Journal of Mobile, Embedded and Distributed Systems, vol. VII, no. 1, 2015
ISSN 2067 – 4074

Implementing Security in a Vulnerable CRM

Alexandru Valentin BESCIU
Department of Economic Informatics and Cybernetics
The Bucharest University of Economic Studies
ROMANIA
besciualexandru@gmail.com

Abstract: This paper work objective is to scan and analyze a web application developed in early 2013 by the author of this paperwork. The application has been improved constantly since then, but without having a security plan included. Unfortunately the application arrived at a point where the security isn't optional anymore and it needs to be improved. In order to do this I scanned the web application files with Acunetix Web Vulnerabilities Scanner. After the analysis the results pointed which vulnerabilities the application has and how to fix them. After I had fixed the vulnerabilities I rescanned the application to see if there were any others which appeared because of the new code. After the scanning the results were good, Acunetix WVS showing only notices.

Key-Words: Web, Security, Vulnerabilities, Scanner, Application

1. Introduction

In the last two decades the World Wide Web growth exponential till now. In year 1995 there were about 16 million users using WWW, and by the end if the year 2013 their number growth to 4.3 billion. Because of this and because of their demands who were getting more and more complex new hardware and software technologies needed to be invented. For this paperwork I will present and describe in short terms and chronological order the web technologies which were developed in this period. These were: *HTTP*, *HTML 1, HTML 2, cookies, SSL, HTML 3, JavaScript, PHP, ASP, Flash, XML, AJAX*, and many others.
In order to be used this technologies web browsers were created. A web browser is a software which can translate each of the technologies above in web pages. They are capable to display text, images, audio, video and many other resources. They are the client in a server-client environment, the client used by internet users in order to make request to web servers.
One of the most used languages in web development is called *PHP*. *PHP* it's an open-source and server-sided scripting language. Initially it was used only to develop dynamic web pages, but since version 4.3 it can be used in *CLI* (command *line interface*) therefore allowing powerful web application to be developed.
Another language used in application development was *JavaScript*. This is again one of the top most used scripting language, but as opposed to *PHP* it client-sided. This means that in order to be processed by the client it gets downloaded to the user's machine and then executed by the browser. Because is client-sided there is no security implementation at *JavaScript* level. Security will be implied only on web server. The main reason for this is that when the *JavaScript* is downloaded on the user's machine it can be modified how the user wishes. JavaScript purpose is to ease the life and experience the user has online, and *PHP* purpose is to execute safely what the user wants.
The third and last language used in the development of the application is *SQL*, or *Structured Query Language.* By using SQL the application insert, edits and deletes record in the database. The application purpose was to manage its own user financial status and help the company to increase its own profit.

2. Web Security Vulnerabilities

At the beginning of year 2013 I designed an internal web application used by a company in order to manage financial status of their customers. This application is used to create and manage customer's

loans. When a customer comes and pays his debts to the company the application is used to update his loan status and generate payments files.
Now, in 2014 the bank manager asked me to improve application security because he wants to let user access their financial status. Since the application is installed only in internal network, I used an external database to upload user's loan status. On external database the page accessible to customers has only read attributes. On the other hand on the server to server communication, only the internal application can write to external database. In this way I ensure that no user could update his loan status how he wants.
The purpose of this article is to show the web vulnerabilities that I had found in the internal application that would allow company employees to perform internal attacks.
In order to see the web vulnerabilities I used a tool called Acunetix Web Vulnerabilities Scanner in order to analyze the application. The Acunetix WVS's report showed a critical vulnerability called cross site scripting and a few vulnerabilities with medium severity (Figure 1).

Figure 1. Acunetix WVS

In the following pages I will describe some of the web vulnerabilities I found by using Acunetix WVS, then I will describe how the exploit could be used, and then the solution to fix them. Because I didn't get the permission to publish web application vulnerabilities I used some abstract examples.

