Defensive Analysis on Web-Application Input Validation for Advanced Persistent Threat (APT) Attack `

Myo Myint Oo, Tun Myat Aung
University of Computer Studies, Yangon
myomyintoo.cu@gmail.com , tma.mephi@gmail.com

Abstract

Nowadays, any kinds of cyber-attacks are firstly attacked to web site and the site contains information about business, government and other organizations. So, the role of web page security is becoming the essential part of cyber security. Most of the web pages are suffered from attacks such as Advanced Persistent Threat (APT). An APT is an extended campaign targeted at a specific organization to achieve a clear objective. Most of the web pages have much vulnerability due to insecure source codes. In this paper, it will be defined web page vulnerabilities on PHP source code. It will be collected the attacks from the scenarios. The result can be concluded the attack vectors via the source code. Each level of source code can be generated by the attributes of source code. These attributes can be measured in order to secure source codes and can be evaluated by measuring the vulnerabilities metrics.

1. Introduction

Nowadays, the rise in the use of computers and the growth of the internet brought about cyber-crimes. Meanwhile, cyber-attacks have become more sophisticated than ever. In respond to the developments, the ways of the attack and defense between cyber-crimes and information security technologies occur have become increasingly complicated. One of the most complex and advanced cyber-attacks in recent years is the Advanced Persistent Threat (APT), which attacks corporations and government agencies. The some of the prominent cases for APT attack include Stuxnet, Duqu, Red October, Mask, etc, and each of these attacks had a different target and purpose. Most of the APT attack cases are suffered from Web Application by using watering hole attack. Moreover, web applications are one of the most common platforms for information and services delivery over internet nowadays. Most of the web applications may contain security vulnerabilities which enable the attackers to exploit them and launch attacks. Web sites conducting business, containing valuable information for a malicious hacker, are at more vulnerability risk than others. E-commerce websites hold valuable information such as credit card numbers, private and personal data, and are also placed at a high risk position. Therefore, confidentiality, integrity and availability of information are lost. Web application security is becoming more essential at the present time. Most of the web sites are developed by the PHP in this day. It has many advantages but it still has considerable number of vulnerabilities in order to exploit them.

2. Literature Review

As a highly exploited set of vulnerabilities, input validation errors have generated a significant amount of academic interest. A brief

review through some of the current research topics is provided. In 2010, the two authors, Molnar & Livshits proposed the research paper, SCRIPTGARD: Preventing Script Injection Attacks in Legacy Web Applications with Automatic Sanitization [1]. Their research was an analysis of existing 400,000 lines of code in web application. They developed a system for preventing such problems by automatically matching the correct sanitizer.

The second paper is proposed by Scholte in 2012. The title was Preventing Input Validation Vulnerabilities in Web Applications through Automated Type Analysis [2]. His research was the novel techniques for preventing XSS/SQLi using automated data type detection of inpu

tparameter. This paper claims 65-83% success against the tested vulnerabilities with no additional overhead for the developer. And the two authors, Scholte and Balzarotti was proposed An Empirical Analysis of Input Validation Mechanisms in Web Applications and Languages in 2012 [3].

The authors performed an empirical study of over 7000 input validation vulnerabilities. They used 79 web application frameworks in what is the largest meta-study in the field to date. In 2011, the author, Samuel purpose Context-Sensitive Auto- Sanitization in Web Templating Languages Using Type Qualifiers [4]. This research was strived to bring better auto-sanitization to web code being developed within Java and PHP web templating frameworks.

In 2014, the author, Yinzhi Cao purposed the paper, PathCutter: Servering the Self-Propagation Path of XSS JavaScript Worms in Social Web Networks [5]. He exploited JavaScript XSS vulnerabilities rampantly infect millions of web pages. He proposed PathCutter as a new approach to severing the self-propagation path of JavaScript worms. PathCutter works by blocking two critical steps in the propagation path of an XSS worm.

3. Attack Vectors from Input

Nowadays, the new web-based attack types and vectors are coming out. This can cause in businesses, communities and individuals to take security seriously now more than they ever have in the past. With the development of advanced technologies, attack vectors consists of viruses, e-mail attachments, web pages, pop-up windows, instant messages, chat rooms, and deception. All of these methods involve programming, except deception, and weakening system defenses. In this paper, the attack vectors from the input validations such as SQL Injection, Cross-Site Request Forgery (CSRF) and Cross-Site Scripting Attacks (XSS), produce the grade of security rank and evaluated the quality of web application on certain types of attacks.

