CROSS SITE REQUEST FORGERY PREVENTION USING BROWSER FINGERPRINTING

Jaya Gupta and Suneeta Gola

College of Science & Engineering, Department of Computer Engineering, India

ABSTRACT:

The internet is now an essential component of every-one’s life. But, as we are getting more dependent on the internet, in the same ratio, the attackers are also more interested in exploiting websites and other web related information systems. CSRF (Cross Site Request Forgery) has emerged as the powerful warning to web world. The reason behind is, HTTP protocol is a state-less protocol and a harmful website can replicate the victim user’s browser to send the invalid requests to a trusted site. Cross Site Request Forgery (CSRF) attack is one of the most common risk, but this is also not much familiar to web developers, that’s why it can be forged easily in web based systems.

Keywords - Cross Site Request Forgery, web application vulnerabilities.

[1] INTRODUCTION

Cross Site Request Forgery [1] (abbreviated as XSRF or CSRF, sometimes also called “Session Riding”), denotes a relatively new class of attack against web application users. CSRF [2] is an attack that coerces an unaware user to send unauthorized requests to any web domain by interacting with a seemingly unrelated entity on malicious website. Therefore, CSRF impersonates a legitimate user and exploits the trust that the website has for him/her. CSRF has listed among the top 10 web vulnerabilities. In practice, CSRF [3] attacks are mounted on the top of XSS (Cross Site Scripting) vulnerabilities. A Cross Site Request Forgery is a Confused Deputy Attack against web browser. That means, it is web program which can easily fooled by any other party which is misusing its authority.

[2] OVERVIEW OF CSRF ATTACKS

A Cross Site Request Forgery is a type of attack that forces an end user to execute malicious actions on the web applications on which they are currently authenticated logged in. Specifically, the CSRF is involved with state changing requests but not in the theft of data. The attacker
does not have any media to see the forged request’s response. But with the help of email or via link, attacker can trick the authenticated user into executing actions of attacker’s choosing. If the victim is normal user (not a web developer or analyst), then a successful CSRF attack can forge the user to perform the state changing requests like transferring funds, exchange of email addresses etc. We will discuss more about Cross Site Request Forgery in further sections. Section 3 illustrated the literature review and Section 4 illustrated the conclusion on the basis of previous researches.

[3] LITERATURE REVIEW

In previous years, there is lot of research work has been done in this field. In the previous researches, researchers had proposed techniques and solutions to prevent and defense against the Cross Site Request Forgery.

[3.1] PREVENTION AND DEFENSE TECHNIQUES AGAINST CSRF

In 2006 [1], [Jovanovic, N.; Kirda, E.; Kruegel, C.] proposed the Proxy based solution. In this approach, solution to the problem was to decouple the necessary security mechanism from the application and to provide a separate module that can be plugged into existing systems with minimal effort. More precisely, they proposed a proxy that was placed on the server side between the web server and the target application. This proxy was very well sufficient to check and change the requests sent by client and the replies to itself extend applications by using the shared secret technique. In particular, the proxy had to ensure that replies to an authenticated user had to modified in such a way that future requests originating from (through hyperlinks and forms) should contain a valid token, and take countermeasures against requests from authenticated users that did not contain valid token. By decoupling the proxy from the actual application, the XSRF protection could be offered transparently for all applications.

In 2010 [2], [Alexenko, T.; Jenne, M.; Roy, S.D.; Wenjun Zeng] proposed the Defense mechanism to prevent CSRF. In this approach, Defense mechanism included 2 techniques for the solution.

1. Referrer Privacy Guard

The Referrer Privacy Guard revealed how a constant flow of random HTTP requests could mess up the browsing history at the server side, thus preventing infiltrators from getting access to user browsing trends.

2. Detection and Discouragement

In this section, the focus was on how to detect CSRF signatures in web pages and stop it before commencement. The defense attribute first verified the Client side code before each and every page load and found the CSRF attack involved.
In 2010 [3], [Shahriar, H.; Zulkernine, M.] proposed CSRF Attack detection approach that was divided in multiple sections.

Section A: Attack detection framework

In the section they had assumed that a browser could have multiple windows. A trusted website could be viewed by a user in window after performing an authentication process and the session information was saved in the browser. In this section the following processes were followed.

1. Request Checker
2. Window and form checker
3. Request Differentiator
4. Attack Detection policy
5. Attack handler module

Section B: Visibility checking

The proposed notion of visibility relies on examining windows containing web pages and forms presented in a browser. If a request was GET type, they checked whether it contained any query string or not. If no query string was present, no need to examine it further. However, if a query string was present, then tokenize the string to identify the set of parameters and values. And related the extracted parameters and values with a webpage containing forms with similar fields and values. Note that the form action or target field value should match with the resource file of the suspected request. While examining a webpage, two possible scenarios might arise. These were discussed below:

1. Window and no form
2. Window and form

Section C: Content Checking

Content checking relied on the matching of the response of a suspected request with the expected response. A suspected request often resided as part of an HTML tag attribute value or within. A response page might contain various types of elements (static HTML, JavaScript, and Style Sheets). As a result, they relied on the content type of a webpage to differentiate an attack request from a user initiated request based on the identified tag that contained the request. The content type was often specified in the META tag of a page and was accessible from the response header. After that they discussed the comparison of the expected and the actual content type and how to launch a suspected request in next 2 subsections.

