An analysis of security in a web application development process

Florent Gontharet

Ethical Hacking

University of Abertay Dundee

MSc Ethical Hacking

2015

Table of Contents

Abstract **2**

Introduction **3**

Procedure and results **4**

Thinking security while programming 4

Vulnerability Assessment 7

Discussion **10**

References **11**

Table of Figures

Figure 1: ZAP scanning interface 7

Figure 2: w3af setup 8

Figure 3: w3af testing gui and command 8

ABSTRACT

- **Introduction**

Security has to be a concern for web applications, vulnerabilities use vectors of attack that are well known and exposed in the introduction section. Tools such as w3af and ZAP are useful to ensure the security controls are covering the known vulnerabilities.

- **Procedure and results**

A first case study of the security concern in the development process of a web application, with the solution used in order to address the main threats of the OWASP Top 10 web application vulnerabilities.

Once covered, the security of the application is tested against the ZAP and w3af security testing applications, as well as the SQLmap script, to ensure SQL injections are deeply covered.

- **Discussion**

The inclusion of security in the development process allows the web application to be secured against the known vulnerabilities, and testing is important to ensure the security controls fill their role and covers all known vulnerabilities. This has to be included in a long time process as vulnerabilities are discovered.

INTRODUCTION

As Internet grow, Web application security has become a huge concern, fed with the numerous past events of data leakage and SQL injections. Every user can now interact, and that interaction can be used for other purposes. It is important for users to ensure the Web application they use is secured, as their data security depends on it. It is also important for companies, and developers have to include security in their process.

However, including security in the process is not enough, it is important to perform tests, and they are necessary to prevent most of the known attacks to success. Web application security testing can be performed with dedicated tools such as ZAP, w3af, Burp Suite, or SQLmap.

The practical will test an existing PHP programmed platform, as PHP represents 82% of the actual market (W3Techs.com, April 2015). Different kinds of attacks exist, the SQL injection remains one of the most dangerous as it targets data. Here are some of the main attacks presented:

- Broken Authentication and Session Management: The implementation is important, as a flaw can allow an attacker to compromise users' identity and personal data. Sessions are identified with an ID, if this one is predictable, or visible, an attacker could steal it to impersonate another user of the web application (Huluka and Popov, 2012).
- Cross Site Request Forgery (CSRF): While users don't pay attention if they are still logged into a web application, URLs to perform specific actions on the website can be forged in order to be sent to the users. Once they are clicked, the action is immediately performed as legitimate (Top Ten Project, 2013). This can go from buying a product to deleting all entries in a database...
- Cross-site scripting (XSS): The most common publicly-reported security vulnerability (Bates et al. 2010). The attack consist in injecting an external script into a web application, mostly to interfere with the user's session on the website, and per instance, steal his session. It is performed when user data is directly use for displaying into the web application.
- Remote and local file inclusion: Targets dynamic file include, and mostly PHP applications (iMPERVA, 2012). The user data is used to generate a path to dynamically load a file, what can be used in order to access a malicious script, and execute actions directly on the server.
- SQL injection: A web application can take user data as an input, what can allows a user to craft a dedicated string in order to execute malicious code, that can include data leakage, change, or deletion (Ashish, 2015).

PROCEDURE AND RESULTS

THINKING SECURITY WHILE PROGRAMMING

For the experiment, the website developed by the author accessible at http://ecf.wr0ng.name will be analysed and tested. The platform has been chosen because of its code accessibility, and the different approaches it offers, as it includes sessions, file uploads, database storage, user data and such.

At first, in order to address SQL injections, all queries requiring user data are handled by the PDO class (http://php.net/manual/en/class.pdo.php) from PHP 5, a query is as follow:

```
$qry = "SELECT * FROM users WHERE iduser=:iduser AND pwd=:pwd";

try{
	$res = $db->prepare($qry);
	$res->bindParam(':iduser', $idUser, PDO::PARAM_INT);
	$res->bindParam(':pwd', $pwd, PDO::PARAM_STR);
	$res->execute();
}
catch( PDOException $e )
{
	die( $e->getMessage());
}
```

We can see the prepared query, and the parameters for PDO, followed by the `bindParam()` method from the PDOStatement object `$res`, using the dedicated constants (http://php.net/manual/en/pdo.constants.php). That way, the query and its parameters remain unchanged, and the user data is handled separately.

