A Non-Programming Secure Protocol for Cross-Domain JavaScript Data Communication

Chi-Che.Wu, and Jen-Wen. Ding

***Abstract*—**Due to the popularity of the World Wide Web (WWW) and the rapid development of the Internet and smart phones in the recent years, the application program usage model has migrated from native applications to web applications. For program developers, it is no longer required to develop different versions of applications for different operating systems. At the same time, users do not have to install a whole lot of programs or consider upgrade issues. In other words, users can use many different kinds of software services by simply installing a browser, connecting to the Internet, and entering a web address, such as Gmail, YouTube, Google Document, Facebook, etc…

Along with the mass number of web application development, the need arises for cross-site data exchange. This paper investigates the strengths and weaknesses of the current cross-site data exchange methods, such as JSONP, YQL, and HTML5 Post-Message. The proposed new mechanism can effectively solve the problem caused by the current mechanism, avoiding common CSRF attacks.

***Keywords*—**Same-Origin Policy, Cross-Site Request, CSRF, Cross-Site Scripting, HTML5 Post-Message, Web Application

I. Introduction

DUE to the popularity of the world wide web (www) and the strong development of the internet and smart phones in the recent years, program developers have gone from desktop application program to mobile application program development, for some, even web application programs. Developers no longer develop multiple versions of the same application based on different operating systems. Now, there are an increasing number of applications that is dependent of browsers. In other words, by simply installing a browser and connecting to the internet via a web address, users can utilize the application services provided by developers.

While many benefits come with web applications for both developers and users, there are underlying risks, especially if the developer is not as competent. The web application may fall victim to hacking by XSS, CSRF, SQL injection, and Session Fixation. When the attack is successful, severity can be as simple as operation failure to loss of data, which may cause monetary losses. Understanding the severity of the aforementioned, most browsers restrict JavaScript cross-site data access, currently, to prevent XSS and CSRF attacks. The browser's same-origin policy strictly prohibits the execution of JavaScript cross-site data access.

Even though most attacks similar to XSS and CSRF can be avoided by the browser's same-origin policy, but it also causes great inconvenience for program developers. For example, a developer would need to develop a program that ensures flawless communication between Gmail and Facebook to resolve the communication failure problem caused by same-origin policy, because the entire program structure are basically three sections, our website's application, Gmail's, and Facebook's.

To resolve the problems caused by same-origin policy, make resolutions were created: JSONP, YQL, and Post-Message HTML5, etc. However, these all present to have insufficient support issues or are a burden to program developers monetarily in terms of development. With that in mind, this paper offers a whole new protocol. With this protocol, developers can put their focus on program logic and allow the browser to conduct cross-site communications, reducing the risk burden for developers.

Chi-Che.Wu is with the National Kaohsiung University of Applied Sciences , No.58, Shenzhong Rd., Yanchao Dist., Kaohsiung City 824, Taiwan (R.O.C.) (jerryclass@gmail.com , 1101405110@gm.kuas.edu.tw).

Jen-Wen. Ding. is with the National Kaohsiung University of Applied Sciences , No.58, Shenzhong Rd., Yanchao Dist., Kaohsiung City 824, Taiwan (R.O.C.) (e-mail: jwding@cc.kuas.edu.tw).

II. Related Work

This paper mainly investigated research conducted on current cross-site data request attacks in browsers and the different situations involved with cross-site limitations or the inability to limit in browsers. Lastly, investigation was done on current cross-site data request methods adapted by Google and Facebook as well as the analysis of other multiple cross-site data request methods.

A. Active Attack and Passive Attack

Due to the popularity of Internet applications in the recent years, more and more applications have migrated from traditional desktop computers to the WWW. Attack methods have also grown from traditional desktop viruses to host attacks. Attacks have been categorized into active attack and passive attack.

An attack active is defined as a targeted attack through the safety loopholes of a server, which may cause a server meltdown or data alteration, and also cause damage to personal data or the illegal obtainment of personal information, which then causes loss of user information, causing customers to lose their trust in the company.

