Are You Coding Safely? A Guideline for Web Developers

Chun-Chan (Bill) Cheng
Department of Computer Science and Engineering
Texas A&M University
Email: aznchat@tamu.edu

Chia-Cheng (Jeremy) Tso
Department of Computer Science and Engineering
Texas A&M University
Email: jjjjj222@tamu.edu

I. Introduction

With the proliferation of hand-holding devices, the ubiquitous accessibility has been a norm for today's applications. Basically every service today supports on-demand access through all kinds of devices, and users should be able to use whatever service they like to receive. To meet this requirement, more and more service providers tend to roll out their product as a web application.

Web applications have many advantages over the traditional download-and-install ones: First of all, they work on every platform as long as there's a web browser. This takes a lot of pains off software engineers, because once a job is done, it works everywhere – no more customization or porting for different platforms. Secondly, web applications can be patched instantly. Every user will have the same version of application at any given time, and it relieves engineers from tiresome backward-compatible requirements. Moreover, applications can be used immediately without tedious download-install process. This fast deployment, fast prototype, and light-weighted characteristics greatly facilitates the project development in terms of increasing time-to-market speed. It also helps service providers get the instant feedback from market, and thus alter their direction of strategy accordingly in early stage.

Due to these advantages, lots of programming languages with their frameworks have emerged for web development. With the help of these new tools, building a web application is not an "geek job" anymore: anybody can build a web application by simply following some basic tutorials. However, although these tools are easy to use, they have many pitfalls as well. For instance, in PHP 4.2.0 or lower, PHP had global variable that could be access any where in the application. Since PHP does not require variables to be initialized when declared, if programmers do not take steps to protect this variable. It is likely that this variable may be compromised by malicious user by using various of techniques, such as script injections.

Once a developer steps on some of them, the newly-built web application could be another puppets controlled by malicious hackers. Continuing the example stated above, if the malicious user decides to gain root access to the server folder of your website by using the compromised variable. See the example code below:

```
if (authenticated_user()) {
    $authorized = true;
}

Program continues on....

if ($authorized) {
    Sensitive_Data();
}
```

Since $\$authorize$ can be initialized even without $authenticated\_user()$ and it has not been initialized to false in the beginning. Attacker can take advantage of this, defining $\$authorized$ through various techniques, for example, defining the variable $\$authorized$ through $register\_globals$. Boom! Your website is compromised, all the sensitive data are exposed to the attacker!

To prevent this from happening, we want to provide some simple, clear but useful guidelines to help programmer code with "good habits". Normally, while engineers are busy on bringing out all the functionalities, they do not have enough time to work on security. By following our guidelines, they only have to pay a little extra attention, while getting a more secure application.

Note that our goal here is not to cover every vulnerabilities, which is intuitively impossible. Our goal here is to cover as many problems as possible with minimal efforts. You might think it's not good enough. However, the concept of security is that if the value of the data in your website least than the effort that one need to break in, then, in this case, the protection should be sufficient.

II. Problem Statement

Engineers have to face many pitfalls while working on an web application, such as various framework structures and different language behaviors. On developing, they are often too busy to pay attention on securities. This phenomenon leads to vulnerable sites, which could be used as medium to carry out malicious behavoirs by others.

III. Proposed Solution

Our target here is to provide some simple, clear but useful guidelines to help engineer code with "good habits", and thus eliminate most of vulnerabilities with minimal efforts.

IV. Related Work

Different web application vulnerability have been proposed before. In this section we will include some of the most common vulnerabilities that have been proposed till now. First of all, one of the most famous and most ancient attack is the sql injection attack which was first documented by Jeff Forristal, more know by the name of rain forest puppy [1]. SQL injection vulnerabilities have been described as one of the most serious threats for Web applications [2] [3]. In 2006, Professor Halfond classified the SQL injection attacks and proposed some countermeasure [4].

Secondly, another famous web application attacks is the Cross Site Scripting(CSS). Cross-site scripting carried out on websites accounted for roughly 84% of all security vulnerabilities documented by Symantec as of 2007 [5]. Cross-site scripting (XSS) attacks target an application's users by injecting code, usually a client-side script such as JavaScript, into a Web application's output. These attacks can cause small petty security ricks to devastating data theft depending on how sensitive are the data in the websites.

