I.J. Information Engineering and Electronic Business, 2014, 5, 10-15
Published Online October 2014 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijieeb.2014.05.02

CSRFDtool: Automated Detection and Prevention of a Reflected Cross-Site Request Forgery

Omar A. Batarfi, Aisha M. Alshiky, Alaa A. Almarzuki, Nora A. Farraj
King AbdulAziz University/Faculty of Computing and Information Technology Jeddah, 21542, Saudi Arabia
Email: { obatarfi, amalshiky}@kau.edu.sa, {aalmarzuki0001, nfaraj0003}@stu.kau.edu.sa

***Abstract*—**The number of Internet users is dramatically increased every year. Most of these users are exposed to the dangers of attackers in one way or another. The reason for this lies in the presence of many weaknesses that are not known for ordinary users. In addition, the lack of user awareness is considered as the main reason for falling into the attackers' snares. Cross Site Request Forgery (CSRF) has placed in the list of the most dangerous threats to security in OWASP Top Ten for 2013. CSRF is an attack that forces the user's browser to send or perform unwanted request or action without user awareness by exploiting a valid session between the browser and the server. When CSRF attack success, it leads to many bad consequences. An attacker may reach private and personal information and modify it. This paper aims to detect and prevent a specific type of CSRF, called reflected CSRF. In a reflected CSRF, a malicious code could be injected by the attackers. This paper explores how CSRF Detection Extension prevents the reflected CSRF by checking browser specific information. Our evaluation shows that the proposed solution is successful in preventing this type of attack.

***Index Terms*—**OWASP, CSRF, HTTP, CSRF Detection Extension, reflected CSRF, Chrome extension.

I. Introduction

As known, the security must be considered to satisfy web users. One of the threats that may occur during using the web is Cross Site Request Forgery (CSRF) [1]. This attack has placed in the list of the most dangerous threats to security in OWASP Top Ten for 2013 [1]. Also, it is one of the top ten attacks that HTML5 new features can increase its threat [2]. It is considered as an active, application layer attack [3]. Although the Internet security is improving at super speed by adopting new technologies, however attackers still find vulnerabilities in websites and exploit them to carry out their attacks against servers and clients. One of the most dangerous attacks is accessing the confidential data that associated with user accounts (e.g., e-mail accounts, social networking accounts, and bank accounts which are considered the most sensitive information [4,5]. Penetration of these accounts will cause great harm to the Internet users where he/she may lose money and the attacker may use these data to perform unwanted and malicious operations. When the user initiates an action online such as submitting a form or doing registration on website, his/her data can be sent using two methods either "POST" or "GET" of HTTP protocol. Then the browsers will send the request depending on user cookies [5,6,7]. The cookies (session identification) define how the website (server) identify who you are and store your privileges in the your browser. The attacker gets the benefit of active session and sends a malicious link to the user, and then the user click on the link that is may tend to CSRF occurrence [8]. Therefore, the problem is how to protect the active sessions in order to prevent the attacker from stealing user's secret information, and using this information to initiate an unwanted action.

During the last five years, many efforts have been made by researchers to confront at this attack. Many researches have been published and many solutions have been proposed and developed. However, being one of the top ten threats in OWASP is a strong indicator that these solutions have limitations because the attackers are still capable of enforcing CSRF. Although these solutions seem to be efficient in stopping this attack, but clients and developers do not use them in their browsers and websites respectively [9]. This is probably because of the overhead introduced by the proposed solutions in terms of time and space.

This paper proposes a Google Chrome Extension that is able to eliminate a specific type of this attack which is Reflected CSRF. Google Chrome Extension gets the required information of web browser (tab ID, browser window ID and IP address) that is used by the user to login to a website, and then it concatenates this information with an active session to prevent any attack from different sources such as different taps in the same window.

This paper sections arranged as follows: section 2 is an overview of CSRF attack. Then section 3 discusses different solutions were proposed for detecting and preventing this attack. Section 4 states the problem and the hypotheses. The solution methodology and their test are discussed in section 5 and 6 respectively. Authors' simulator is presented in section 7. Then section 8

discusses the results. Finally, section 9 concludes this paper and suggests future work.