2.1 Cross-site scripting

Is also called XSS and is a web security vulnerability often found in web applications. It makes possible attacks where the malicious users could inject client-side scripts into web pages viewed and accessed by other users.
This type of vulnerability could be used by attackers in order to bypass access controls. According to a Symantec document, in 2007 cross-site scripting was accounted for 84% exploitation of websites vulnerabilities. The main reason for this vulnerability to be in top of most used is that its relative easy to do.

2.1.1 Exploitation

The best example of this vulnerability Exploitation is when a common user like Alice vists Bob's website. Bob's website is used to buy food for animals therefore it stores sensible data like account number, and others. Mallory, a malicious user has tested Bob's website and find out that when you search for something the search page returns the following message: "*The search results for the following string <<string>> are:* ". Therefore he tried to search the following string: *"cute puppies <script> alert('XSS possible')</script>"*. The result page poped out an alert dispplaying: *"XSS possible."*.
Mallory discovered the XSS vulnerability in Bob's website. Then he emailed a link to Alice where he written the following link: *"<a href='www. Bobsite.com /search_page?s=cute%20puppies<script src=`mallorysite.com/evilscript.js'></script>Cute puppies"*. As you can see in the link he provided to Alice in email is included an evils script from his website. That script will be processed by the browser without Alice knowing anything. Mallory's script downloaded a copy of Alice's session cookie and uploaded it in his own site. Now he can use it to login in Bob's website as Alice.
If Mallory would have written that link on a form inside the website, that XSS exploitation would have become persistent, therefore anyone who would have clicked that link, would have been exploited.

2.1.2 Solution

The solution to this vulnerability is to filter any input coming from the user. For example if the search page would have usen php's function htmlspecialchars() in

ISSN 2067 – 4074

order to validate the string provided by the user, the result of Mallory's exploitation would have been:

```
""&lt;a href='www.
Bobsite.com/search_page?s=cute%20pu
ppies&lt;script
src=`mallorysite.com/evilscript.js
'&gt;&lt;/script&gt;Cute
puppies&lt;/a&gt;".
```

As you can see all malicious chars that are used to declare an HTML tag are gone. This function basically converts all HTML special characters in string.

2.2 Application error message

This vulnerability appear when one page has an error message which display sensitive information about the web application used. This thing occurs when the developer forgets to stop all error's warning, notices, fatal errors and so on. This thing could be done by adding a code line to php file, or by setting it inside php configuration file.

2.2.1 Solution

In order to prevent this type of vulnerability you should always set the php.ini file when moving the application to production. If you don't want to alter the php.ini file or you don't have access to modify it you could use one of PHP's functions to set error reporting level. The function used to do this in PHP is "*error_reporting(0);*".

2.3 Directory listing

This vulnerability appear when the web server is configured to display the list of files contained in every directory. This setting is not recommended because the directory may contain files that are not normally exposed through links on the web site. In this way one malicious user could use the "remote file inclusion" vulnerability in order to study the page's behavior.

2.3.1 Exploitation

This kind of vulnerability is very easy to exploit because the only thing needed to be done to check if Directory Listing is enabled is to check a basic directory like images directory.
The exploiter could try to access different links often used by web developers like *"www.website.com/images"* or *"www.website.com/config/".* If he can see an web page which is listing all directory files then Directory Listing option is enabled.

2.3.2 Solution

As simple is to check for this vulnerability as simple is the solution against it. In the *.htaccess* file you have to add some lines which specify which directory you want to protect, and inside that line's tag you have to specify "Options –Indexes" (Figure 2). This option removes automatic index generation in directories where aren't index files.

```
205 <Directory "D:/www/apache
206     Options -Indexes
207     AllowOverride None
208     Order allow,deny
209     Allow from all
210 </Directory>
```

Figure 2. Directory Listing Solution

On the other hand you could add an empty index file in each directory your website has, or an index file which says: access to this files is strictly prohibited.
By doing one of these two options you will protect your website against Directory Listing vulnerability.

