WEB APPLICATION SECURITY - CROSS-SITE REQUEST FORGERY ATTACKS

RadhaRani Sankuru
Pursuing M.tech (CSE),
Vasireddy Venkatadri Institute of Technology, Affiliated To JNTUK,
Nambur, Guntur, Kakinada, AP, India.

MadhuBabu Janjanam
Asst. Professor, Department of Computer Science Engineering,
Vasireddy Venkatadri Institute of Technology, Affiliated To JNTUK,
Nambur, Guntur, Kakinada, AP, India.

Abstract—Cross-Site Request Forgery (CSRF) is an attack outlined in the OWASP Top 10 whereby a malicious website will send a request to a web application that a user is already authenticated against from a different website. This way an attacker can access functionality in a target web application via the victim's already authenticated browser. Targets include web applications like social media, in browser email clients, online banking, and web interfaces for network devices.

As browser holds valid session information of each request, a browser is the first place to look for attack symptoms and take actions. Current client side detection methods allow performing request to a trusted website by white listed third party websites. These approaches are not effective if policies are specified incorrectly, they do not focus on all the requests and cross check of response content type.

To overcome these limitations, we acquaint a client side detection mechanism for the CSRF attack. Our approach relies on concept of a unique CSRF token which tends to change for each and every request. We can do that by using a unique number generator to generate the token. Then we try to match the token in the user's session data and invalidate it when we see it as a match or no token at all. This makes the token a used once. This protects us against repeated attacks. Moreover to overcome an attacker's attempt to circumvent form visibility checking, we compare the response content type of a suspected request with the expected content type.

The current approach detects CSRF attacks through HTML form submissions and other source of requests that might cause program state retrieval or modification which is compatible to latest versions of popular browsers such as IE, Firefox, and Chrome.

As proposed approach checks all the requests which might change program state and compatible to popular browsers this approach can reduce the CSRF attacks by detecting the significant number of attack requests, hence our evaluation results indicate that our approach can detect most of the common form of CSRF attacks.

Keywords- Browser security, client-side attack detection, cross site request forgery, cross-site scripting, OWASP.

I. INTRODUCTION

CSRF is web application vulnerability where a malicious web site can make legitimate requests to a vulnerable web site under the disguise of a logged-in user without that user's knowledge. This vulnerability has been rated as one of OWASP (Open Web Application Security Project) Top 10 vulnerabilities – A5, and Common Weakness Enumeration (CWE – 352).

Cross Site Request Forgery (also known as XSRF, CSRF, and Cross Site Reference Forgery) works by exploiting the trust that a site has for the user. Site tasks are usually linked to specific URLs allowing specific actions to be performed when requested. If a user is logged into the site and an attacker tricks their browser into making a request to one of these task URLs, then the task is performed and logged as the logged in user. Typically an attacker will embed malicious HTML or JavaScript code into an email or website to request a specific 'task URL' which executes without the users knowledge, either directly or by utilizing a Cross-site Scripting Flaw. These sorts of attacks are fairly difficult to detect potentially leaving a user debating with the website/company as to whether or not the stocks bought the day before was initiated by the user after the price plummeted.

We develop a detection framework for CSRF attacks by using a unique identifier named CSRF token for each and every request. It is independent of any cross-origin policy. Moreover, the approach can detect attacks where requests contain partial information or request with no query strings and which intend to change server side program states.

We can find example of CSRF attack in the below code snippets in java program in Figure 1 (jsp a client side view) and Figure 2 (request processing server side content). Let us assume that a user is logged on to a site (www.abc.com) that has product information to be sent to user via email. The send details page includes a contact email address with value as user@abc.com. The client side interface (Figure 1) provides a view (sendDetails.jsp) to set the new email address of a logged on user to send the product details. A new email address provided by a user (Line 5) is updated by the server side code (i.e., the ProductDetailsController.java at Line 3). The request of the email address change is sent to ProductDetailsController.java by a hidden field (toEmail) at Line 4.

```
1. <HTML>
2. <BODY>
3. <FORM action = “prodDetails” method = “POST”>
4. <INPUT type = “hidden” name = “action” value = “toEmail”>
5. <INPUT type = “text” name = “emailAddr” value = “”>
6. <INPUT type = “submit” value = “SendProductDetails”>
7. </FORM>
8. <BODY>
9. </HTML>
```