If user data has to be displayed, in order to address Cross-site scripting issues, data will pass through the PHP `htmlentities()` function, that converts all characters part of the HTML syntax (such as `<` and `>`), into their corresponding HTML entity (respectively `<` and `>`). This avoids all interpreted string, such as the include of a iframe, or a script to steal the session cookies. Here is the use:

```
$uName = htmlentities($_POST['username']);
```

Dynamic includes can allow attacks known as local or remote file injection. It consist in including local or remote files, when user data is used as path for an include. In order to address the problem, multiple defences can be used. The chosen one is the white list, as the amount of pages is not really important, it does not represent a problem, more important platform will probably want to add a dynamic solution. The following defines all the allowed content for the variables, and redirects on the error:

```
$routes = array(
                    'index'=>'',
                    'login'=>'',
                    //    ...
                    'logout'=>'');
$file = (isset($routes[$_GET['page']])?$_GET['page']:$routes[0]);
include_once($file.'.php');
```

In the case of a value not existing in the list, the script will use the “index” value, at the index 0.

To avoid troubles with file upload, the original file name is changed for one generated for the occasion. In order to read the file, the chosen function is `file_get_contents()`, made to read the content as raw text. The output is here again sanitized using `htmlentities()` before being displayed.

In order to ensure information security, the password storage uses `bcrypt` that implements the Blowfish algorithm, the safest hash solution offered to us (Provos and Mazières, 1999), well known but still not cracked. As recommended by the PHP doc, here is the implementation with a cost of 12:

```
$options = array(
    'cost' => 12,
    'salt' => mcrypt_create_iv(22, MCRYPT_DEV_URANDOM),
);

$hash = password_hash($pwd, PASSWORD_BCRYPT, $options);
```

The data that a user can store when saving a personalized config file is considered as sensible, as it could eventually include passwords, personal data, or anything. In the eventuality of a leak, data is here again stored using the Blowfish encryption algorithm, but as it have to be decrypted, the PHP implementation is done that way, with the use of a master key, kept out of the public repositories:

```
function decrypt_blowfish($data,$key){
  $iv=pack("H*" , substr($data,0,16));
  $x =pack("H*" , substr($data,16));
  $res = mcrypt_decrypt(MCRYPT_BLOWFISH, $key, $x, MCRYPT_MODE_CBC,
                                                                $iv);
  return $res;
}

function encrypt_blowfish($data,$key){
  $iv_size = mcrypt_get_iv_size(MCRYPT_BLOWFISH, MCRYPT_MODE_CBC);
  $iv = mcrypt_create_iv($iv_size, MCRYPT_RAND);
  $crypttext = mcrypt_encrypt(MCRYPT_BLOWFISH, $key, $data,
                                          MCRYPT_MODE_CBC, $iv);
  return bin2hex($iv . $crypttext);
}
```

Finally, in order to restrict access to the system and the upload repository, a .htaccess forbids all direct access to them (the first part is for Apache 2.4, the second if for Apache 2.2, due to their different syntaxes):

```
<IfModule mod_authz_core.c>
  Require all denied
</IfModule>
<IfModule !mod_authz_core.c>
  Order deny, allow
  Deny from all
</IfModule>
```

VULNERABILITY ASSESSMENT

In order to ensure that the security controls are implemented the right way, and is efficient, multiple tests will be performed. The first test has been run with the OWASP ZAP application. The application is visible on Figure 1, all plugins have to be updated in order to include all last vulnerabilities.