II. Overview of Cross Site Request Forgery

CSRF also may be abbreviated as XSRF. This attack also known as one-click attack or session riding. In the beginning of 2001 the first CSRF attack was registered. CSRF is an attack that forces the user's browser to perform unwanted request or action without user awareness by exploiting the valid session between the browser and the server [5]. Fig. 1 illustrates this attack.

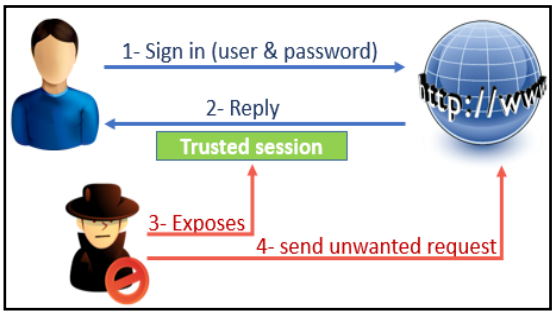

Fig 1. CSRF attack mechanism.

Consider the following scenario: a user login to a bank website using an individual username and password. Bank web server verifies user authorization for requested service and initiates a valid session with client. Attacker uses illegal strategies to deceive the novice user or even the experts to send unintended money transfer request by attract them to a trap (fake link) which is hosted on untrusted third party server. When the user clicks that link, the CSRF attack started [10].

Here is an example of an attack that aims to transfer an amount of money without the user's consent:

- Let us say that the next link is the link for transferring money from account to another: http://hackedbank.com/transfer.php?account=sender &amount=amount&for=reciever
- The harmful link will be hidden behind image tag as: <imgsrc="http://hackedbank.com/transfer.php?accout =sender&amount=amount&for=reciever">

Despite this attack appears easy, it requires a hacker to know many things to be able to implement the attack successfully. For example:

- Attacker must find a bank website that does not validate the referrer header.
- Then Attacker must be aware of the form submission that used for money transaction and the attacker should be able to write the values correctly.
- Finally, the attacker must find a smart and perfect way to attract the user to the harmful link, then to lure her/him to click the element that start the attack and trigger the malicious code of CSRF attack [5].

The actual steps followed by attacker are:

- First: Attacker has to study the target website in order to understand its functionality. For example, it is so helpful to know the structure of the form used for money transaction on bank website.
- Second: Understanding the functionality is still not enough where attacker has to identify the weakness and vulnerabilities of the target website. He can expose the old cookies or exploits the weak protection.
- Finally: Attacker has to test the malicious code to ensure it works as desired.

There are two types of CSRF attack (reflected and stored) [11].

A. Reflected Cross-Site Request Forgery

This type of CSRF attack is the main subject of this paper. In reflected CSRF, the malicious code could be injected in a fake website by the attacker which emulates the target one. Then, it can expose the valid session between the user and the genuine website [8]. Attacker has to trick user to click the malicious link that will trigger the malicious code. Reflected CSRF is well illustrated by using data flow diagrams in [2, 5]. Fig. 2 depicts this attack.

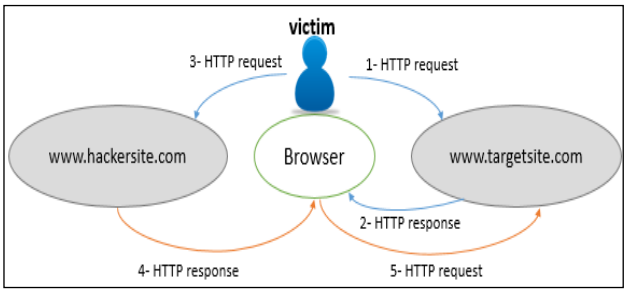

Fig 2. DFD model of reflected CSRF attacks

B. Stored Cross-Site Request Forgery

In stored CSRF attack, the malicious code already exists in the genuine webpage. It can be downloaded from a trusted web server. Stored CSRF can be found in blogs and forums [8].

For both types of CSRF, the malicious code may be hidden under several HTML tags (e.g. IMG, SCRIPT, IFRAME or Image Object) [5].

The difference between both is very clear, it is in the way by which the malicious code is being delivered. First type is triggered from third party domain (untrusted domain), while the second is executed in the same domain (targeted domain). The common factor between them is "both require knowing the target website's functionality" [12].