3.1. SQL Injection

SQL injection attack exploits the weakness of web application's back-end database. This kind of exploits occurs when user input is not cleaned for sting escape characters and the web application submits code amounting to the database command to the database server. In 2006, Scambray et al described the SQL injection and the attack involves the following steps. The first step is to insert invalid data into a web application's SQL database input field. The second step is to manipulate the input until you can map out the inner workings of the unseen SQL statement.

The third step is to craft an input that will successfully escape the data input context and

allow the ability to enter database commands. The fourth step is to map the database by with SQL queries, either by guessing table names, brute force or some other techniques.

The last step is to read/ write/ delete the data of interest with a SQL query [6]. A simple example of SQL query is as follows:

SELECT AMOUNT

FROM CUSTOMER

WHERE USERNAME = 'John Smith' AND PASSWORD = 'S123';

Now, by submitting the following text in the USERNAME and PASSWORD fields, the attacker can craft his own queries to the database.

USERNAME = ' ' OR 1=1 - - /

PASSWORD = 'anything'

So, the resulting query may be the following:

SELECT AMOUNT

FROM CUSTOMER

WHERE USERNAME = ' ' OR 1 =1 - -/

AND PASSWORD = 'anything'

Since the input field is in this case not cleaned of escape characters, the double dash is interpreted by the parser as meaning that everything to right is a comment and thus dropped. So, the parsed query that gets into the database is

SELECT AMOUNT

FROM CUSTOMER

WHERE USERNAME = ' ' OR 1=1

Which is interpreted as "return all customers' USERNAME where the username is a null value or 1=1". This string will always be true and thus dump all of the stored AMOUNT.

3.2. Cross-Site Request Forgery (CSRF)

Cross-Site Request Forger (CSRF) is an attack that forces an end user Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they are currently authenticated. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email addresses, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application [9].

There are numerous ways in which an end user can be tricked into loading information from or submitting information to a web application.

In order to execute an attack, it must be understood how to generate a valid malicious request for our victim to execute. It is considered the following example: John Smith wishes to transfer 10000 dollars to Merry Smith using the bank.com web application that is vulnerable to CSRF. David, an attacker, wants to trick into sending the money instead of John Smith. The attack will compromise the following steps. The first step is to build an exploit URL or script. The second step is to trick John Smith into executing the action with social engineering.

3.2.1. GET scenario in PHP

If the application was designed to primarily use GET requests to transfer parameters and execute actions, the money transfer operation might be reduced to a request like:

GET http://bank.com/transfer.do?acct =JohnSmith& amount=10000 HTTP/1.1

The attacker, David now decides to exploit this web application vulnerability using John Smith as his victim. David first constructs the following exploit URL which will transfer 100,000 from John Smith's account to his account. He takes the original command URL and replaces the beneficiary name with himself, raising the transfer amount significantly at the same time:

http:// bank.com/transfer.do?acct=David& amount= 100000

The social engineering aspect of the attack tricks John Smith into loading this URL he is logged into the bank application. This is usually done with one of the following techniques. The first method is to send an unsolicited email with HTML content and the second method is to plant an exploit URL or script on pages that are likely to be visited by the victim while they are also doing online banking.

3.2.2. POST scenario

The only difference between GET and POST attacks is how the attack is being executed by the victim. Let's assume the bank now uses POST and the vulnerable request looks like this:

POST http:// bank.com/transfer.do HTTP/1.1
acct=John Smith&amount=100

Such a request cannot be delivered by using standard A or IMG tags, but can be delivered by using a FORM tag:

<form action = “http://bank.com/transfer.do” method = “POST”>
<input type= “hidden” name= “acct” value= “David”/>
<input type= “hidden” name= “amount” value= “10000”/>
<input type= “submit” value= “View my pictures”/>
</form>

This form will require the user to click on the submit button, but this can be also executed automatically by using PHP code.

3.3. Cross-Site Scripting (XSS) attack

A cross-site scripting attack is one of the top 5 security attacks carried out on a daily basis across the Internet, and PHP scripts may not be immune. Also known as XSS, the attack is basically a type of code injection attack which is made possible by incorrectly validating user data, which usually gets inserted into the page through a web form or using an altered hyperlink. The code injected can be any malicious client-side code, such as JavaScript, VBScript, HTML, CSS, Flash, and others. The code is used to save harmful data on the server or perform a malicious action within the user’s browser. Unfortunately, cross-site scripting attacks occur mostly, because developers are failing to deliver secure code. Every PHP

programmer has the responsibility to understand how attacks can be carried out against their PHP scripts to exploit possible security vulnerabilities. Reading this article, you will find out more about cross-site scripting attacks and how to prevent them in your code.