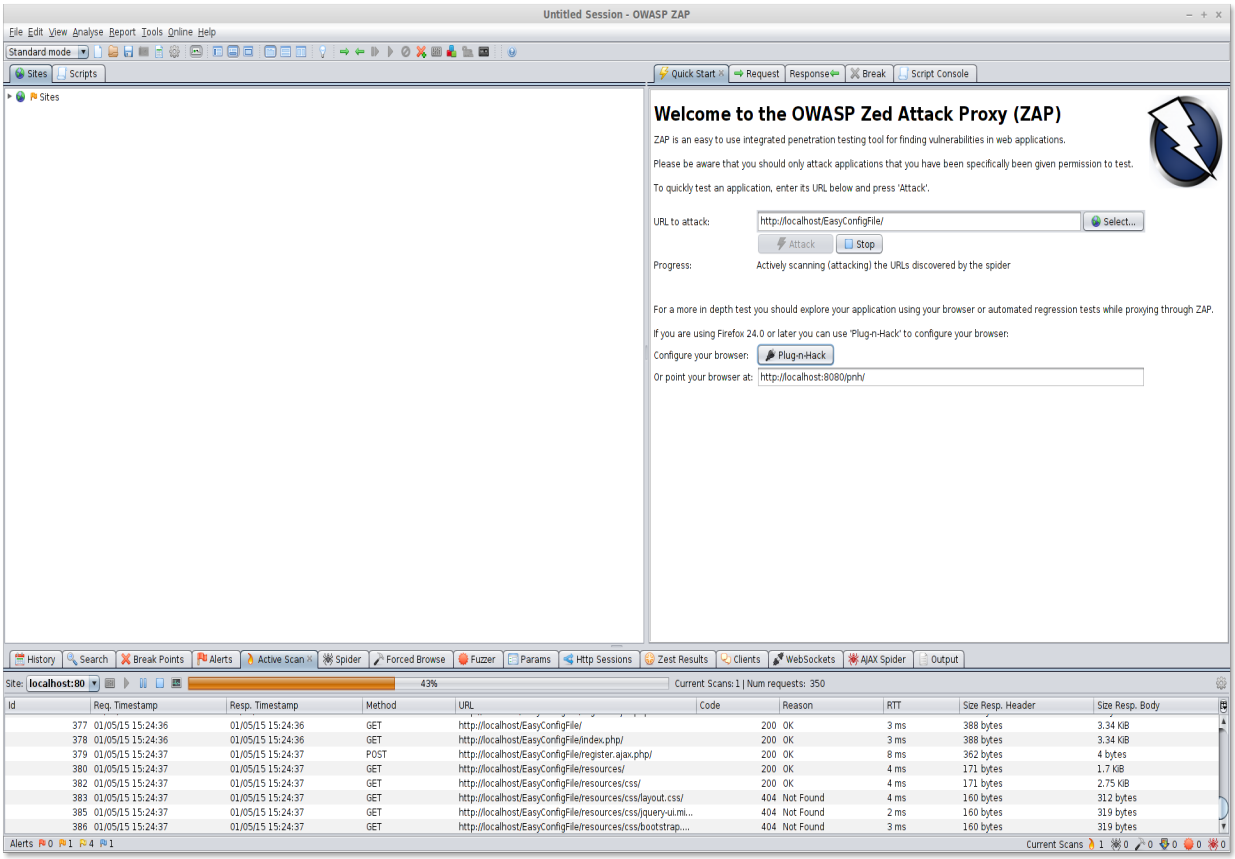

Figure 1: ZAP scanning interface.

Because we do need to be authenticated to access advanced interactions, a test account has been set up and given to the crawler, as well as the use of the Plug-n-hack Firefox plugin in order to manually ensure that the crawler got all the links.

No highly classified vulnerable issue has been found, and 65 medium issues have been reported. They all regard the server-side configuration, as a local server has been set up in order to perform the test.

