© 2016 IJSRSET | Volume 2 | Issue 1 | Print ISSN : 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

Secure Web Application: Preventing Application Injections

Chokhawala Kirit I., Dr. Vinit Kumar Chuabay, Dr. A. R. Patel

Department of Computer Science and Engineering,. Mewar University, Rajasthan, India

ABSTRACT

In the recent years, web applications are the number one source of vulnerabilities targeted by Hackers. Although traditionally companies have used intrusion detection and prevention systems which monitor the network in general, there is now a widespread use of Web Application Firewalls as a security solution that monitors and protects only web applications. A web application is a software application that is accessed over the Internet using HyperText Transfer Protocol (HTTP). In a typical web application a client, such as a browser, interacts with a web server by exchanging a series of messages that are made up of HTTP requests and responses. An attacker often exploits vulnerabilities that exist in a web application to launch attacks. The focus of this research paper is to study and analyze the application level attacks for secure web application. Application level attacks covered Cross Site Scripting attack, SQL injection attack, Command Injection Attack and Cookie Poisoning attack.

Keywords - Web application, Cross Site Scripting attack, SQL injection attack, Command Injection Attack and Cookie Poisoning attack.

I. INTRODUCTION

Nowadays web applications have become ubiquitous. As the number of web applications increases the amount of traffic on the internet is also growing up. This results in the increasing threat of web applications being attacked. They continue to be a prime vector of attack for criminals, and this trend shows no sign of abating; attackers increasingly launch attacks like cross-site scripting, SQL injection and many other techniques aimed at the application layer. Web application vulnerabilities can have many things including poor input validation, insecure session management, improperly configured system settings and flaws in operating systems and web server software.

Certainly writing secure code is the most effective method for minimizing web application vulnerabilities. However, writing secure code is much easier said than done and involves several key issues.

Security has been the critically important part of majority of web applications. The web applications access the web server which in turn accesses the database servers. Thus proper security has to be implemented at every step during the access mechanism. Analysis carried out by Common Vulnerabilities and Exposures (CVE) [1] reports that majority of today‘s security loop holes are found in web applications.

Application level attacks known attacks include Cross Site Scripting attack, SQL injection attack, Command Injection and Cookie Poisoning etc, whose main aim is to tamper or deface web applications or impersonate as a real legitimate user. Web applications provide users with client server functionality by accessing a series of web pages. These web pages often contain dynamic interactive web content and script code which gets executed in the user browser. Thus web applications are continuously subjected to attacks [2][3][4] such as cross-site scripting, cookie stealing, session hijacking, browser hijacking, and the most recent being self-propagating worms in Web-email and social networking sites. In fact most of the research conducted shows that web application attacks are the most common problems on the internet today.[5]

II. SECURE WEB APPLICATION: PREVENTING APPLICATION INJECTIONS

A. Cross Site Scripting Attack

Cross Site Scripting (XSS) vulnerabilities have been the nightmare for Web applications for years now. Recent studies have shown that XSS has become the most common security problem. An analysis of the WASC [6] reveals that 100,059 XSS vulnerabilities have been detected by analyzing 31,373 Web sites. Cross Site Scripting (XSS) vulnerabilities penetrate web applications by injecting client side script into web pages viewed by other users. Majority of the websites including Face book, Twitter, McAfee, MySpace, eBay and Google have been the targets of XSS exploits. XSS occurs because of various limitations of security existing in many Web applications .i.e. when user inputs are not properly sanitized. The code to execute XSS is written in popular languages like PHP, Java,.NET. Attackers inject malicious code through these inputs, thereby causing unintended script executions through clients‟ browsers. Although a number of solutions have been proposed by researchers over time ranging from static analysis to complex runtime protection mechanisms, the data collected by semantic as of 2007 reveal that 80.5% of all security vulnerabilities are XSS.

Let‘s demonstrate XSS with a simple example. Assume there's a public forum where people can ask Questions regarding computer science. Each question is stored in a database and rendered as a list, if someone requests the relevant section of the forum. Such a list might look like this (No XSS embedded here):

Sample forum listing:
<html>
<head>
<title>The Question and Answer example forum – Computer Science section</title>
<link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
List of questions:
Q: "Which is the best OOP language in current times

Q: "What are the attributes of RDMS?"
</body>
</html>

When a hacker visits this page he will immediately notice that the text OOP language is rendered italic in his browser and conclude that the user that posted the question added the corresponding tags himself. Now the hacker might post a "question" in a different way like this:

<Script>alert (‘you have been XSSed') ;< /script>
Forum listing with embedded XSS attack:
<html>
<head>
<title>The Question and Answer example forum – Computer Science section</title>
<link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
List of questions:
Q: "Which is the best OOP language in current times

Q: "What are the attributes of RDMS?"
Q: "<Script>alert (_you have been XSSed') ;
< /script>"
</body>
</html>

Now, every time a user requests this list, a pop-up will be generated and appear in that user's browser that displays the words "you have been XSSed". While only some clever users will actually consider this an attack, other will surely not pay any heed and consider it as a normal pop up. By this way of injecting malicious scripts into web pages, an attacker can gain high access-privileges to sensitive page content, cookies, and a variety of other information maintained by the browser on behalf for user, making cross-site scripting attacks a unique case of code injection [7].