Fig. 1 Active attack of schematic

A passive attack means an attack that doesn't connect to the server directly, but actually lure users onto a spoof website where the attack is conducted through the safety loophole of the browser used. Three commonly seen passive attacks are:

◎ Luring users to a spoof website
◎ Using the target host to embed a trap
◎ Using cross-site data request

B. Browser Same-Origin Policy

Because most attack methods, now, are done through the weaknesses found in browsers, the Sandbox concept was adapted to prevent situations as such, protecting end user computers from falling victim to browser loophole targeted attacks by hackers. The strict limitations of same-origin policy are listed below:

◎ Browser domain names must be identical
◎ Scheme used must also be congruent
◎ Most browsers also set limitations for port numbers

Even though browsers have adapted the same-origin policy, but CSRF attacks still cannot be completed avoided. Also, same-origin policy brought much burden for developers in the process of development. One example of the burden is the requirement of developing an integrated system to allow users to receive messages collaboratively from Facebook, Plurk, Google, etc., otherwise communication will fail due to the limitations applied through the same-origin policy. Owing to this problem, we investigated common resolutions that are currently offered.

1. JSONP (JavaScript Object Notation with Padding)

Due to the same-origin policy, when making cross-site requests using Javascript, the maintaining the same-origin is required. However, scr in <Script> tag does not have the same-origin policy limitation. JSONP uses this to accomplish cross-site data access.

2. YQL (Yahoo! Query Language)

YQL is a cross-site request resolution created by Yahoo. In the paragraph above, we mentioned, although JSONP can accomplish cross-site request, but it has the following two limitations.

(1) All server requests must be supported by JSONP

(2) The only data transmission method that can be used is GET. However, when we need to make a request for a third-party data, the third-party server may not allow JSONP support for security purposes, which means JSONP will not be able to execute. Through the YQL subquery method, the request can be made through JSONP to YQL's server, and then YQL's server will make the query to the third-party server. Since the third-party request made by YQL does not involve the browser, the action is not restricted by the browser's safety policy.

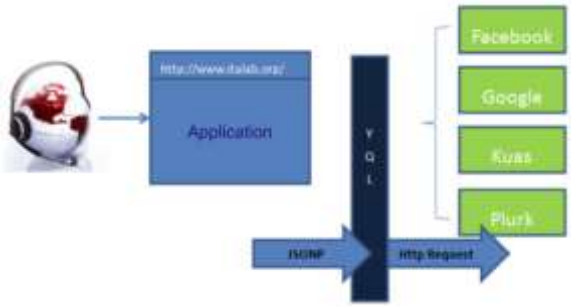

Fig. 2 YQL of schematic

III. Weaknesses of Native HTML5 Post-Message

In the previous section, we mentioned a few common cross-site request methods, and the newest method within them is HTML5 Post-Message. Currently, many larger websites support HTML5 Post-Message communication protocol, but there are underlying risks involved within the native protocol.

According to Steve Hannax, Eu Chul Richard Shinz, Devdatta Akhawex, Arman Boehmz, Prateek Saxenax, Dawn Songx, and other scholars in The Emperor's New APIs: On the (In) Secure Usage of New Client-side Primitives, currently, if the Post-Message does not apply limitations to the source and destination address, it may result in CSRF attacks by hackers. Using Facebook as an example, the source and destination authentication of Facebook's native API only authenticates the first time. It will not attempt to authenticate at second query. This attack method will be explored in this paper.[1]-[3]

Fig. 3 Post-Message First Communications

In figure 3 shows the Post-Message communication method provided by Facebook used by a third-party developer. It begins by using the embedded method to embed the iframe with an origin of Facebook into the end user's browser. Then, it sends the API obtained from Facebook and its own domain information to Facebook's iframe using Post-Message. Facebook's iframe will send the message to Facebook's server through Ajax. When Facebook's server received the message, it will authenticate the API and domain to determine the match. If they match, it will send a secret key to Facebook's iframe (4) through Ajax, and then "bundle" the destination location through Post-Message to send the secret key to italab's frame.

In figure 4 explains the steps involved with obtaining data from Facebook after logging in, the two encrypted parameters, S,K, used for query has been obtained. Therefore, during the

second query with Facebook's iframe, owing to not knowing the other side's domain (Facebook's API can be spread sporadically throughout different servers), this portion's targetOrigin is set to "*". Since Facebook's iframe will not authenticate whether the source is correct, because there are many sources, and Facebook is unable to set and catch every single query, Facebook will assume the authenticity of the user by using S,K, the two secret key.