Next step, the use of w3af (“web application attack & audit framework”), another penetration testing tool. It comes with profiles, one called OWASP Top 10, it is the one we are going to use, as it concerns the most important vulnerabilities (Figure 2).

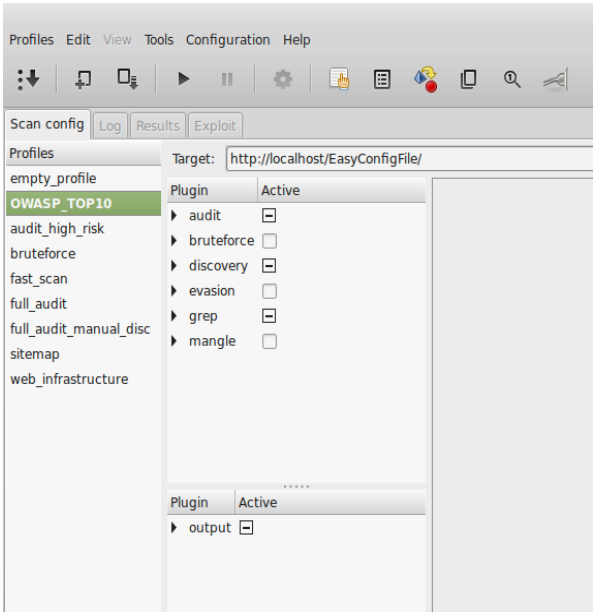

Figure 2: w3af setup.

Here as well, we have to set the credentials in order to have full access while testing the website. This is made using the HTTP settings of the profile. A full report is generated, here again it needs to be analysed, as the amount of information is huge. Once the test started, the interface is as shown on Figure 3.

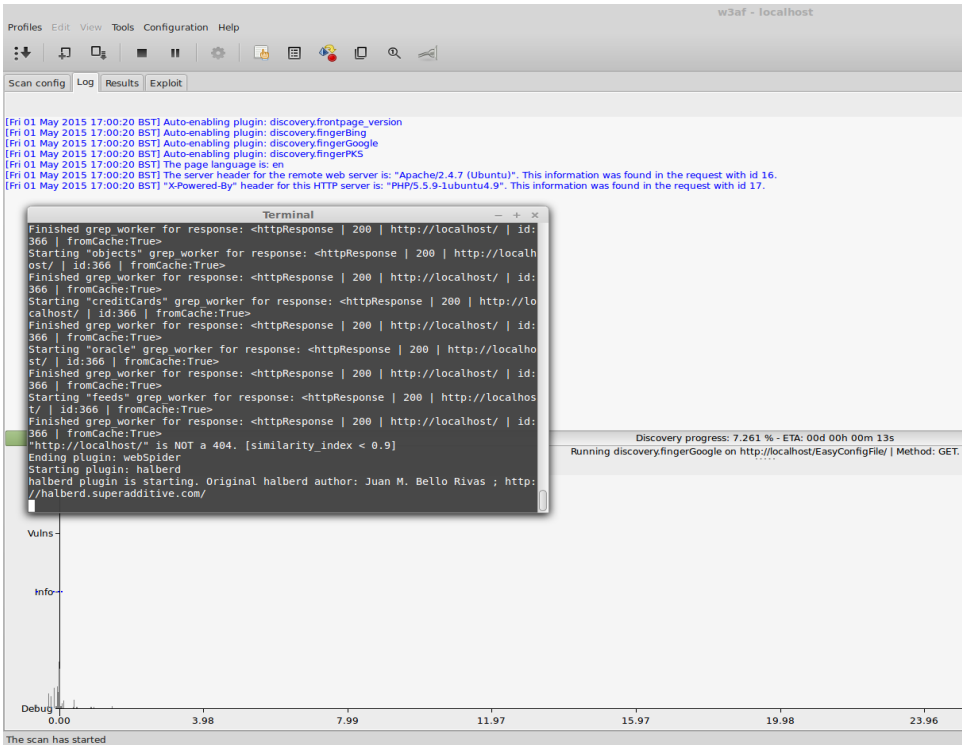

Figure 3: w3af testing gui and command.