2.4 HTTPS not used

By default when HTTP protocol is used all request and responses are sent in clear text. Therefore an malicious user could listen the network for this packets. Whenever he will see username and password fields he will also see the destination server address. From that moment he could use your credentials in order to access your account on that web page. To avoid this thing and to improve security, web developers are advised to use HTTPS protocol over HTTP because the first one use encryption to hide data contents (Figure 3).

Figure 3. The protocol used to send data

2.4.1 Exploitation

For the following example let's assume that Alice wants to use the company application in order to do her daily tasks. She logs in and starts working. Mallory, her malicious colleague is sitting at his desk, using the same network as Alice does, and listen network packets. To do this he installed Wireshark application that has the ability to set network card to listen all incoming packets. When Alice logged in the application using HTTP protocol she sent some unecrypted sensible information to server and whoever was listening the network. Mallory saw her unencrypted packet and checked it to search for her username and password. The bad thing is that he found them in clear text. From now on he could use them in order to log in the application.

2.4.2 Solution

To avoid sending user credentials in clear text you will have to use HTTPS protocol instead of HTTP. In order to do this you have to have Apache Web server with OpenSSL library installed. Then you set up in *httpd.conf* file the following line: *"include conf/extra/httpd-ssl.conf"*. This line tells the Apache web server to use the configuration file of *httpd-ssl.conf*. In order to see some changes you have to complete the *httpd-ssl.conf* file so he could use your certificate to encrypt data. For internal applications users could use self-signed certificates. For online web application is recommended to buy an certificate signed by Root CA's.
After you do this you must restart the Apache Web server so that he could start with his new options. In order to take advantage of *HTTPS* you should access the web application by using *https* instead of *http* in URL. If you want to do this automatically you either set up this redirection inside apache web server configuration file, or as a developer you write a line in PHP file that redirects the user to the https page instead of http one.

2.5 Clickjacking

A click jacked page tricks a user into performing undesired actions by clicking on a concealed link. On a click jacked page, the attackers load another page over it in a transparent layer. The users think that they are clicking visible buttons, while they are actually performing actions on the hidden page.
The hidden page may be an authentic page; therefore, the attackers can trick users into performing actions which the users never intended (Figure 4).

Figure 4. Example of clickjacking attack

There is no way of tracing such actions to the attackers later, as the users would have been genuinely authenticated on the hidden page.

2.5.1 Exploitation

For example let's imagine an attacker who builds a web site that has a button on it that says "*You've won an iPhone. Click here to claim it*". However, on top of that web page, the attacker has loaded an invisible iframe with your Facebook account, and lined up exactly the "*Disable account*" button directly on top of the "*claim button*". The victim tries to click on the "*claim button*" button but instead actually clicked on the invisible "*Disable account*" button. In essence, the attacker has "hijacked" the user's click, hence the name "Clickjacking".

ISSN 2067 – 4074

2.5.2 Solution

The solution of this attack is also related to the web server configuration file and can be resolved by adding a simple command to set the X-Frame Option in the header of HTTP requests. This way you will be able to see any hidden frame that wants to trick you without even noticing of their existence, you will also need to restart your web server after making this modification in the configuration file.

2.6 Session cookie unflagged

Session cookie without HTTPOnly and Secure flags properly configured are the reason for this type of vulnerability.
A cookie is basically a method to transport data from the user to the server and it's active only while the user is online on that web server, so if the cookie is set to make this transportation in a secure layer with HTTPS then you will have no issues but if the cookie is send over HTTP without any kind on encryption then the data will be easily listened to and for sue the attacker will be able to get that sensitive data of the user on this unsecure environment.

2.6.1 Exploitation

This kind of vulnerability is correlated with XSS attacks. If you have read the XSS subsection above, then add to that example the next scenario. If Mallory's malicious script tried to access a cookie secured with HTTPOnly flag, then it couldn't because cookies with this kind of flag set cannot be accessed by JavaScript code.