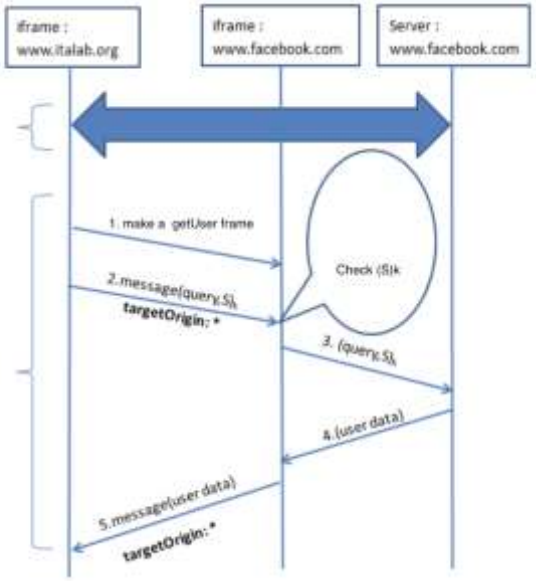

Fig. 4 Post-Message Second Communications

A. *Using loopholes to conduct CSRF attacks*

Since there the authentication for source happens only once during Facebook's Post-Message query, during and after the second query, S,K will be recognized. This setting may not seem to have many problems, but, in reality, there is the underlying risk of CSRF attacks by hackers.

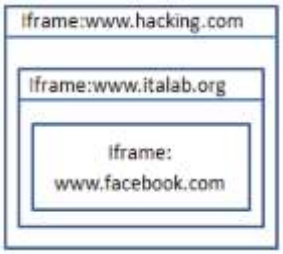

Fig. 5 A hacking attack architecture

In figure 5 states, hackers will setup a malicious phishing website and embed the phishing website within the original website (www.italab.org). This way, the phishing website will seem to be almost like the real original. Unless one looks at the web address carefully, otherwise, it is easy to not recognize the difference and fall victim to phishing.

In figure 5 explains how to utilize the loopholes within Facebook's API to conduct a CSRF attack. Due to the fact that the targetOrigin was set as *, when the user send the secret key to Facebook, the message can be received as along as the other iframes received it. So as the figures shows, when the user sends a message, the hacked iframe receives the same message. Therefore, using the same method, after the hacked iframe receives the secret message, it can send the message to Facebook's iframe. Since Facebook's iframe will not be authenticating the source, it will accept the request. This completes a CSRF attack.

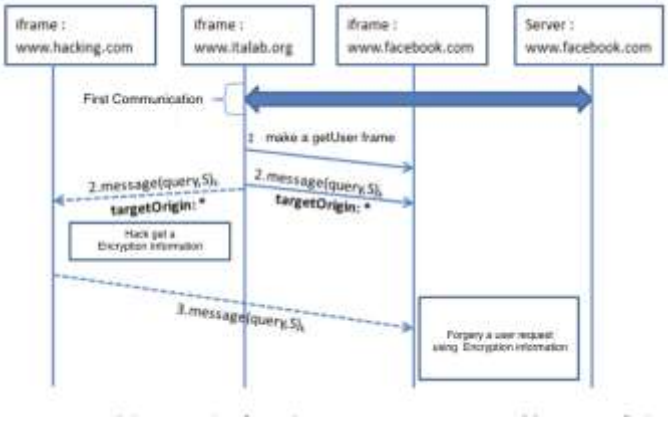

Fig 6. Conduct a CSRF attack

B. *Using loopholes to steal user information*

Using the method mentioned in 3.1, we used the method of phishing websites to embed www.italab.org. Because other than authenticating the secret key and limiting the destination domain at the first query, all queries after that will not be authenticated for destination. Since Facebook sets the targetOrigin as * when sending back user information, therefore, the hacker will receive user information as well as www.italab.org, causing loss of user information.

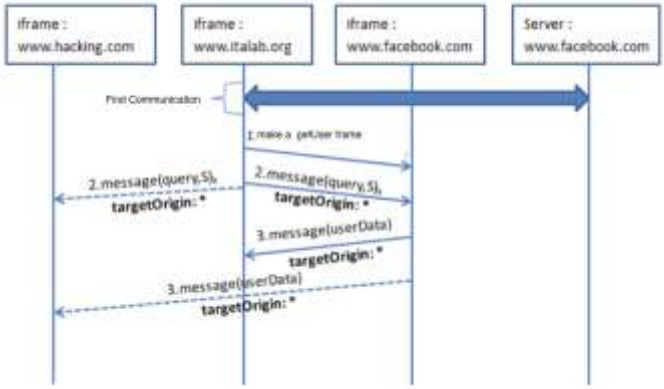

Fig. 7 Conduct a CSRF attack

In figure 7, 2~3 methods are shown. These are methods that can easily cause user information loss or CSRF attacks. However, these attacks are not based on or caused by Post-Message loopholes. The attacks are successful, partially, because targetOrigin was ignored during program writing by the developer or API service provider. In reality, it is difficult to solve because we are not sure of the targetOrigin during the developmental stage. Just like Facebook communications, the destination is unknown, thus, creating the existence of the loophole.[4]-[6]

IV. Browser Authentication Mechanism

Because native HTML5 Post-Message causes great burden for program developers during the developmental stage, it

became the main reason why the developers ignored it. The above mentioned causes security loophole problems. With that in mind, this paper presents a browser authentication mechanism that can reduce the developer's burden and increase security of the authentication mechanism.