Third of all, the session management and user authentication. Web applications have to handle user authentication and establish sessions to keep track of each user's requests as HTTP does not provide this capability. If there is a breach in the session management of a web application, the attacker can then steal the user cookie and impersonate this person on the web. However, cookies are not accessible via non-HTTP methods, such as calls via JavaScript, by using document.cookie, and therefore cannot be stolen easily via cross-site scripting [5].

Last of all, the insecure direct object reference. If the web application design does not have mistake-proofing mechanism and assumes that users will always follow the application rules, then it is likely that a user may accidentally jump into a page that it was not granted access. This is the benign security breach, since the user may not want to steal any information, but what if a malicious user starts stealing the information or to make things worse, a hacker that can read the URL or can guess the hidden field of the URL [6]. If this happens, you web application can be in big trouble. [7].

V. PHP.ini

PHP.ini file is an important configuration file for PHP. You declare all the configuration settings in this file. Since the php.ini file is read each time PHP is initialized, it is important that PHP developer know the basic security configuration of the PHP.ini file. Below we will give some basic guidelines for developers on what to set to the environment variable in the PHP.ini to prevent petty attacks. Even though this configuration won't by itself fend off a determined attacker, but it will lower visibility to attacks that rely on simple techniques to scan for vulnerable targets [8].

A. expose_php

`Expose_php`, when turned on will tell the web server to send back the `X-Powered-By` header, in other words, `Expose_php` will return back in every request what version of PHP is installed.

1) Vulnerability: If a web application is using a PHP version that is out of date or using a PHP version with known vulnerabilities, malicious attackers can exploit this vulnerability causing the web application to be unsecure [9].

2) Recommendation: Setting the `Expose_php` to `off` is recommended. Even though determined attackers can still get the version of your php but it would take them more time trying to achieve this. In addition, the web application can stand up to simple attack scanning for specific version of php vulnerabilities. In conclusion, it is always better to set `Expose_php` to `off` than never.

B. post_max_size

The environment variable `post_max_size` is the variable that sets the size that `$_POST` method can send over to the server.

1) Vulnerability: If this environment variable is not set to a realistic value, for example, if you set the value to 1024 MB, then you are giving attackers permission to put your server at stress by sending oversized POST request. [10]

2) Recommendation: By setting a realistic value here you can mitigate some of the damage by those attacks. This protection allows you to limit the maximum size POST request that PHP will process.

References

[1] Jeff Forristal (signing as rain.forest.puppy). NT Web Technology Vulnerabilities. In *Phrack Magazine 8 (54 (article 8))*, December 1998.

[2] D. Aucsmith. Creating and maintaining software that resists malicious attack. www.gtisc.gatech.edu/bioaucsmith.html, 2004.

[3] T. O. Foundation. Top ten most critical web application. www.owasp.org/documentation/topten.html, 2005.

[4] William G.J. Halfond, Jeremy Viegas, and Alessandro Orso. A Classification of SQL-Injection Attacks and Countermeasures. In *Proceedings of the IEEE International Symposium on Secure Software Engineering*, Arlington, VA, USA, March 2006.

[5] Symantec Corp. Symantec Internet Security Threat Report Trends for JulyDecember 07. http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_exec_summary_internet_security_threat_report_xiii_04-2008.en-us.pdf, April 2008.

[6] Michael Cobb. Five common web application vulnerabilities and how to avoid them. http://searchsecurity.techtarget.com/tip/Five-common-Web-application-vulnerabilities-and-how-to-avoid-them, 2013.

[7] Test. Test test. www.google.com/, 212123.

[8] PHP Manual. The configuration file. http://php.net/manual/en/configuration.file.php.

[9] PHP Security Consortium. Phpsecinfo test information: expose php. http://phpsec.org/projects/phpsecinfo/tests/expose_php.html, 2006.

[10] PHP Security Consortium. Phpsecinfo test information: post max size. http://phpsec.org/projects/phpsecinfo/tests/post_max_size.html, 2006.