server will send the StaticID and the Secret to the client in step 4 (Figure 1), and after that the authentication phase has finished. Steps 3 and 4 require an https channel because the password, StaticID, and Secret are very confidential. The normal phase is step 5 -8 (Figure 3), in which a client has to compute the DynamicID before sending the HTTP request to the server.

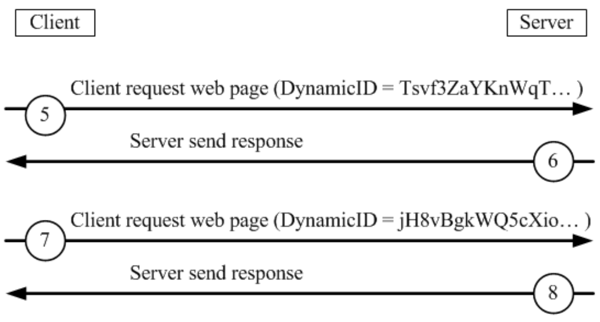

Client
Server
Client request web page (DynamicID = Tsvf3ZaYKnWqT…)
5
Server send response
6
Client request web page (DynamicID = jH8vBgkWQ5cXio…)
7
Server send response
8

Fig. 3. The processes inside normal phase.

Calculating the Dynamic Session ID (DynamicID) on the client side can be done by creating Part1_String and Part2_String. The Part1_String is computed from the equation below:

Part1_String = Hash(Secret + strClientTime)

Part2_String is calculated from:

Part2_String = StaticID + strClientTime

Lastly, DynamicID can be generated by using

DynamicID = Part1_String + Part2_String

Note: - The notation "+" means appending a string.
- The notation "–" means subtracting a string.

The strClientTime is a string that is converted from the client's time to a string format like "1290135252.953." This 14 byte string of data is the number of seconds from 1 January 1970 00:00:00 GMT to the present. The client will get the StaticID and Secret from the server during step 4, after which he must create a DynamicID before sending a request in step 5. Both the StaticID and Secret are stored in the browser memory.

For implementation, we can use MD5 or SHA1, or some other hashing function. If we use SHA1 (stronger than MD5), Part1_String will be as long as 40 bytes. If we implement a 20 character long StaticID, we will have a 40 + 20 + 14 = 74 digit DynamicID (see Figure 4). From this example, when we consider the first 40 bytes of the DynamicID (Part1_String), it will be changed with every request because the client's time has changed. The first 20 bytes of Part2_String may be consistent, because it is a StaticID, but the last 14 bytes is changed with the change of time. Although Part2_String can be sniffed or captured (hackers can read the Part2_String), they cannot decode the Secret from Part1_String because it is made with a hashing function.

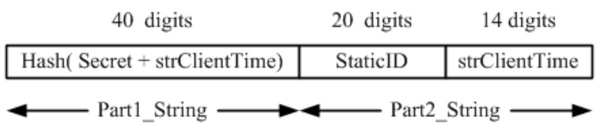


Fig. 4. DynamicID

On the server side, after it receives a request from the client, it must validate the DynamicID by answering a few questions, such as:

- Is DynamicID valid at this time?
- To which user does the DynamicID belong?
- Is this DynamicID real or fake?

Verifying a DynamicID can be done with the following six step procedure:

A. Read StaticID from part2 of DynamicID, and then use it for looking up Secret (and username) from the server's table.

B. Read strClientTime from part2 of DynamicID, and then append Secret (from A) with strClientTime.

C. Calculate Hash(Secret + strClientTime)

D. Read part1 of DynamicID and compare it with the results of C.

E. If the comparison in D is a match, this session belongs to the user in A. If it is not a match, this session is incorrect.

F. If the session in E is correct, verify ClientTime. If ClientTime is different from the server's time (ServerTime), the session is timed out.

Although we can use NTP, it is still hard to adjust the client's time to equivalence with the server's time. They may differ from each other by approximately 100-300 ms. Thus we have to use what is known as "Interval." Interval is the maximum difference between a client's and server's time. Normally, we would set the Interval to 1000 ms, or 1 s. If absolute (ServerTime - ClientTime) > Interval, it means that this DynamicID cannot be used. It is possible that hackers could steal this DynamicID and use it for replay attacks. In this research, we set Interval to 1 second. However, this number can be adjusted to more appropriate values later. After absolute(ServerTime - ClientTime) is checked and found to be less than Interval, the server has to check the age of the StaticID. If the StaticID has expired (see Figure 2), the server will redirect the client to the login page. If it has not expired, the server then goes to the checking IP address step. If the request has not come from the same IP address, as shown in the table (Figure 2), the user may have been hacked with MITM or XSS. It means that this DynamicID is unreliable. If the request comes from the same IP address, the server can ensure that this DynamicID comes from a real user (and not a hacker) by validating the signature.

4. Security Analysis

We have designed this model to protect web sites from Session Hijacking, by which hackers steal cookie/session IDs (by sniffing on an LAN or XSS over a network). Hackers use an obtained session ID to impersonate a victim. Hackers tend to use Sniffing cookie/session ID on an LAN after the authentication process. There are several session IDs sent to a server every time a user clicks an e-mail link in their inbox.