III. Literature Review

In the last five years, there have been a lot of proposed and developed solutions in order to protect users against CSRF attack. These solutions adopt different techniques. This section discusses three different solutions, that are seems similar to the proposed solution in this paper and similarities and differences between them are explained.

In 2009, Masaru proposed different schemes in order to solve three different problems [13]. The concerned schema is proposed to protect the browser against CSRF

that has ability to hide the violation of the same origin. It was the major property that the author had to overcome. In this schema each request has 2 checksums, the first is generated by the browser, and the second is generated by the server. The first is generated from applying hash function of two values (browser secret key and URL of the server) where the second one is generated from applying hash function of two values (incoming checksum and server secret key). Next, the browser is responsible for comparing each new second checksum that coming from server as a reply for each request. By doing this, it can detect the reply that comes from different server. Masaru's solution concentrated to identify the evil request in malignant website page.

Regarding to the overhead issue, there is a performance overhead as a result of implementing three exponentiations and two hashes. In addition to this, the proposed solution was not implemented; therefore, there are no evaluation results about the effects of this solution. When the trusted session started, the browser does not except any manual changing of the address in that tab until the session end. Masaru's solution based on using secret keys for both browser and server and each server should register in a common deposit agent that holds all secret keys for all servers which is relatively hard to implement. Moreover, the browser should retrieve the secret key for server for each request. The connection with the deposit agent infrequent will impose more performance overhead.

Our research proposed a solution relies on useful information that can be provided by the browser directly without intervention of any third party. This information helps to differentiate each session with its startup tab and other information.

Reference [14] stated that Hossain and Mohammad implemented a prototype for Firefox plug-in to protect client from CSRF attack. The proposed plug-in consists of four modules, and each module has its role in detecting and handling this attack as following:

1) Request module checks if the request is POST or GET request with parameters are filled in the form, and forwards the request to the next module.
2) Window and form module checks two things:
   - If none of the open windows is displaying a webpage from the destination domain of the request, the request is counted as an attack, and the attack handler module takes its turn.
   - If there is no form in the opened windows, it considers the request as an attack and forwards the request to the attack handler module.
3) Content checking module differentiates or modifies the request by removing all parameters. Then, the modified version is launched, and the response contents are checked to see whether they are similar to the expected contents or not. If they are different, the attack handler takes its turn.
4) Attack handler module stops the request and warns the user.

Although it seems efficient, this solution could make the browser slower by doing all these checks. In addition, it does not consider browser tabs and the client is responsible for detecting and preventing the CSRF attack because the server is not involved.

In 2011 a paper titled "A Study of the Effectiveness of CSRF Guard" has discussed one of the strategies that has attempted to prevent and block CSRF attack [15]. CSRF Guard has tried to verify the integrity of HTTP request by injecting a different security token to each active HTTP session between the authenticated client and web server. The protection that the CSRF Guard has offered against the CSRF attack deepened on the token generation and validation. The token has been injected with specified protected resource. Then the token verified when the user request protected resource.

- There is a similarity with our proposed solution in the server check process with each request. However our solution is different in the content that the server checks. In CSRF Guard, server checks for a valid token which is associate with session but in our solution the server will check on stored static information (tab, window and IP ID) which are associate with session whether it is the same or not.
- In CSRF Guard the token might be stolen by using malicious JavaScript. The attacker could steal the token from the active session between the client and the server. It is difficult to protect against a CSRF if there is a script in the webpage which sends the CSRF token to the attacker. After that, the CSRF attack can occur from different sources by using stolen token. While in our proposed solution if the information that server stored has been stolen and there is CSRF attempt from different sources, the server will reject this attempt because it comes from different sources (tab ID, Window ID).
- The CSRF Guard and our proposed solution could not defend against login CSRF attack because all of checking processes are done after the login process and depend on active session which initiated depending on login and associate with it.