1. Comparison between an expected and an actual response content type.
2. Suspected request modification

Section D: Attack detection coverage and attribute checks

The proposed approach was able to detect a variety of CSRF attacks. Some non-exhaustive CSRF examples were highlighted and related with the checks to detect them.

1. Visible form with field and no value
2. Invisible form with field and value
3. Static/Dynamic HTML tag and URL at tribute
4. Program state retrieval or modification
5. Pre- or post-rendering

In 2011 [7], [Boyan Chen; Zavarsky, P.; Ruhl, R.; Lindskog, D.,] described the known CSRF vulnerabilities. Then they illustrated the working and risks of CSRF Guard. CSRF Guard was verifying the integrity of HTTP requests by inserting a special security token to every active HTTP session established among the authenticated client and the web server. Essentially, the CSRF Guard was doing the filtration of the requests coming in. It was executing following functionalities.

1. Inserted a token to the defined preserved resource.
2. Did the verification of the token when the preserved resource gets requested. The token origination and certification was used to give the protection against the CSRF attack.

In 2011 [6], [Siddiqui, M.S.; Verma, D.] suggested approach to protect against CSRF attacks by using some or all of these:

1. **Use of random tokens**
   To use random tokens each time with a form submission could make very difficult for the attacker to guess the next random pattern to fill in the URL.

2. **Need to Use post method in form instead of Get**
   Get and Post are the 2 methods of form submission. Post Method was secure for form submission. In Get method anyone could see the variables and values in URL as a query strings.

3. **Limiting the lifetime of authentication cookies**
   Limit the lifetime to a short period of time. If user was going on other website then the cookies were expired after a short period of time. If the attacker was trying to send any HTTP request to user which he was able to know and he would not fill the password again.

4. **Damage limitation**
   Damage limitation involved those steps which reduced the damage from CSRF. For example if an attacker did manage to perform CSRF on a website then any action done by him was required an authentication every time to limit the damage.

5. **Force user to use your form**
   It was forcing user every time to use the form of website. Use of hidden fields was helpful for this purpose. But this way of protection was easy to bypass.
In 2013 [8], [Yin-Chang Sung; Cho, M.C.Y.; Chi-Wei Wang; Chia-Wei Hsu; Shieh, S.W.] suggested the approach to prevent the CSRF attack, they proposed a labeling mechanism called Content Box; the Content Box consisted of a labeling function and UCC quarantine policies. The labeling function was used to isolate the UCCs, while the UCC quarantine policy enforces propagation rules for the labeled UCCs. The CSRF attack could be prevented using the Content Box when an untrusted UCC try to access a service that contains sensitive/private information.

The main idea was to divide the content into 2 different types. One was called the “trusted contents”; these contents were created by the web server administrator or the content viewer/user. Since these contents were created by the rightful owner, it was that the scripts within the contents were free from the CSRF attack. The other type was called the “untrusted contents” which were created by other users. Since these contents were provided by users other than the rightful owner, the scripts within these contents might cause the CSRF attack. It was important to differentiate the contents of the webpage since the client browser always trusted the contents of a web page even if the authors of the contents were not trusted by the client. In Content Box, they intended to distinguish the untrusted contents and prohibiting the untrusted contents from accessing web services that contain sensitive data.

Initially in web, UCC was the source of the CSRF attack problem. However, most UCCs were harmless providing that if it was created by the current client. This kind of UCC should be classified into trusted contents since the CSRF attack rarely happened when both of the attacker and the victim were identical. Labeling was used to differentiate the contents and ensured that every HTTP request was labeled, provided that the label cannot be disrupted by the client browser. In addition to labeling the contents of a web page, an access control mechanism was required to patrol the accesses of web services.

1. Trusted label had the freedom to access the contents with trusted or untrusted label.
2. Untrusted label could only access the contents with untrusted label.

Once the contents with trusted label were contaminated by untrusted label, its label became untrusted.

[4] CONCLUSION

At the end, the proposed approaches have some conclusions and limitations. These conclusions are described below.

[4.1] A Proxy Based Solution

The concept presented in this approach relied on the assumption that all request triggers (such as hyperlinks and action attributes of forms) were directly available in the output generated by the target application. If that was not the case, reply processing missed certain request triggers, which could result in subsequent false XSRF alarm.
[4.2] Window and Form Content Checking

In this approach, it proposed the detection of CSRF attacks with the notion of visibility and content checking of suspected requests. This approach was not relied on cross-origin policy or server side program states. But it did not investigate how to adjust policy to correctly detect CSRF attacks that exploit the cases where servers incorrectly sets response types or did not set the response type at all.

[4.3] CSRF Guard

CSRF did not secure the application against Cross Site Scripting (XSS) vulnerabilities and Session Hijacking.

[5] FUTURE WORK

As we have seen that in this field there have been so many researches are already taken places. In future, we will work on the detection of Cross Site Request Forgery using Browser Fingerprinting. Here, we will give brief introduction about browser fingerprinting. The Browser fingerprinting is an information collected about remote computing device for identification purpose. Basic web browser configuration information can be collected by the various parameters possible.

REFERENCES