The findings made by w3af shows 1427 vulnerabilities alert have been stored in the log, that amount sounds impressive, however, they are all false positive, such as:

The URL: "http://localhost/EasyConfigFile/resources/uploads/?C=D&O=A" possibly discloses a US Social Security Number: "48-2-6949". This vulnerability was found in the request with id 1812.

That represents nothing. Also, all are marked as Low, and the verification of found URLs confirms every time that this is a false alert. Multiple alerts are classified as Information also take part of the report, but none regards the code.

In order to continue the experiment, SQLmap has been used in order to test the security of the login and registration forms, as they are representing of the fields and they are the most exposed to the outside. Also, as we use PDO for all, it should be representative. The following command is used in order to target POST values for SQLmap:

```
python sqlmap.py --data "login=xyz&pwd=xyz" -u "http://localhost/EasyConfigFile/"
```

It has been run on the index.php a first time, in order to target the login mechanism, the second test has targeted register.ajax.php, as it is the second public form. All outputs can be summarized in that one line:

```
[CRITICAL] all tested parameters appear to be not injectable.
```

What seems to be a good point here. PDO handles injections correctly, and no parameter can be injected.

DISCUSSION

In conclusion of this experiment, we can see that a few habits can secure efficiently a web application against known vulnerabilities. Security has to be included in the development process (Scott and Sharp, 2002).

Also, in order to implement all the controls presented, note that you require an up-to-date version of PHP. It is a major point in order to prevent known vulnerabilities in the installation to be used because of a lack of upgrade.

In order to remain safe, it is also important to stay updated about vulnerabilities, and ensure security controls are enough and efficient (iMPERVA, 2013). In order to make sure of it, tests are useful and have to be performed on regular basis.

Open source databases provide a great knowledge, it is important to use them and participate as well.

In this report however, no focus has been given to the server configuration, as the logs contain multiple alerts related to it, more time could be given to extending the scope of the project in order to address server-side configuration issues and guidelines, on Apache servers per instance.

REFERENCES

- Ashish, 2015. SQL Injection Prevention by Adaptive Algorithm, *IOSR Journal of Computer Engineering*.

  [online] http://www.iosrjournals.org/iosr-jce/papers/Vol17-issue1/Version-3/E017131924.pdf [accessed on April 30, 2015]

- Bates et al., 2010. Regular Expressions Considered Harmful in Client-Side XSS Filters.

  [online] http://dl.acm.org/citation.cfm?id=1772701 [accessed on April 30, 2015]

- Huluka and Popov, 2012. Root Cause Analysis of Session Management and Broken Authentication Vulnerabilities, *World Congress on Internet Security*.

  [online] http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6280203 [accessed on April 30, 2015]

- iMPERVA, 2012. Remote and Local File Inclusion Vulnerabilities 101, *Hacker Intelligence Initiative, Monthly Trend Report #8*.

  [online] https://www.imperva.com/docs/HII_Remote_and_Local_File_Inclusion_Vulnerabilities.pdf [accessed on April 30, 2015]

- iMPERVA, 2013. Web Application Attack Report, *Edition 4*.

  [online] https://www.imperva.com/docs/HII_Web_Application_Attack_Report_Ed4.pdf [accessed on April 30, 2015]

- Provos and Mezières, 1999. A Future-Adaptable Password Scheme, *The OpenBSD Project*. USENIX

  [online] https://www.usenix.org/legacy/events/usenix99/provos.html [accessed on May 1, 2015]

- Scott and Sharp, 2002. Abstracting application-level web security.

  [online] http://dl.acm.org/citation.cfm?id=511498 [accessed on May 1, 2015]

- Top Ten Project, 2013. The Ten Most Critical Web Application Security Risks, OWASP.

  [online] https://www.owasp.org/index.php/OWASP_Top_10 [accessed on April 30, 2015]