Figure 1. Client side code (sendDetails.jsp)

```
1.if (session.getAttribute(“UserInfo”)!=null){
2.if(request.getParameter(“action”) !=null && (String)
request.getParameter(“action”).equalsIgnoreCase(“toEmail”)){
3.if(request.getParameter(“emailAddr”) !=null){
4.    sendProductDetails((String) request.getParameter(“emailAddr”));
5.}
6.}else{
7.          log.error(“Invalid session!”);
8.        response.sendRedirect(“error.jsp”);
```

Figure 2. Server side code (ProductDetailsController.java)

At server side in ProductDetailsController.java checks whether the user session is valid, if not system will redirect to the error page and if yes, the request is processed and update system state by sending product details to the user’s new email Id. If user request sends the email then the HTTP request becomes http://www.abc.com/sendDetails?action= toEmail & emailAddr=user2@abc.com.

Let us assume that the user is logged on to www.abc.com as well as visiting another site that contains a hyperlink http://www.abc.com/ sendDetails?action= toEmail & emailAddr=attacker@abc.com. If the user clicks on the link, the contact email address is changed to attacker@abc.com. The user becomes a victim of a reflected CSRF attack.

II DETECT CSRF ATTACK

This paper mainly deals with reflected CSRF attack, the attack payloads reside in third party websites that are vulnerable to XSS. Some of the approaches have been dealt with a unique CSRF token which needs the change in server side program code to implement. As this approach of handling CSRF attacks wraps the client side code we can avoid attacks earlier.

This approach uses a unique CSRF token for each and every request by considering different types of requests specified. Whenever an authorized user requests the website, each valid request is appended with unique CSRF token and this token will be stored in session data and this valid request will be processed at server side.

When CSRF attack request passes the request to the website, as an attacker is not aware of CSRF token the request will not be appended with token, hence in the token verification process the request will be rejected and not processed. If the attacker guesses the token, in the token validation process the request token will be validated for the format, if not matches the request will be rejected and not processed.

If the request matches the format and matches with the set of stored CSRF tokens then the request will be rejected hence it protects us against replay attacks (since the number is only valid on the first submission), In replay attacks the attacker watch the original request so they can steal the request payload. Then, they can re-submit the request while altering form data to do what they want. If the request token is valid and not matching with existing tokens then request will be cross checked for the response content type with the expected values, if matching then request will be processed else not.

The overview of CSRF Detection method is shown in Figure 3 and Figure 4.