IV. Methodology

To tackle the problem, we have created a Google browser extension (client–side) that helps server in preventing the reflected CSRF from happening. In general, extensions allow the developer to add new functionalities to the Chrome browser. To develop a browser extension, some programming languages are needed such as HTML, JavaScript, CSS and Local server (WAMP server)

In CSRF Detection Extension, there are two main steps:

- First step is extracting the tab ID. This step is important to specify the request source whether comes from different tab or from the same one of a valid user. The box below shows the code for extracting the tab ID.


```
chrome.windows.getCurrent(function(currentWindow)
chrome.tabs.getSelected(currentWindow.id,function(s
electedTab)
TabID = selectedTab.id;
```

- Second step is sending the tab information to the Server. The below code shows how we post the information to the server.

```
window.onload = myOnload;
functionmyOnload() {
varjax = new XMLHttpRequest();
jax.open("POST", "http://localhost/log.php",true);
jax.send(TabID);
}
```

- After a user login, the extension gets and sends to the server userID, IP, browser window ID and tab ID.
- The server would store the information on the database and reply with session ID to the client. The server stores the information after hashing the information by Hash method and define the $Var variable that is the value of tab ID that is retrieved from extension. Next, the value of tab ID is hashed by SHA-1. After that store hashing value into tabs table that exist in the database into t_ID column.

This process repeats for each piece of information before inserts the information in database table.

- The extension sends the following information with every HTTP request: session ID, IP, browser window ID and tab ID.
- For sensitive requests that include post function, server creates hashes of information and verifies if the request is generated from the same tab of a browser. This verification is performed by comparing the stored hashing information with the hashing information that is sent with each request. The request will be executed if the comparing result is true otherwise the session will be destroyed. Fig. 3 shows all steps of the CSRF detection tool.

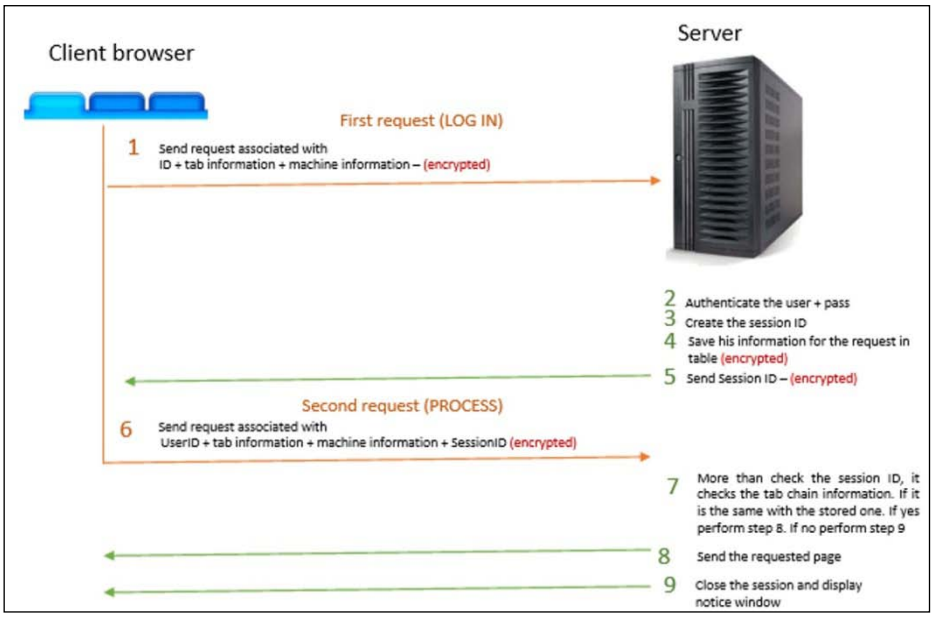

Fig 3. Steps of CSRF detection tool

V. Methodology Test

There are two famous purpose of the testing. First, to make sure that the result you find is the same as what you already specified. Second purpose is to know the problems and try to fix them before applying the proposed solution. We have done testing to ensure the efficiency of our solution.

- Test Environment:
  - WampServer is a Windows web development environment. It allows the developer to create web applications with server-side languages like PHP and MySQL database. Also it helps the developer to manage databases easily.
  - Teacher & Students Academic Communication is a website that is written by PHP, HTML, JavaScript and MySQL. It saves cookies that are important element in CSRF and creates sessions then it stores encrypted information of each request from clients.
- Test Scenario:

  Case 1: Without the CSRF Detection Extension

  1- The Administrator login to the website and tries to add a teacher (one of the most process that should be done only by administrator). Fig. 4 shows this process.