Let's take the following code snippet.

```
<form action= "post.php" method= "post">
 <input type= "text" name= "cmdTest"
value= "">
 <input type= "submit" name= "submit"
value = "Submit">
</form>
```

Here we have a simple form in which there is a text box for data input and a submit button. Once the form is submitted, it will submit the data to post.php for processing.

Let's say all post.php does is output the data like so:

```
<?php
echo $_POST[ "cmdTest"];
?>
```

Without any filtering, a hacker could submit the following through the form which will generates a popup in the browser with the message "Your web site has been hacked".

```
<script>alert("Your Web site has been
hacked")</script>
```

This example, despite its being malicious in nature, does not seem to do much harm. But think about what could happen in the JavaScript code was written to steal a user's cookie and extract sensitive information from it? There are far worse XSS attacks than a simple alert() call.

4. Procedure of Advanced Persistent Threat

Advanced Persistent Threat refers to a long-term and sophisticated attack on a specifically targeted entity. The attacker is often state-sponsored and seeks to gain high-value intelligence from other governments, but may also be performed by and target private organizations. There are many steps that must be taken in order for an APT attack to be successful. *The first step* is choosing a target. The attacker first determines whom they wish to infiltrate and what they wish to steal. Is the target of attacker to break confidential financial data or source code or technical drawings? *The second step* is target research. Once a target has been selected, the attacker will do extensive background research on his target. *The third step* is penetration. After a target has been acquired, the attacker typically creates a customized phishing email in the hope that their target will open an attachment that contains an exploit that allows the attacker to plant remote access malware on the target's computer. *The fourth step* is elevation of privileges.

Once the attacker has gained a foothold inside a target's network, an attempt is made to exploit vulnerabilities on other internal computers to gain further access on the network. *The fifth step* is internal network movement. If the attacker was successful in gaining further access inside the network, they can then expand their control to other machines on the network and compromise other computers and servers, allowing them to access data throughout the network. *The sixth step* is data theft. Once network access has been achieved, data can be easily stolen. Passwords, files, databases, email accounts and other potentially valuable data can all be sent back to the attacker. *The last step* is maintenance and administration.

Even after the requisite data has been stolen, an attacker may decide to remain present on the target's network. This requires vigilance on the attacker's part in order to evade detection and maintain surveillance on the target's data assets to ensure further data can be stolen. The attacker first chose the target and then he did the targeted

research. He must do any method to attack the target organization. Most of the cases are firstly used to attack to the web sites [8].

5. Proposed Methodology

In our proposed research method, there will be five main processes. They are

(1) Analysis of web attacks scenarios
(2) Finding vulnerabilities points
(3) Source code analysis
(4) Generating source code attributes
(5) Evaluating with vulnerabilities metrics.

In the first process, the attacks vectors will be found through the past occurrences (scenarios) of web sites, and these resulted attack vectors are exploited on PHP source code. The result of the first process is attack scenarios. In the second process, there will be found the vulnerabilities points of the source code. The result of the second process is vulnerabilities points. In the third process, the source codes of the vulnerabilities points are analyzed in order to defense the vulnerabilities. The result of the third process is the grade or rank of the secure codes. In the fourth process, there will be generated the source code attributes from the source code by using resultant ranks. So, the result is certain kind of source code attributes. In the last process, the attributes can be evaluated as the quality of web application by the using vulnerabilities metric method. So, the last result is vulnerabilities metrics. This research will be done at the second step of APT procedure. The illustration of the tentative method is shown in figure 1.

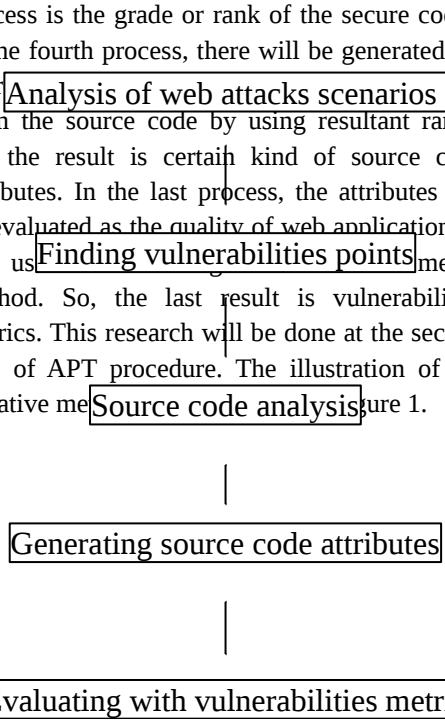

Figure1. Flow of Proposed Methodology

6. Evaluating Web Application by using Vulnerabilities Metric

Vulnerabilities metric is an area in computer security science that has been receiving adequate attention in recent times. Majority of works about vulnerabilities metric is mainly definitional, targeted towards providing guidelines for defining a security metric and specifying criteria for which to strive to achieve security. It is just a little that had been reported on actual metric that have been proven useful in practice. The Web Application Security Metrics (WASE) is a model that measures the security vulnerabilities found in any web page to enable individuals; merchants as well as commercial software developer determines the security strength of their web sites before putting them into e-business transactions on the Internet [7].