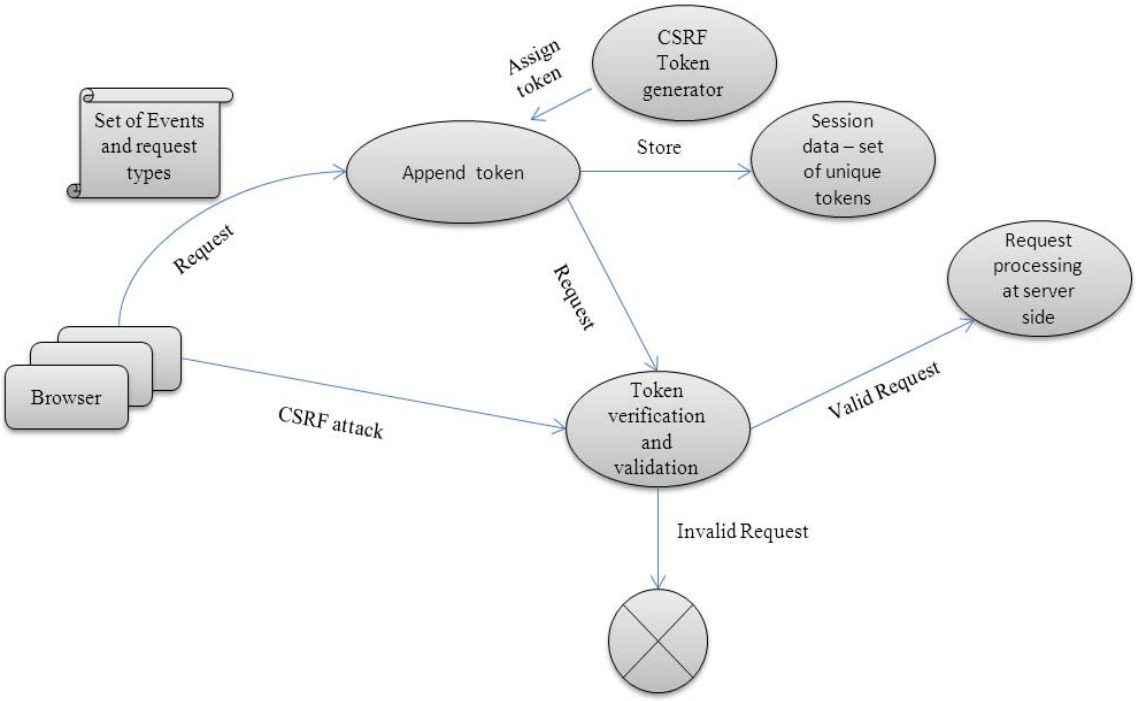

Figure 3. Overview of Cross Site Request Forgery Attack Detection method.

CSRF UNIQUE TOKEN GENERATOR

This module intercepts each and every request raised by the authenticated user and appends the unique CSRF token with a unique format. This includes most possible ways of raising the request example, through links or URLs, Ajax call, form submission by buttons, image links etc. The unique format includes a combination of session id and request time as timestamp whenever request sent by the client. This combination makes the CSRF token unique for each and every request.

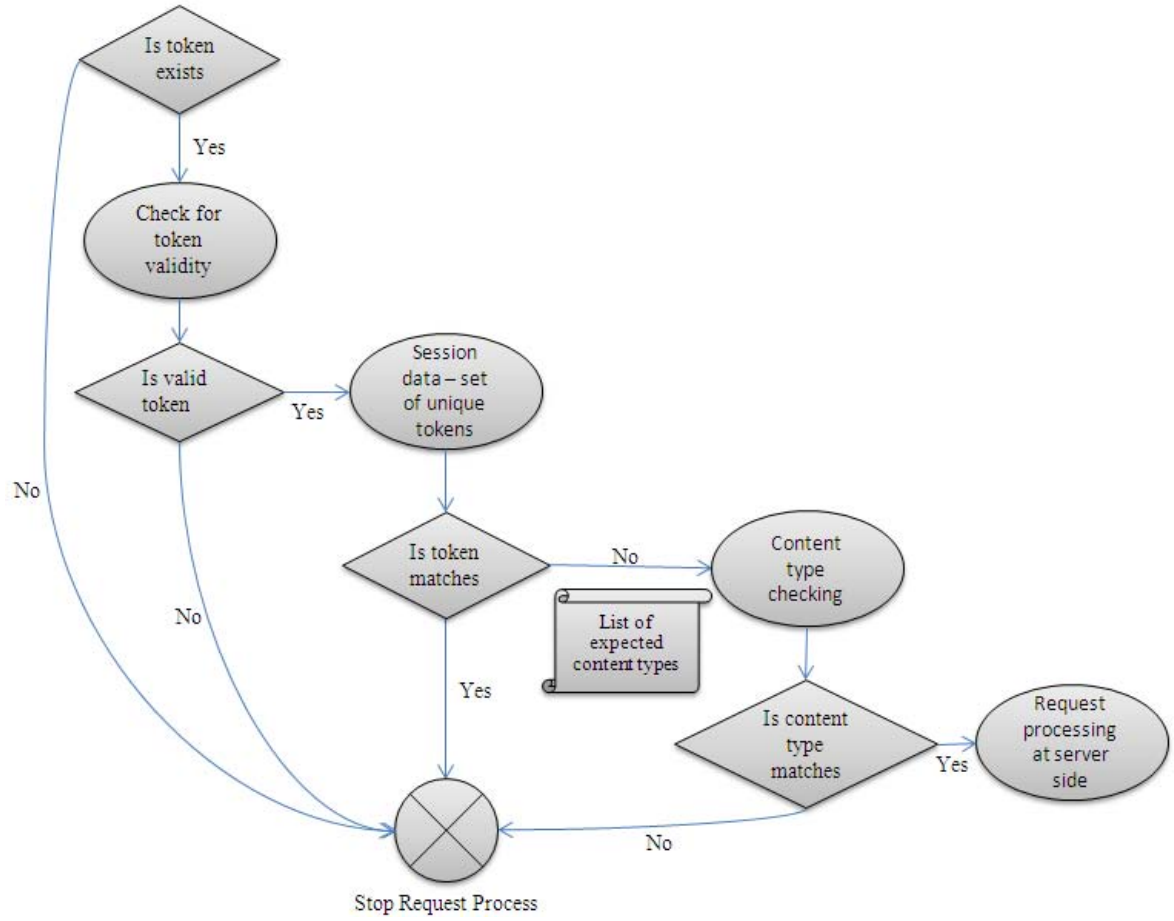

Figure 4. CSRF Token verification and validation process.