  2- The Administrator login to the website and the administrator clicks on malicious link (sent by attackers). Because there is no detection against CSRF attack in this website,

the attackers can easily trick the user through malicious link. Therfore, attackers can access the saved cookies and valid session on the victim's computer.

Fig 4. Teacher registration done

Case 2: With the CSRF Detection Extension
The Admin login to the website and tries to add a teacher. He clicks on malicious link that exploited the saved cookies and valid session. Fig 5 shows how the request from different sources will be rejected and the valid session will be ended as shown in dialog box.

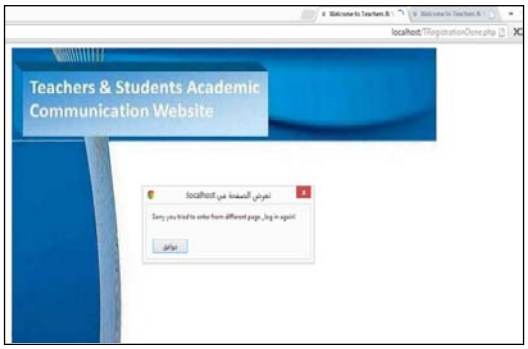

Fig 5. Teacher registration done page

VI. Simulation

In order to complete the implementation, a browser has been developed by using visual basic language in order to use it in the simulation process and also to test the result as shown in fig 6.

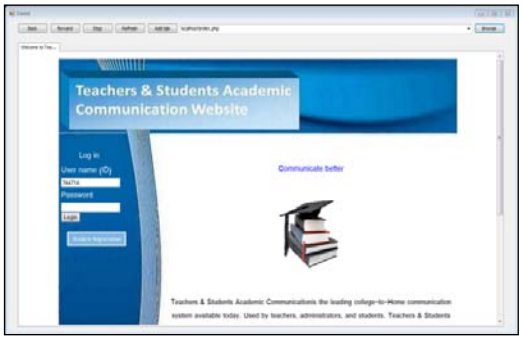

Fig 6. The browser simulator

The benefit of having our own browser is to perform all sorts of tasks over it. Therefore, it will help in sending any needed information to our server (tab index and IP address). As previously mentioned, we could get the tab index by using our Chrome extension, but there is a problem with Chrome extension open source that prevent the posting of the tab index value to the server. Therefore, the developed browser will assist in performing this function.

The tab index and IP address are sent to a shared file between the browser and the server. Hash function simply is applied to exchange data between the browser and server to provide data protection. As mentioned, the two needed values are:

- Tab index: to differentiate the session for specific tab and should be unique.
- IP address: one of the additional information that is used to provide more security over the tab index. IP address is used to identify the host. IP is used here to ensure there is no repetition of the session.

```
  Dim IP As String
  IP = GetLocalIP()
  Dim hashi, hashi2 As String
  hashi =
GenerateHash(TabControl1.SelectedIndex.ToSt
ring)
        hashi2 = GenerateHash(IP.ToString)
  Dim inputString As String
        inputString = hashi + " " + hashi2
+ vbNewLine

My.Computer.FileSystem.WriteAllText("C:\wam
p\www\tab.txt", inputString, True)
```

VII. Result and Discussion

By completing the test, CSRF Detection Extension has demonstrated the ability to prevent the occurrence of the reflected CSRF. Moreover, compared to the solutions mentioned in the literature review section, CSRF Detection tool detects the attack with less number of verification. This leads to faster detection.

The possible limitations of the CSRF Detection Extension are as follows:

- CSRF Detection Extension is designed just for Google Chrome browser and it does not work with other browsers.
- CSRF Detection Extension is limited to detect one type of CSRF attack that called reflected CSRF, while it cannot detect stored CSRF.
- CSRF Detection Extension makes connection between session id and static information (window browser ID, tab ID and IP address) to check user validity. Because of this dependency, it cannot defend login CSRF attack.

VIII. Conclusion

CSRF is placed in the list of the most dangerous threats to security in 2013 according to OWASP. As known, CSRF tends to use the valid session by illegitimate third party without user permission. This attack leads to expose victim's personal information such as bank accounts and any other kinds of critical information.

