A review on Cross Site Request Forgery Prevention System

Ms. Diksha P. Meshram
Computer Science and Engineering
Jhulelal institution of technology
Lonara, Nagpur

Ms. Nisha Balani
Assistant Professor
Computer Science and Engineering
Jhulelal institution of technology
Lonara, Nagpur

Abstract

The web has become an indispensable part of our lives. Unfortunately, as our dependency on the web increases, so does the interest of attackers in exploiting web applications and web-based information systems. Previous work in the field of web application security has mainly focused on the mitigation of Cross Site Scripting and SQL injection attacks. In contrast, Cross Site Request Forgery attacks have not received much attention. In an attack, the trust of a web application in its authenticated users is exploited by letting the attacker make arbitrary HTTP requests on behalf of a victim user. The problem is that web applications typically act upon such requests without verifying that the performed actions are indeed intentional. Because is a relatively new security problem, it is largely unknown by web application developers. As a result, there exist many web applications that are vulnerable to attacks. Unfortunately, existing mitigation approaches are time-consuming and error-prone, as they require manual effort to integrate defense techniques into existing systems. In this paper, we present a solution that provides a completely automatic protection from attacks. More precisely, our approach is based on a server-side proxy that detects and prevents attacks in a way that is transparent to users as well as to the web application itself. We provide experimental results that demonstrate that we can use our prototype to secure a number of popular open-source web applications, without negatively affecting their behavior.

Keywords: Cross Site, Forgery attacks, HTTP, web application.

Introduction

Now-a-days, web users are increasing in manifolds, at the same time attackers also increase in proportionately. So the necessity of security in accessing web is a must for secure organizations, defense personals and financial bank those interact with public. In 2010, Open Web Application Security Project [10] reported the following most critical web application security vulnerabilities that are been exploited Cross site request forgery attack (CSRF). This attack is severe vulnerability in web applications. CSRF vulnerabilities on the Internet. The CSRF attacks are typically as powerful as a user, i.e. any action that the user can perform can also be performed by an attacker using a CSRF attack. Consequently, the more power a site gives a user, the more serious are the possible CSRF attacks. For example, if the victim account has administrator rights, this can compromise the entire web application. Customers are provided with safe web services and too they are protected from many web threats. The web has become an indispensable part of our life. Unfortunately, as our dependency on the web increases, so does the interest of attackers in exploiting web applications and web-based information. There are more number of attacks which exploits the web application and integrity of the web users. By using the web browser, one can access webmail’s, online banking, community websites, search engines, and specific business applications for each sector, etc. from the private network or from the Internet. They may contain sensitive information and it required an authentication.

LITERATURE SURVEY

The detection of web-based attacks has received considerable attention because of the increasingly critical role that web-based services are playing on the Internet. This includes web application
firewalls [19] to protect applications from malicious requests as well as intrusion detection systems that attempt to identify attacks against web servers and their applications [1, 9]. Also, code analysis tools were proposed that check applications for the existence of bugs that can lead to security vulnerabilities [4, 7]. In particular, cross site scripting (XSS) attacks have received much interest, and both server-side and client-side solutions were proposed. For example, in [3], the use of a variety of software-testing techniques (including dynamic analysis, black-box testing, fault injection and behavior monitoring) are suggested to identify XSS vulnerabilities. Alternatively, dynamic techniques on the server side can be used to track non-validated user input while it is processed by the application. This can help to detect and mitigate XSS flaws. Finally, in previous work, we implemented a client-side solution [8] to protect users from XSS attempts. Unfortunately, these solutions cannot be applied to the problem of cross site request forgery, because XSRF attacks are not due to input validation problems. The general class of cross site request forgery (XSRF) attacks was first introduced by Peter W. in a posting to the BugTraq mailing list, and has since been picked up by web application developers. However, it appears to be a little known problem in the academic community and, as a result, has only received little attention. The mitigation mechanisms for XSRF that were proposed so far either provide only partial protection such as replacing GET requests by POST requests, or relying on the information in the Referer header of HTTP requests or require significant modifications to each individual web application that should be protected when embedding shared secrets into the application’s output. Our solution, on the other hand, attempts to retain the advantage of a solution based on shared secrets.