TOKEN VERIFICATION AND VALIDATION

When CSRF attack request passes the request to the website, as an attacker is not aware of CSRF token the request will not be appended with token, hence in the token verification process the request will be rejected and not processed. If the attacker guesses the token, in the token validation process the request token will be validated for the format, if not matches the request will be rejected and not processed.

If the request matches the format and matches with the set of stored CSRF tokens then the request will be rejected hence it protects us against replay attacks (since the number is only valid on the first submission), In replay attacks the attacker watch the original request so they can steal the request payload. Then, they can re-submit the request while altering form data to do what they want.

If the request token is valid and not matching with existing tokens then request will be cross checked for the response content type with the expected values, if matching then request will be processed else not.

CONTENT TYPE CHECK

It consists of the mapping between expected HTML tag attributes containing requests and the list of expected content types and their related URL attributes using JSON.

III EVALUATION RESULTS

Table I. shows a summary of the total number of requests raised for each of the programs, the number of GET and POST requests, the number of requests that retrieve information from program session or database, the Number of requests that modify program session (e.g., logout) or database (e.g., add a project), and the average number of hidden and non-hidden parameters.

TABLE I. TEST SUITE

Program name	Total request	GET	POST	Retrieve	Modify	Avg. hidden	Avg. non hidden
Java Program1	50	30	20	27	23	4	9
Java Program2	40	10	30	10	30	2	8

RESULTS:

Table I shows the summary of the test attacks for the two java programs. Columns 2-7 show the detection result for the three test modes (window and no form, window and form without value, and window and form with value). For each test mode, we note that our approach generates warnings and stops all the attacks (denoted as x/y, where x is the number of attack test cases and y is the number of test cases detected) for program state retrieval (ret.) and modification (mod.) related URLs.

Requests through links: We keep a webpage of a program under test that has hyperlinks to raise the request to the server (e.g., pages having links to get results, which modifies the state). To emulate reflected CSRF attacks, all test scripts (or attack web pages) are placed in a separate web server. We visit attack web pages and observe whether our approach generates any warning or not.

Requests through Ajax calls: In this mode, we traverse a program to reach a randomly chosen page having Ajax call. We provide input to form fields and we emulate reflected CSRF attacks by visiting the attack web pages from a different window.

Requests through form fields or button: In this mode, we first visit pages that contain forms. Moreover, these pages provide the options to add, edit, or remove information. We provide inputs in forms for modification related operations (e.g., add) that match with attack requests appropriately. Note that this test mode excludes all the retrieval related requests.

TABLE II. CSRF ATTACK DETECTION SUMMARY

Program name	Requests through links		Requests through Ajax calls		Requests through form fields or button	
	Ret.	***Mod.***	***Ret.***	***Mod.***	***Ret.***	***Mod.***
Java Program1	27/27	23/23	27/27	23/23	N/A	23/23
Java Program2	10/10	30/30	10/10	30/30	N/A	30/30

TABLE III. COMPARISON SUMMARY OF CSRF ATTACK DETECTION WORKS

Work	Reflected CSRF	Stored CSRF	Response content type check	Deployment location	Modification of response	Information added/removed	GET	POST
Johns and others. [1]	Yes	No	No	Client	URL	Add random token	Yes	Yes
Barth and others. [2]	Yes	No	No	Client	None	Add HTTP origin header	No	Yes
Mao and others. [3]	Yes	No	No	Client	None	remove cookie	Yes	Yes
Maes and others. [4]	Yes	No	No	Client	None	remove cookie	Yes	Yes
CSRFGuard [5]	Yes	No	No	Server	URL	Add unique token	Yes	Yes
Zeller and others. [6]	Yes	No	No	Client and Server	Form field and cookie	Add random number	No	Yes
Ryck and others. [7]	Yes	Yes	No	Client	None	remove cookie	Yes	Yes
Jovanovic and others.[8]	Yes	No	No	Server	URL	Add unique token	Yes	Yes
Jayaraman and others.[9]	Yes	Yes	No	Server	None	None	Yes	Yes
Our work	**Yes**	**No**	**Yes**	**Client**	**None**	**Add unique token**	**Yes**	**Yes**

The approach [1] initially intercepts an HTTP response and identifies active URL links, followed by adding random tokens and saving the modified URLs. Next, if an HTTP request is intercepted, the saved URLs are matched with the current URL. If no match is found, an attack is detected. Otherwise, they remove cookie or sensitive information present in a request. This forces a user to re-login a website.