Types of XSS Attacks:

XSS attacks are mainly categorized into three types:

1. **Persistent or Stored XSS**
2. **Non Persistent or Reflected XSS**
3. **DOM based XSS**

1. Persistent or Stored Attack:

Stored XSS works if an HTML page includes data stored on the Web server (e.g. from a database) that originally comes from user supplied data. All an attacker has to do is to find a vulnerable server and post an attack. From that moment on, the server will distribute the exploit automatically to all users requesting the vulnerable page. Persistent or stored XSS attack is called persistent because it gets stored somewhere on the server and the effect of the attack is not immediate.

An example of this type of attack is when someone writes a HTML formatted review or comments on a review board like social networking websites or forum for other users to read. When some user reads the review the code gets executed on the user‘s browser and does some unwanted stuff like stealing cookies, redirect to some other page including website defacement etc.
For example the code in the comment or review can be like this

 Thank for your review <script>
window.location.href="http://www.abc.com"</script>

The above message will be stored in the database as it is and when some future user visits the page, the comment will be displayed but immediately the code in the script tag will be executed and the victim will be redirected to - abc.com.

2. Non-Persistent or Reflected XSS:

The second type (reflected XSS) works because some part of an HTTP request (usually a URL Parameter, cookie or the referrer location) is reflected by the Web server into the HTML content that is returned to the requesting browser. The word ―Reflected here means that input is written back unaltered. In this case, a hacker would have to craft a malicious URL and make someone else follow/open that link:

http://www.example.com/mypage.asp?id=<script>doBadThings () ;< /script>

This can be done by sending someone a manipulated e-mail (with the link) and use Phishing techniques to make the receiver believe that clicking on the link is a good thing. A second Approach would be to post such a link somewhere on the Internet, e.g. in a blog, forum, and wait for someone to follow it.

3. DOM based XSS:

The third type (DOM-based XSS) is very similar to the reflected attack. The difference is that the attack code isn't embedded into the HTML content back sent by the server. Therefore all server-side XSS detection techniques fail. Instead, it is embedded in the URL of the requested page and executed in the user's browser by faulty script code, contained in the HTML content returned by the server. Faulty means that the script reads a URL parameter and dynamically adds it to the document object model without any validation: document. Write (document.location.href); This way, unwanted tags are added to the DOM locally at runtime and are subsequently executed.

B. SQL Injection Attack

SQL Injection attack [8],[9] is one of the many web attack mechanisms used by hackers to steal data from organizations. It is perhaps one of the most common application layer attack techniques used today. It is the type of attack that takes advantage of improper coding of your web applications that allows hacker to inject SQL commands into say a login form to allow them to gain access to the data held within your database.
SQL injection attacks pose a serious security threat to Web applications: they allow attackers to obtain unrestricted access to the databases underlying the applications and to the potentially sensitive information these databases contain. Although researchers and practitioners have proposed various methods to address the SQL injection problem, current approaches either fail to address the full scope of the problem or have limitations that prevent their use and adoption.
Many researchers and practitioners are familiar with only a subset of the wide range of techniques available to attackers who are trying to take advantage of SQL injection vulnerabilities. As a consequence, many solutions proposed in the literature address only some of the issues related to SQL injection. To address this problem, the different types of SQL injection attacks known to date are listed below.

- Tautologies
- Piggybacked Queries

- Malformed Queries
- Inference
- Union Queries
- Alternate Encodings
- Leveraging Stored Procedures

Following are some example queries showing the above variants of SQLIA

- SELECT acct FROM users WHERE login= _‘OR 1=1—_ AND pin= 0 //tautology
- SELECT acct FROM users WHERE login= __ UNION SELECT cardNo from CreditCards where acctNo = 7032 -- AND pin= 0 //UNION
- SELECT acct FROM users WHERE login= _abc‘ AND pin= 0;
- drop table users //piggybacked queries
- SELECT acct FROM users WHERE login= _abc‘ AND pin= convert(int, (select top 1 name from sysobjects where xtype = _u‘)) //Malformed queries
- SELECT acct FROM users WHERE login= _legalUser‘ AND ASCII(SUBSTRING((select top 1 name from sysobjects), 1, 1)) > X WAITFOR 5 –_ AND pin= //Inferences
- SELECT acct FROM users WHERE login= _‘ AND pion=0; exec(char(0x73687574646f776e)) //Alternate encodings

For strored procedures attackers can invoke these procedures by manipulating the query. Following are some defense mechanisms [8], [9] which will prevent SQL Injection attack.

