Layer 7 Http Methods and Cross-Site Request Forgery Attack

Author

Bharti, Neelam Shrivastava
Dept. of Computer Science Engineering
Ganga Institute of Technology &Mgmt
MDU, Rohtak (Haryana)
Email- bharti.gitam@gmail.com, neelam.sri@gmail.com

Abstract

CSRF is an attack which forces an end user to execute unwanted actions on a web application in which he/she is currently authenticated. With a little help of social engineering (like sending a link via email/chat), an attacker may force the users of a web application to execute actions of the attacker's choosing. A successful CSRF exploit can compromise end user data and operation in case of normal user. If the targeted end user is the administrator account, this can compromise the entire web application.

During the course of internet surfing and keeping myself secure from the attacker links on Social Network Platforms like Facebook, Google Plus, Orkut & many more Social Platforms, I found out that the security issues on posting malicious requests unknowingly from the user credentials on the specific website or Web Server can be easily crafted by a malicious attacker. If the user by mistake clicks the link, he may submit some false information to the Website, Web Application & Web Server. There are no mechanisms using which we can stop the request made by the user. This is because the functionality of the Layer 7 HTTP Protocol methods is to send Requests from clients to the server and receive a response from the server.

EARLIER WORK

Client-Side Techniques

Client-side techniques in the area of web security are characterized by the fact that they are primarily used by the consumers of web applications, rather than by the providers. In this sense, client-side techniques are either targeted at the protection of their users, or at the detection of security flaws in applications for which no source code is available.
Huang et al. described WAVES, a black-box testing tool for the detection of SQL injection vulnerabilities. Since it does not make use of source code during its analysis, it computes its results by means of fault injection and behavior monitoring. In addition, a machine learning component is applied to guide the testing process. Similarly, a black-box approach is used by Kals et al. in their tool called SecuBat. Apart from SQL injection vulnerabilities, they are also able to detect XSS flaws. One of the main drawbacks of black-box analyses is that they cannot guarantee full coverage of all execution paths inside a program. As a result, parts of the application might remain unscanned, which potentially leads to undetected vulnerabilities.

Noxes by Kirda et al. is a client-side, application-level firewall that offers protection against cross-site scripting attempts. To this end, the authors present techniques for the recognition of outgoing browser connections that might be part of an XSS attack. This recognition process includes the analysis of links embedded in a web page, and the computation of the amount of information that might be leaked to an attacker when a link is followed. The main idea behind this technique is that sensitive information can be transmitted either by a single link that is constructed dynamically inside the user’s browser, or by several static links. Based on this information, connection rules are generated on-the-fly, and the user is prompted when a connection violates the existing set of rules.

Vogt et al. enhanced the Firefox web browser with a taint tracking mechanism. This technique dynamically tracks the flow of sensitive information (such as user cookies) inside the browser. Whenever such information is about to be transferred to a third party, an alert is raised. Since it is not possible to detect all types of information flows dynamically, an additional static analysis of the web page is applied on-demand.

Johns and Winter presented a proxy-based, client-side solution against XSRF attacks, which is orthogonal to our approach. They also build upon the token approach, and additionally propose the use of an outside entity for detecting IP-based authentication. For those cases in which JavaScript code initiates HTTP requests, this code is altered automatically to contain the token. In contrast to this technique, which requires a certain extent of automated program understanding, a manual treatment of these rare cases on the server side appears to provide a more stable and efficient solution. Apart from the XSRF prevention techniques presented in this thesis, the approach of Johns and Winter outlined above is, to the best of our knowledge, the only scientific contribution to this topic.
Dynamic Server-Side Techniques

Dynamic techniques that operate on the server side can be divided into two different groups. On the one hand, it is possible to track the flow of tainted values dynamically (i.e., at runtime), and to disrupt program execution whenever a malicious value is used at a sensitive point in the application. On the other hand, approaches based on anomaly detection attempt to recognize suspicious user request before they reach the vulnerable application, and take measures to protect the application from potential attacks.