PHP applications have statistically significant higher rates of injection vulnerabilities than non-

PHP applications, and PHP applications tend not to use frameworks. As most web security experts likely expect, XSS and injection are the most pressing and severe vulnerabilities, as shown by the Open Source Vulnerability Database (OSVDB) and The Open Web Application Security Project (OWASP). In this paper, the input validation attributes of PHP source code may be the use of metacharacters (M), the use of wrong type (W), the use of too much input (TM), the abuse of hidden interfaces (HI), and the use of bearing unexpected commands (UC).

In this research, any kind of PHP application will be measured by using these attributes. These attributes can lead to the attacks such as Cross Site Request Forgery, Cross Site Scripting and SQL injection attacks. For example, there are five web applications by using PHP. The source code of these sites can be measured by the above attributes: M, W, TM, HI and UC. The vulnerabilities percentage in each attribute can be calculated by the Equation (1). The vulnerabilities metric is constructed by using vulnerabilities percentage of each attribute as shown in Table 1. The bar chart of vulnerabilities analysis can be seen in Figure 2. In this figure, there are sample of five PHP web applications such as site 1, site 2, site 3, site 4 and site 5. These sites can contain the web vulnerabilities such as using metacharacter, using wrong types, using too much input, abusing of hidden interfaces, and bearing unexpected commands from the aspect of input validation attributes. These vulnerabilities can be considered as vulnerabilities attributes. Each type of attribute can be computed by using Equation (1). The computed result can be expressed in bar chart with sample of five PHP web applications over the each type of vulnerability attribute. Each site can be expressed with different colors.

$$\frac{number\ of\ vulnerabilities \in each\ attribute}{tot\ number\ of\ vulnerabilities} \times 100 \quad (1)$$

Table 1. Sample Vulnerabilities Metric

	M	W	TM	HI	UC
Site1	20	10	0	25	30
Site2	30	50	40	25	0
Site3	0	60	70	30	50
Site4	40	0	50	35	38
Site5	39	36	20	0	25

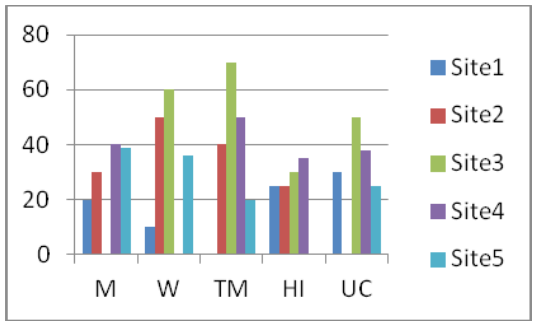

Figure 2. Bar chart of vulnerabilities analysis

7. Conclusion

Vulnerabilities in a web application evolved from the early development of the web technology in APT. As the time passed, some of the vulnerabilities were eradicated and some of them are still there, while new types of vulnerabilities were created and some serious vulnerabilities can be expected in the future. Similarly, sometimes attacks are done with compound vulnerabilities such as injection with XSS or injection with DNS hijacking whose consequences are more severe. In the real world, it would be difficult to say that an application is completely secure. Despite all the web threats, application can attain a maximum security with a better coding approach and the developer's knowledge of web security. This purposed methodology supports the web developers for writing and testing the secure codes.

References

[1] Molnar, Livishits, "SCRIPTGARD: Preventing Scirpt Attacks in Legacy Web Applications with Automatics Sanitization", 2012.

[2] Scholte, "Preventing Input Validation Vulnerabilities in Web Applications through Automated Type Analysis", 2012.

[3] Scholte, Balzarotti, "An Empirical Analysis of Input Validation Mechanisms in Web Applications and Languages", 2012.

[4] Samuel, "Context-Sensitive Auto-Sanitization in Web Templating Languages Using Type Qualifier", 2011.

[5] Yinzhi Cao, "PathCutter: Servering the Self Propagation Path of XSS JavaScript Worms in Social Web Networks", 2014.

[6] Erik Couture, "Web Application Injection Vulnerabilities", May 20th, 2013.

[7] G. E. Okereke, "Security Metrics Model for Web Application Vulnerability Analysis", *African Journal of Computing & ICT*, 2006.

[8] "Advanced Persistent Threat" www.business.att.com

[9] "Cross-Site Request Forgery (CSRF)", https://www.owasp.org