This paper has focused on one type of CSRF which is the reflected CSRF. The proposed solution has demonstrated that it can prevent CSRF attack from occurrence. CSRFD tool is developed to be able to distinguish between the legitimate and illegitimate users to secure web browser communication.

CSRFD tool detects the attack occurrence by checking composed information (user ID, IP, browser ID, tab ID) that associated with each request instead of checking only the user ID (valid session). The proposed solution is implemented and tested by using WampServer and Teacher & Students Academic Communication site which is developed by developers other than the authors. Authors have proved the efficiency of the solution depending on the test results. When request comes from different sources (illegitimate user) during a valid session the CSRFD tool successes in preventing this request to happen and end valid session. In future, the authors will address all the limitations of the proposed solution mentioned in previous.

References

[1] Mateo Martinez, CISSP, "OWASP Latem Tour Venezuela 2013", 2013 the McAfee.

[2] S. Shah, "HTML5 Top 10 Threats Stealth Attacks and Silent Exploits," BlackHat Europe, 2012.

[3] C. Raghavendran, G. N. Satish, and P. S. Varma, "Security Challenges and Attacks in Mobile Ad Hoc Networks," 2013.

[4] P. I. Singh, "Robust Security System for Critical Computers," International Journal of Information Technology and Computer Science (IJITCS), vol. 4, p. 24, 2012.

[5] Siddiqui, M.S.; Verma, D., "Cross site request forgery: A common web application weakness," Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on, vol., no., pp.538,543, 27-29 May 2011.

[6] B. Hill, "Adaptive user interface randomization as an anti-clickjacking strategy," ed: May, 2012.

[7] A. Elias-Bachrach, "CSRF: Not All Defenses Are Created Equal," in AppSec USA 2013, 2013.

[8] Shahriar, H., & Zulkernine, M. (2010, November). Client-side detection of cross-site request forgery attacks. In Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on (pp. 358-367). IEEE.

[9] B. Meshram, "Client Side CSRF Defensive Tool," International Journal of Information and Network Security (IJINS), vol. 1, pp. 171-180, 2012.

[10] R. D. Kombade and B. Meshram, "CSRF Vulnerabilities and Defensive Techniques," International Journal of Computer Network and Information Security (IJCNIS), vol. 4, p. 31, 2012.

[11] F. van der Loo, "Comparison of penetration testing tools for web applications," Master thesis, Radboud University Nijmegen, 2011. http://www. ru. nl/publish/pages/578936/frank_van_der_loo_scriptie. pdf, 2011.

[12] Jovanovic, N.; Kirda, E.; Kruegel, C., "Preventing Cross Site Request Forgery Attacks," Securecomm and Workshops, 2006 , vol., no., pp.1,10, Aug. 28 2006-Sept. 1 2006.

[13] Takesue, M., "An HTTP Extension for Secure Transfer of Confidential Data," Networking, Architecture, and Storage, 2009. NAS 2009. IEEE International Conference on , vol., no., pp.101,108, 9-11 July 2009.

[14] Shahriar, H., & Zulkernine, M. (2010, November). Client-side detection of cross-site request forgery attacks. In Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on (pp. 358-367). IEEE.

[15] Boyan Chen; Zavarsky, P.; Ruhl, R.; Lindskog, D., "A Study of the Effectiveness of CSRF Guard," Privacy, security, risk and trust (passat), 2011 ieee third international conference on and 2011 ieee third international conference on social computing (socialcom) , vol., no., pp.1269,1272, 9-11 Oct. 2011.

Authors' Profiles

Omar A. Batarfi is assistant professor in the Faculty of Computing and Information Technology at King Abdulaziz University. His research interests lie in several areas including security, e-learning, web development and distributed computing. He has a Master from the George Washington University and a PhD from Newcastle University.

Aisha M. Alshiky is a master student in IT Department at King Abdulaziz University, interested in Network Attacks and Technology Managment.

Alaa A. Almarzouqi is a master student in IT Department at King Abdulaziz University, interested in Database, Security and Artificial Intelligent.

Nora A. Farraj is a master student in IT Department at King Abdulaziz University, interested in Internet Technology, database and operating system.