One of the most prominent examples for dynamic taint tracking is Perl’s taint mode. Analogously,

Nguyen-Tuong et al. and Pietraszek et al. described modifications to the PHP interpreter that permit the tracking of tainted values.

Haldar et al. adapted the Java virtual machine to achieve a similar effect for the execution of Java classfiles, without requiring the correspondingsource code.

Su and Wassermann base their work on a formal definition of SQL injection attacks. In their definition, SQL injection occurs when the intended syntactic structure of SQL queries is changed by tainted input. To be able to check whether this policy is violated by a program, they track tainted input dynamically by enclosing it within randomly generated makers. When the program issues an SQL query, the markers indicate the points of the query that contain potentially malicious values. The server-side techniques outlined above operate by tracking the flow of malicious values.

PRESENT WORK

The web server can be scanned using Scanning tools provided by various security organizations. I have tried a lot of tools to protect my security, but it is really difficult to implement the security update for the vulnerability even if the problem is identified. This is because of lack of knowledge of an average web developer. The security measures implementations require manual approach till now, In order to protect the security of the Web Server and the websites. Manually managing a lot of sites is on a single web server is not easy and I am not willing to allow someone to compromise my security due to lack of technical knowledge about vulnerability scanning and implementing the accurate security measure. Many Web applications are fairly good at identifying users and understanding requests, but terrible at verifying origins and intent.

LITERATURE SURVEY

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP has been in use by the World-Wide Web global information initiative since 1990. The first version of HTTP, referred to as HTTP/0.9, was a simple protocol for raw data transfer across the
Internet. HTTP/1.0, as defined by RFC 1945, improved the protocol by allowing messages to be in the format of MIME-like messages, containing meta-information about the data transferred and modifiers on the request/response semantics. Practical information systems require more functionality than simple retrieval, including search, front-end update, and annotation. HTTP allows an open-ended set of methods and headers that indicate the purpose of a request. It builds on the discipline of reference provided by the Uniform Resource Identifier (URI), as a location (URL) or name (URN) [2].

The HTTP protocol is a request/response protocol. A client sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content over a connection with a server. The server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity meta-information, and possible entity-body content.

In Order to send a request from Server to the client we need the methods to send requests and receive responses. HTTP Methods are described in the following:

1. Safe Methods
2. Idempotent Methods

**IMPLEMENTATION DETAILS**

The request I made:
GET /Vulnerable-face book-page HTTP/1.1
Host: www.facebook.com
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Cookie: PREF=ID=110a6ee3c4ee3e55

I found out a CSRF Vulnerability in the response, which allows me attack a logged in user:

The Attack Page Code:

```html
<!DOCTYPE HTML PUBLIC '-//W3C//DTD HTML 4.01 Transitional//EN'>
<html lang="en">
<head>
<title>My Scanning Software</title>
</head>
<body>
<img src="http://www.facebook.com/?ref=logo"/>
</body>
</html>
```
RESULT

VICTIM SYSTEM IS HACKED
CONCLUSION

CSRF attacks are relatively simple to diagnose, exploit and fix. Sites can be analyzed in a matter of seconds; attacks can be constructed in a matter of minutes. The most plausible explanation for the prevalence of these attacks is that web developers are unaware of the problem or think that defenses against the better known cross site scripting attacks also protect against CSRF attacks. I hope
the attacks I have presented show the danger of CSRF attacks and help web developers to give these attacks the attention they deserve. Once web developers are made aware of CSRF attacks, they can use tools like the ones i have created to protect themselves.

I suggest the creators of frameworks add CSRF protection to their frameworks, thereby protecting site built on top of such a frameworks. Adding CSRF protection at the framework level frees developers from duplicating code even the need to understand CSRF attacks in detail. Until every site is protected from CSRF attacks, users can takes steps to protect themselves using our browser plug-in for Firefox. Similar plug-in could be written for other browsers.

The root cause of CSRF and similar vulnerabilities probably lies in the complexity of today’s web protocols and the gradual evolution of the web from a data presentation facility to a platform for interactive services.

REFERENCES