A. Browser Authentication Procedure

Based on research findings, the same time hackers are conducting a CSRF attack, they must lure the user to the phishing website. An invisible iframe or a tag that is not limited by same-origin policy must be embedded into the phishing website in order for the CSRF attack to occur. Therefore, this research is mainly built on a communication protocol mechanism where the server will authenticate the user's login as well as verifying whether the top level domain has the same login from the initial login after receiving the query. The query will be rejected if the login information is not a match and a message prompt will be shown, requesting the user to authorize. If the user authorizes the query, communication will continue. If the user does not, then the communication will be denied. The procedure are as follows:

Step 1: User Login Stage

1. User enters login account information to gain access
2. Browser sends user login information to target website, including top level domain information.
3. Server receives the information from the browser and proceeds to authenticate. If the authentication passes, the authenticated information is recorded as well as the top level domain location for future authentication purposes.

Step 2: User Usage Stage

1. User executes website related software application
2. User calls website API information and sends relative information and top level domain location to the remote server
3. After the server receives the user request, it will authenticate the user login information and top level domain location to verify a match.
4. If the user request belongs to the same top level domain are a match to the previous, request is granted. If it is not a match, connection is denied, and a warning message stating the top level domain has been changed is sent to the user, informing possible data loss. The user will also be asked whether to authorize is request or not.
5. If the user gives authorization, the request is granted, otherwise connection will be denied.

B. Security Analysis

In this section, we will discuss whether CSRF attack or user data loss will continue to occur when authentication will continue to be conducted through the browser authentication method.

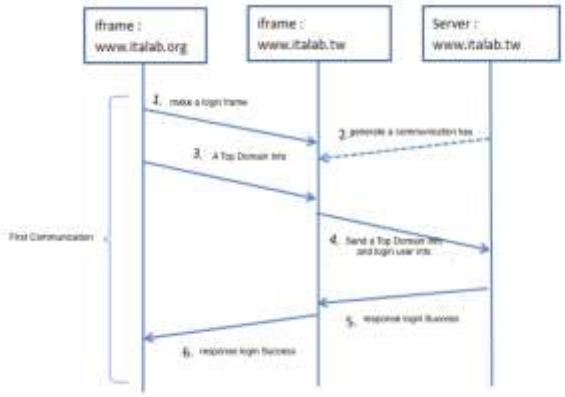

Fig. 8 Login of Flowchart

Figure 8 shows the top level domain authentication communication procedure. At the beginning, when www.italab.org attempts communication with www.italab.tw, we continue to use Post-Message as the method of communication with the other server. The difference lies in the sending of the login message. The browser will send additional top level domain information to the server of www.italab.tw. When the server receives this request, the login information and the source identity are both saved in the session.

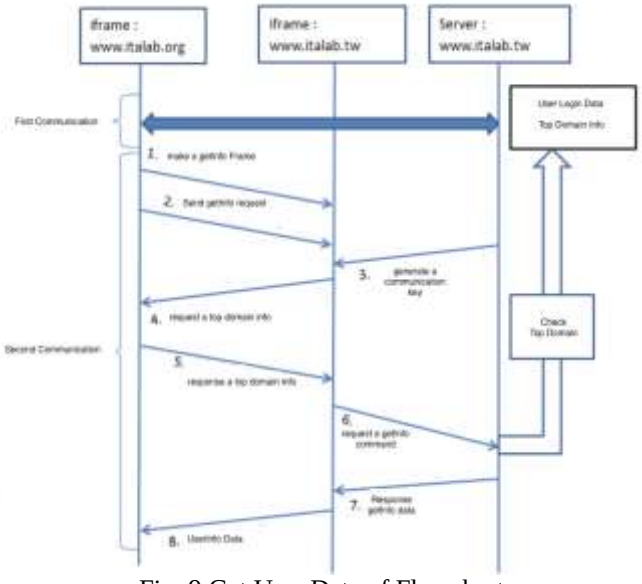

Fig. 9 Get User Data of Flowchart

After the user logs in, in every steps of the communication process, whenever www.italab.org sends a request, the browser will send an additional top-level domain message for the server to verify, preventing possible CSRF attacks. Figure above displays the communication process after the user logs in.

