Review of Group E by Group A

Mustapha Malik Bekkouche Yumer Adem Yumer
Steffen Mogensen Oscar Felipe Toro

May 13th 2016

Contents

1 Review of the External System

1.1 Background

Developers of the external system: *Christensen, Tore Rhsle Luntang, Cosma, Vlad Paul, Engelhardt, Alexander Grtz, Luca, Irina Alina Gabriela*

Date of the review: May 13th 2016

1.2 Completeness in Terms of Functionality

Does the system meet the requirements given in the assignment?

Table 1: Completnes

No	Completness	
1	The system should allow the user to upload pictures	yes
2	The user can share his own pictures with other named users on a pictureby-picture basis	yes
3	The user can view his own pictures and pictures other has shared with him	yes
4	The user can comment on any picture he can view	yes
5	The user can view comments on any picture he can view	yes

1.3 Architecture and Security Concepts

The architecture of the system can be described by the interaction of three components: a Postgres database to save comments and pictures, a Wildfly server wich is serving an HTTP API, and finally, in order to make use of this API, the client side use AngularJS to connect to the server, but also to interact with the client and make request to the API. The communication to the database is handled by the HTTP API wich comminicates to the database to perform CRUD operations.

We consider that the architecture for the system use a very common style of design, used in moderns applications. In this case the team should consider to harden the Ubuntu Server, the database, the WildFly server and the code for the Client Side. Since the time is very concise, they need to focus on the most critical parts of the system and that can be reflected in the risk analysis.

The risk analysis identifies database records, pictures, and comments as assets that have to be protected. Furthermore, they expect not very sophisticated attacks and natural catastrophes as potential threaths. Accordingly, the system make use of prepared statements to secure the database records, for instance.

1.4 Implementation

The countermesure against the SQL injection, and XSS are correctly implemented. However, they are using Ubuntu 14.04 which is not the latest stable version (that would be 16.04).

While looking at the solution, we found some vulnerabilities.

- **Cleartext submission of passwords (login):** -passwords are sent over unincrepted connection, this may let someone listening to the network traffic aqcuire the user's password. this would especially be dangerous if the user uses a public wi-fi. even if in this case the web service does not contain sensitive data, a lot of people re use the same passwords on different platforms and even for online banking.
- **Access to images:** Anyone can access all the shared images on the fakestagram website on www.fakestagram.com:8080/img/"imagename" even without been logged in, the user has to enter the image name he wants to see,(one can surely guess some easy ones fx: me.jpg or dog.jpg) this violates the confidentiality requirement
- **Cross-site scripting (reflected)**
  The value of the username request parameter is copied into the HTML document as plain text between tags. The payload ¡script¿alert(1)¡/script¿ was submitted in the username parameter. This input was echoed unmodified in the application's response.

  This proof-of-concept attack demonstrates that it is possible to inject arbitrary JavaScript into the application's response.

  To solve this issue, a very good way is to validate user input. For example, personal names should consist of alphabetical and a small range of typographical characters, and be relatively short; A year of birth should consist of exactly four numerals; And so on.
- **DOM based redirection:**
  The website is vulnerable for DOM-based open redirection. This type of redirection appears when a client-side scripts reads data from a controllable part of DOM like URL and processes it in unsafe way. This

vulnerability is used in fishing attacks to force the user to visit malicious sites without realizing it; it opens a door for a hacker to inject malicious code on the page. In the website data is read from document.location and passed to document.location via the following statements: var a = document.location.toString().substr(0,document.location.toString().length-1)+:8080/sec; document.location = a; The recommendation is input to be validated before redirection.

System scan
For a overall scan of the system, we have used OpenVAS 8, which have found the following vulnerabilities:

- **phpinfo() output accessible**

  Impact
  Some of the information that can be gathered from this file includes: The username of the user who installed php, if they are a SUDO user, the IP address of the host, the web server version, the system version(unix / linux), and the root directory of the web server.

  Solution
  Delete them or restrict access to the listened files.