- Parameterize all Queries
- Validating input
- Limiting Permissions
- Use Only Stored Procedures
- Concealing Error Messages
- Segregate data
- Use encryption/hash functions where appropriate
- Limiting Damage

C. Command Injection attack

The purpose of the command injection attack [10] is to inject and execute commands specified by the attacker in the vulnerable application. In situation like this, the application, which executes unwanted system commands, is like a pseudo system shell, and the attacker may use it as any authorized system user. However, commands are executed with the same privileges and environment as the application has. Command injection attacks are possible in most cases because of lack of correct input data validation, which can be manipulated by the attacker (forms, cookies, HTTP headers etc.).

The variants of the command injection attack are discussed below.

Attacker adds his own code: The attacker extends the default functionality of the application without the necessity of executing system commands.

OS Command Injection: An OS command injection attack occurs when an attacker attempts to execute system level commands through a vulnerable application.

D. Cookie Poisonning Attack

Cookie Poisoning [11] attacks involve the modification of the contents of a cookie (personal information stored in a Web user's computer) in order to bypass security mechanisms. Using cookie poisoning attacks, attackers can gain unauthorized information about another user and steal their identity.

Many Web applications use cookies to save information (user IDs, passwords, account numbers, time stamps, etc.). The cookies stored on a user's hard drive maintain information that allows the applications to authenticate the user identity, speed up transactions, monitor behavior, and personalize content presented to the user based on identity and preferences. For example, when a user logs into a Web site that requires authentication, a login CGI validates his username and password and sets a cookie with a numerical identifier in the user's browser. When the user browses to another page, another CGI (say, preferences.asp) retrieves the cookie and displays personalized content according to the values contained in the cookie.

The cookies are as shown below

GET /store/buy.asp?checkout=yes HTTP/1.0 Host: www.onlineshop.com
Accept: */* Referrer: http://www.onlineshop.com/showprods.asp

Cookie: SESSIONID=570321ASDD23SA2321; BasketSize=3; Item1=2892; Item2=3210; Item3=9942; TotalPrice=16044;

The request includes a cookie that contains the following parameters: SESSIONID, which is a unique identification string that associates the user with tthe session of the user. This session id can be tampered to poison the cookie.

III. CONCLUSION

This paper carried out analysis of various application level attacks and classified those attacks. The information contained in this paper could be very useful for new application/web developers for developing smarter and secure applications running over the web. Although a complete secure application is not guaranteed in the modern world, but still a considerable amount of work and research has been done in this area. Completely securing a web application seems to be a daunting task for developers today.

IV. REFERENCES

[1] Sandeep Bhatkar, Abhishek Chaturvedi, and R. Sekar.Dataflow anomaly detection. In IEEE Symposium on Security and Privacy, May 2006

[2] E. Chien. Malicious Yahooligans. http://www.symantec.com/avcenter/reference/malicious. yahooligans.pdf, 2006.

[3] Open Web Application Security Project. The ten most critical Web application security vulnerabilities http://umn.dl.sourceforge.net /sourceforge/owasp/ OWASPTopTen2004.pdf,2004

[4] The Samy worm. http://namb.la/popular

[5] MITRE. Common vulnerabilities and exposures. http://cve.mitre.org/cve/, 2007

[6] Xie and A. Aiken, ―Static Detection of Security Vulner-abilities in ScriptingLanguages,Proc. 15th Use nix Security Symp. (Use nix-SS 06), vol. 15, Use nix, 2006, pp.179-192.

[7] https://www.isecpartners.com/media/11961/CSRF_ Pape r.pdf

[8] William G.J. Halfond, Alessandro Orso, Member, IEEE Computer Society, and Panagiotis Manolios, Member, IEEE Computer Society, ―WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation‖,, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 34, NO. 1, JANUARY/FEBRUARY 2008

[9] Justin Claarke, SQL Injection Attack and Defenses. U. S.: Syngress Publishing, Inc., 2009.

[10] Infodox, Insecurity Research, Online], http://insecurety.net/?p=403

[11] Imperva Online], https://www.imperva.com/ resources /glossary?term=cookie_poisoning.