In our model, the session ID (DynamicID) has been changed with each request (see Figure 3). If hackers sniff (capture) some requests during a moderate amount of time, he will get a certain number of DynamicIDs. However, these DynamicIDs are unique. Some hackers select one of them for hijacking, while others prefer to test all of them. They spend time capturing, copying, and replacing session IDs to a special browser (Opera or Firefox with cookie-editor add-on). Because our model uses Interval, the hacker cannot use the session ID (DynamicID) quickly enough. Thus, Session Hijacking or replay attacking does not work with our model. In practice, XSS is used more often than sniffing on an LAN. Although there are tools today that help hackers hijack quickly (such as Data Tamper as a Firefox add-on or the web-scarab tool), hackers spend at least 10 seconds on a replay attack, while our Interval is only 1 second, and it can be decreased if necessary. If the hacker creates special software to implement a real time replay attack on our model, he must remain in the same network with the victim and will need to do a complete ARP spoof. Within these network environments, attacking our model can be done successfully only if the hacker and the victim use the same proxy or NAT. In the authentication phase, we assume that the https is secure enough. If a hacker can break the https and obtain a user's Secret, the hacker can then successfully impersonate the user. However, if the hacker breaks the https, it is not necessary to use the Secret and it wastes time writing a special program to launch the attack. He just uses the only password to access the victim's account easily, because the password is also sent in the same channel as the Secret.

5. Conclusion

This research proposes a model to protect against Session hijacking by using a Non-Static Session ID instead of a Static Session ID. With this model, a victim's session ID captured by hacker will not be able to be used for replay attacks.

6. References

[1] Zouheir Trabelsi and Wassim El-Hajj, On investigating ARP spoofing security solutions, *International Journal of Internet Protocol Technology*, Volume 5, Number 1-2, 2010, pp 92-100

[2] Chomsiri, T.; , Sniffing Packets on LAN without ARP Spoofing, *Third International Conference on Convergence and Hybrid Information Technology, 2008. ICCIT '08.* , vol.2, no., pp.472-477

[3] Noiumkar, P.; Chomsiri, T.; , Top 10 Free Web-Mail Security Test Using Session Hijacking, *Third International Conference on Convergence and Hybrid Information Technology*, 2008. ICCIT '08. , vol.2, no., pp.486-490

[4] Ryan D. Riley, Nada Mohammed Ali, Kholoud Saleh Al-Senaidi and Aisha Lahdan Al-Kuwari, Empowering Users Against SideJacking Attacks, *ACM SIGCOMM Computer Communication Review*, Volume 40 , Issue 4 (October 2010)

[5] Moxie Marlinspike. New tricks for defeating ssl in practice, July 2009. http://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf.

[6] Goel, S.; Kumar, S.; , An Improved Method of Detecting Spoofed Attack in Wireless LAN, *First International Conference on Networks and Communications, 2009. NETCOM '09.* , vol., no., pp.104-108, 27-29 Dec. 2009

[7] SMAC: MAC-address Changer, http://www.softpedia.com/get/Network-Tools/Misc-Networking-Tools/SMAC-MAC-Address-Changer.shtml

[8] OWASP Top 10 2007, http://www.owasp.org/index.php/Top_10_2007

[9] Session hijacking attack, http://www.owasp.org/index.php/Session_hijacking_attack

[10] Wenjian Xing; Yunlan Zhao; Tonglei Li; , Research on the Defense Against ARP Spoofing Attacks Based on Winpcap, *Second International Workshop on Education Technology and Computer Science (ETCS)*, 2010 , vol.1, pp.762-765, 6-7 March 2010

[11] TD Morgan , Weaning the Web off of Session Cookies, 2010, http://www.packetstormsecurity.org/papers/web/WeaningTheWebOffOfSessionCookies.pdf

[12] J. Franks, P. M. Hallam-Baker, J. L. Hostetler, S. D. Lawrence, P. J. Leach, A. Luotonen, and L. C. Stewart. HTTP authentication: Basic and digest access authentication. *IETF RFC 2617*, June 1999.

[13] Chris Masone, Kwang-Hyun Baek, and Sean Smith. WSKE: Web server key enabled cookies. *In Proceedings of Usable Security 2007 (USEC '07).*

[14] Anderson A. L. Queiroz and Ruy J. G. B. de Queiroz, Breach of internet privacy through the use of cookies, *Proceedings of the 3rd International Conference on PErvasive Technologies Related to Assistive Environments*, Samos, Greece, 2010.

[15] Chen, Chin-Ling; Lin, I-Hsien. 2010. Location-Aware Dynamic Session-Key Management for Grid-Based Wireless Sensor Networks. *Sensors*, 10, no. 8: 7347-7370.

[16] Ruopeng Ye, Agnes Chan, Feng Zhu, Efficient Cookie Revocation for Web Authentication, *IJCSNS International Journal of Computer Science and Network Security*, VOL.7 No.1, January 2007.

[17] A Miklas, S Jain, Smart Cookies The Unstealable Authentication Cookie, 2006, http://www.eecg.toronto.edu/~lie/Courses/ECE1776-2006/Projects/SmartCookies-proposal.pdf

[18] Robert Bobek, HTTP COOKIES EXPLOITING THE USER,2006, http://web2.uwindsor.ca/courses/cs/aggarwal/cs60564/Assignments/Bobek_privacy.doc

[19] Markus Huber and Martin Mulazzani and Edgar R. Weippl, Who On Earth Is Mr. Cypher? Automated Friend Injection Attacks on Social Networking Sites, *in Proceedings of the IFIP International Information Security Conference 2010: Security & Privacy*, Silver Linings in the Cloud, 2010.