In this instance, because the communication was considered normal without malicious interference, therefore, the top-level domain address remained to be www.italab.org. Thus, when www.italab.org sends a request to the server, other than the request itself, it sends the additional top-level domain information together to the server. When the server receives this request, it will authenticate the top-level domain information within the request to see if it is a match with the

previously submitted domain information. If it is not a match, the server will use Ajax to communicate with www.italab.tw, sending a message prompt to the user, asking whether the communication should be granted or denied. If granted, the user will receive the requested information. Otherwise, communication will be denied.

V. Research Results

This section mainly investigates using research investigation to conduct attack testing and explore whether the method brought up in this paper can indeed prevent cross-site attacks.

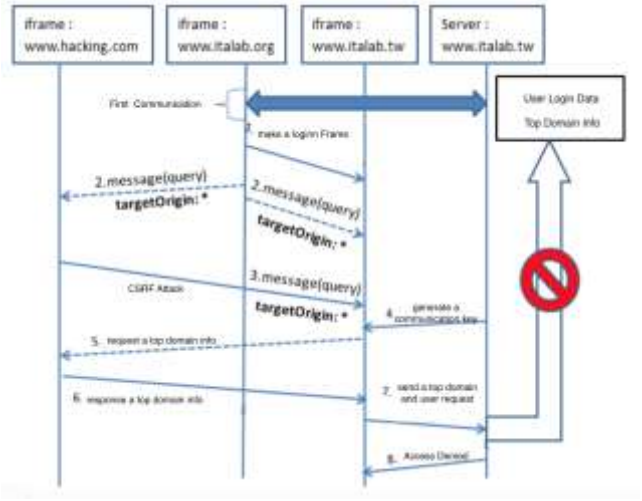

Fig. 10 Simulate CSRF Attack

As figure 10 indicates, www.hacking.com uses the embed method to embed www.italab.org into it. Since the user uses the normal website (www.italab.org) to log in and the login source web address (www.italab.org) and the login information are saved on the server.

If when a malicious or phishing website adapts (www.italab.org) to conduct a CSRF attack, even though www.hacking.com can impersonate www.italab.org to send a query to www.italab.tw, but because www.italab.tw will send a request to the top-level domain for identity confirmation at the same time the query is received, and because the malicious website (www.hacking.com) will not have the original login website's (www.italab.org) identity information, the server will deny the request based on different source. It will also user Ajax to send www.italab.com a message, indicating user source website has changed by a message prompt, asking whether the user would like to authorize the connection. If connection is authorized, then this source will be added to the server's database. If not, connection will be terminated. Based on this mechanism, most CSRF attacks will be stopped, and the message prompts will alert the user of the possible phishing website situation.

After the user logs in, the user end will request a user information iframe. However, the user had not realized he had entered a malicious website built by the hacker at the moment. Since the server end will utilize the authentication mechanism to determine whether the top-level domain source was the same as the initial login, when the server realizes the domain information has been changed, access will be denied to protect the user from malicious attack.

VI. Conclusion

In the simulation model of CSRF attack shown above, we know malicious websites need to lure users into the website before a malicious attack can take place. Therefore, as long as the server receives the query and uses the HTML5 Post-Message method to send a verification message to ensure identity to the top-level domain, top-level domain information will not be changed by the hacker. The user will also not fall victim to a CSRF attack, causing personal or server provider losses.

References

[1] Yi Wang; Zhoujun Li; Tao Guo; , "Program Slicing Stored XSS Bugs in Web Application," Theoretical Aspects of Software Engineering (TASE), 2011 Fifth International Symposium on , vol., no., pp.191-194, 29-31 Aug. 2011

[2] 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

[3] 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

[4] Jung-Ying Lai; Jain-Shing Wu; Shih-Jen Chen; Chia-Huan Wu; Chung-Huang Yang; , "Designing a Taxonomy of Web Attacks," Convergence and Hybrid Information Technology, 2008. ICHIT '08. International Conference on , vol., no., pp.278-282, 28-30 Aug. 2008

[5] Siddavatam, I.; Gadge, J.; , "Comprehensive test mechanism to detect attack on Web Services," Networks, 2008. ICON 2008. 16th IEEE International Conference on , vol., no., pp.1-6, 12-14 Dec. 2008
doi: 10.1109/ICON.2008.4772620

[6] Razzaq, A.; Hur, A.; Haider, N.; Ahmad, F.; , "Multi-Layered Defense against Web Application Attacks," Information Technology: New Generations, 2009. ITNG '09. Sixth International Conference on , vol., no., pp.492-497, 27-29 April 2009