- **php Multiple Vulnerabilities**
  Installed Version: 5.5.9

  CVE: CVE-2015-4148, CVE-2015-4147, CVE-2015-2787, CVE-2015-2348, CVE-2015-2331 CVE: CVE-2015-4026, CVE-2015-4025, CVE-2015-4024, CVE-2015-4022, CVE-2015-4021 CVE-2015-3329, CVE-2015-3307, CVE-2015-2783, CVE-2015-1352 CVE-2015-6831, CVE-2015-6832, CVE-2015-6833 CVE-2015-3330

- **php Multiple Remote Code Execution Vulnerabilities**
  CVE: CVE-2015-0273, CVE-2014-9705

- **php Use-After-Free Remote Code Execution Vulnerability**
  CVE: CVE-2015-2301

- **php Use-After-Free Denial Of Service Vulnerability**
  CVE: CVE-2015-1351

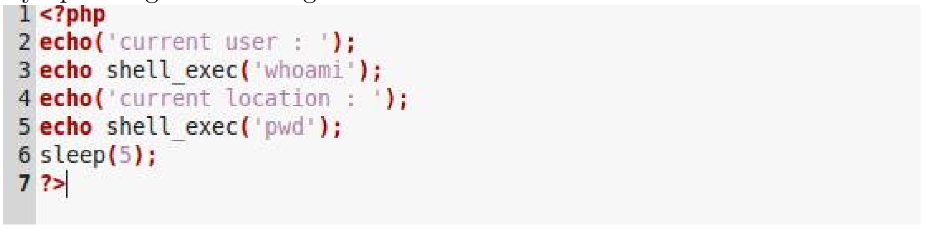
- **php 'serialize_function_call' Function Type Confusion Vulnerability**
  CVE: CVE-2015-6836

- **php 'phar_fix_filepath' Function Stack Buffer Overflow Vulnerability**
  CVE: CVE-2015-5590
- **php Multiple Denial of Service Vulnerabilities**
  CVE: CVE-2015-7804, CVE-2015-7803
- **php Out of Bounds Read Memory Corruption Vulnerability**
  CVE: CVE-2016-1903
- **Apache HTTP Server Multiple Vulnerabilities**
  CVE: CVE-2015-3185, CVE-2015-3183

1.5 Backdoors

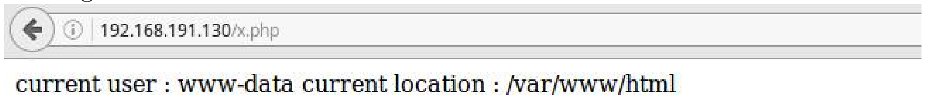
The system is running the website on a WildFly service, at port 8080. On the website (fakestagram), it is possible to upload images to the sever (and almost every other type of file), which will be saved, and can then be found at http://[IP]:8080/img/. There is also running a version of apache on the system. It is looking at the same location on the system that the files get uploaded to. Therefore it is possible to upload PHP files from fakestagram, on the WildFly service, and execute the PHP with apache.

By uploading the following PHP code:



```php
<?php
echo('current user : ');
echo shell_exec('whoami');
echo('current location : ');
echo shell_exec('pwd');
sleep(5);
?>
```

We can see from from what user that executes the code, and from where it is being done.



```
192.168.191.130/x.php
```

```
current user : www-data current location : /var/www/html
```

If defining a backdoor as a way to gain root access to the system. Then we didn't manage to exploit any.
But we do believe that by poking a bit more around in the system from the remote code execution with PHP, we might have gained root access.

1.6 Comparison

Comparing the two systems with each other, is a some similarities, such as both systems beeing Ubuntu 14.04, and the main purpose of the webapplications.

Differences between the systems are:

- **Password length**
  The reviewed system has a 10 character minimum for the password.
  Our system will allow password.

- **Technologies**
  Our system is build with Python flask, and a SQLite database. The reviewed system, is build with Java, Angular.js, Hawk and a POSTgreSQL as the database.
  The http server that we used is SimpleHTTPServer, and for the reviews system, they have used both WildFly and Apache. (Apache is only implemented for backdoors reasons).

- **Image upload**
  We have choosen to use a whitelist approach, where we have selected the file types that are allowed to be uploaded.
  They choose a blacklist approach, where they sort out the file types that they don't want the user to upload.