Several server side approaches are applied to detect CSRF attacks. The CSRFGuard [5] adds filters in web servers that host programs vulnerable to CSRF attacks. A filter maps between resources (e.g., a server side web page) and the corresponding code that intercepts HTTP requests to detect CSRF attacks. The response page is searched for HTML forms and links, and inserted with appropriate unique token parameter values and stored in a session table. For a new request, the value of a token parameter is compared with the saved token. If there is no match, the request is considered as a CSRF attack.

The approach [9], encode the behavior of a web application by deterministic finite automaton (DFA). In DFA, a state is a server-side script and state transitions occur through HTTP requests. The hyperlinks and forms contained in each web page determine the set of valid requests that a user may issue in that state. A CSRF attack is a deviation between a set of known valid requests and an actual request not applicable for a state.

IV. CONCLUSIONS AND FUTURE WORK

Current client side detection methods allow performing request to a trusted website by white listed third party websites. These approaches are not effective if policies are specified incorrectly, they do not focus on all the requests and cross check of response content type. Some of the approaches have been dealt with a unique CSRF token which needs the change in server side program code to implement. As this approach of handling CSRF attacks wraps the client side code we can avoid attacks earlier.

This paper proposes the detection of CSRF attacks is a mechanism to intercept a suspected each and every request using a unique identifier named CSRF token. Moreover, the approach can detect attacks where requests contain partial information or request with no query strings and which intend to change server side program states.

The experimental results show that the approach suffers from zero false positive and negative rates for attack requests that retrieve or modify program states.

Moreover, our approach enables a user to specify detection policy based on the user's needs. We also contribute to the development of a test suite to perform the evaluation using several real world vulnerable programs.

The current approach considers HTML form submissions as the primary source of state modification or retrieval to server side programs by different means of approaches. Thus, we plan to detect CSRF attacks through other source of requests that might cause program state retrieval or modification.

Our future work includes detection of complex attacks and the evaluation of performance penalties for legitimate requests.

V. REFERENCES

[1] Category: OWASP CSRFGuard Project, http://www.owasp.org/index.php/Category:OWASP_CSRFGuard_Project
[2] Threat Risk Modeling, http://www.owasp.org/ index.php/Threat_Risk_Modeling
[3] Lin, X.L., Zavarsky, P., Ruhl, R., Lindskog, D.: Threat Modeling for CSRF Attacks. In: the 2009 International Conference on Computational Science and Engineering, Vancouver (2009) In: BruCON Security Conference 2010, Brussels (2010)
[4] Balduzzi, M., Egele, M., Kirda, E., Balzarotti, D., Kruegel, C.: A Solution for the Automated Detection of Clickjacking Attacks. In: 5th ACM Symposium on Information, Computer and Communications Security, New York (2010)
[5] Mao, Z.Q., Li, N.H., Molloy, L.: Defeating Cross-Site Request Forgery Attacks with Brower-Enforced Authenticity Protection. In: International Conference on Finance Cryptography and Data Security, (2009)
[6] CSRF Guard Testing, http://pentesterconfessions.blogspot.com/2008/06/csrfguard-testing_05.html
[7] HTTP 1.1, http://tools.ietf.org/html/rfc2068
[8] Vialy Shmatikov. Web browser security, http://www.cs.utexas.edu/~shmat/courses/cs380s/12browser.ppt
[9] CSRFGuard 3 Configuration, http://www.owasp.org/index.php/CSRFGuard _3_Configuration
[10] Ryck, P.D., Desmet, L., Heyman, T., Piessens, F., Joosen, W. : CsFire:Transparent client-side mitigation of malicious cross-domain requests.
[11] A look at the 'Clikcjacking' Web Attack and Why You Should Worry, http://www.webmonkey.com/2008/10/a_look_at_the_clickjacking web_attack_and_why_you_should_worry/
[12] Son, S.: Prevent Cross-site Request Forgery: PCRF.
[13] CSRF Guard 3 token injection, http://www.owasp.org/index.php/CSRFGuard_3_Token_Injection