**MOTIVATION**

Cross site request forgery (CSRF; also known as XSRF or hostile linking) is a class of attack that affects web based applications with a predictable structure for invocation1. This class of attack has in some form been known about and exploited since before the turn of the millennium. The CSRF name was given to them by Peter Watkins in a June 2001 posting to the Bugtraq mailing list. CSRF flaws exist in web applications with a predictable action structure and which use cookies, browser authentication or client side certificates to authenticate users. The basic idea of CSRF is simple: an attacker tricks the user into performing an action of the attacker’s choosing by directing the victim’s actions on the target application with a link or other content. This is easiest to understand in the example of a HTTP GET. For example, the link http://www.google.com/search?q=iSEC+Partners causes anyone who clicks it to search Google for “iSEC Partners”. This is both harmless and by design. But a link like http://www.iseccpartners.com/EditProfile?action=set&key=emailAddress&value=evil@iseccpartners.com could tell an application which authenticated users only by cookie, browser authentication or certificate to edit a user’s profile and change their email address. Links can be easily obfuscated so they appear to go elsewhere, and to conceal words that would disclose their actual function. CSRF attacks effect applications that use either HTTP GET or HTTP POST to call their actions, although actions invoked with HTTP GET are often easier to exploit.

**PROBLEM STATEMENT**

Cross site scripting (XSS) vulnerabilities, CSRF vulnerabilities can be divided into two major categories: stored and reflected. A stored CSRF vulnerability is one where the attacker can use the application itself to provide the victim the exploit link or other content which directs the victim’s browser back into the application, and causes attacker controlled actions to be executed as the victim. Stored CSRF vulnerabilities are more likely to succeed, since the user who receives the exploit content is almost certainly currently authenticated to perform actions. Stored CSRF vulnerabilities also have a more obvious trail, which may lead back to the attacker. In a reflected CSRF vulnerability the attacker uses a system outside the application to expose the victim to the exploit link or content.
This can be done using a blog, an email message, an instant message, a message board posting, or even a flyer posted in a public place with an URL that a victim types in. Reflected CSRF attacks will frequently fail, as users may not be currently logged into the target system when the exploits are tried. The trail from a reflected CSRF attack may be under the control of the attacker, however, and could be deleted once the exploit was completed.

**OBJECTIVE**

Some applications are easier to attack with CSRF than others. Certainly applications with stored CSRF vulnerabilities are reliably exploitable, but other decisions developers make also affect exposure.

Many applications direct HTTP GET calls to the same handler that is used for HTTP POST. In Java servlets, for example, the doGet() method simply calls the doPost() method, redirecting the parameters. This makes simpler image or link based exploits possible, and eases the exploitation of CSRF flaws.

Some applications, particularly intranet sites and administrative consoles in switches, access points, bridges, and other network devices use integrated browser authentication. This type of authentication doesn’t expire, and the credentials remain available until the browser is closed. For someone who works with their browser all day, this can be hours or even days. This is a very wide window for attack, and this design decision increases the effectiveness of reflected CSRF attacks. Many applications have extremely long cookie lives, allowing users to return to the site without re authenticating. Long session lives can expose users to the risk of CSRF hours or even days after using a site. Popular sites with this configuration are of course even worse as attackers can guess that a large number of people selected at random are users of those services. A few applications allow users to change their passwords without entering their old password. If these password change mechanisms are vulnerable to CSRF, then attackers may target this feature.

**METHODOLOGY**

1. A user signs into www.good-banking-site.com using forms authentication. The server authenticates the user and issues a response that includes an authentication cookie. The site is vulnerable to attack because it trusts any request that it receives with a valid authentication cookie.
3. The user selects the submit button. The browser makes the request and automatically includes the authentication cookie for the requested domain, www.good-banking-site.com.
4. The request runs on the www.good-banking-site.com server with the user's authentication context and can perform any action that an authenticated user is allowed to perform.

**Token-based authentication**

When a user is authenticated, they're issued a token (not an antiforgery token). The token contains user information in the form of claims or a reference token that points the app to user state maintained in the app. When a user attempts to access a resource requiring authentication, the token is sent to the app with an additional authorization header in form of Bearer token. This makes the app stateless. In each subsequent request, the token is passed in the request for server-side validation. This token isn't encrypted; it's encoded. On the server, the token is decoded to access its information.
CONCLUSION

In this paper, the work were focused to prevent CSRF attacks on the server side, banks and merchants should transition from cookies that perform session-tracking to session tokens that are dynamically generated. This would make it more difficult for an attacker to get a hold of a client’s session.

REFERENCES