2.6.2 Solution

According to Michael Howard, Senior Security Program Manager in the Secure Windows Initiative group at Microsoft, the majority of XSS attacks target theft of session cookies. A server could help mitigate this issue by setting the HTTPOnly flag on a cookie it creates, indicating the cookie should not be accessible on the client.
If a browser that supports HTTPOnly detects a cookie containing the HTTPOnly flag, and client side script code attempts to read the cookie, the browser *returns an empty string* as the result. This causes the attack to fail by preventing the malicious (usually XSS) code from sending the data to an attacker's website.

2.7 Slow response time vulnerability

If one page has a big response time, like more than 10 seconds, that page could be used by malicious users to launch DoS attacks. DoS attacks or Denial of Service can be used to overload servers with many requests. If someone has this kind of pages on his web page it should restrict access to it only to trusted persons. In this way no malicious user could exploit the big response time vulnerability and launch DoS attack on your web server.

2.8 Auto-complete Enabled

Auto complete Enabled vulnerability is one of the most seen vulnerabilities on the internet because a lot of beginner programmers don't disable this option to their portal or application without even thinking for a moment what can this do to their users or clients.

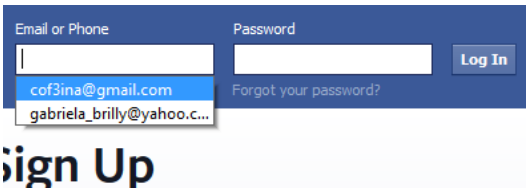

Figure 5. Example of autocomplete function

For example (Figure 5) when you access a page with this option enabled it means that if your clients will leave his laptop for a moment and an attacker change the type of password filed to an unsecure text, then the password of your client will be shown in clear text and the attacker can login using your client credentials.

2.9 Weak Session Management

Weak Session Management is also a common vulnerability seen on the internet, and it means stealing a session information or cookie from a user to get you as an attacker to login using their credentials.
There are four methods that an attacker

could use in order to steal someone's session cookie. These methods are session fixation, simple copy of cookie file when having direct access to victim's files, by using cross-site scripting, and the last method used is listening the network. An example of this vulnerability is when an attacker use a cross-site scripting in order to steal someone's cookie and then use it as it own. After this step the attacker can be easily logged in the application with victim's credentials.

3. Conclusion

As you already expect to see (Figure 7), after correcting all the vulnerabilities noticed by Acunetix WVS, I rescanned the whole application. The result was very satisfying as the new improved application has been cleaned up.

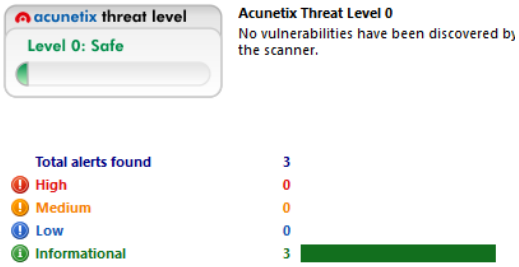

Figure 6. Acunetix Thread Level 0

The only thing I could recommend in the matter of web vulnerabilities is to check once a month your web applications for vulnerabilities, read speciality articles and why not, try to break your own application from time to time. You may succeed and find out new ways to break in. Only by thinking as an attacker, you could protect against one.

Acknowledgement

Parts of this paper were presented at The 7th International Conference on Security for Information Technology and Communications (SECITC 2014), Bucharest, Romania, 12-13 June 2014.

References

[1] Bryan Sullivan, *Web Application Security, A Beginner's Guide, 2011, ISBN-13: 978-0071776165, pp. 40-52*

[2] Steven Palmer, *Web Application Vulnerabilities: Detect, Exploit, Prevent, 2007, ISBN-13: 978-1597492096, pp. 150-180*

[3] Joel Scambray, *Hacking Exposed WEB Applications, 2010, ISBN-13: 978-0071740647, pp. 86-114*

[4] Dafydd Stuttard, *The Web Application Hacker's Handbook, 2007, ISBN-13: 978-0470170779, pp 530-600*

[5] Jon Erickson, *The Art of Exploitation, 2nd Edition, 2008, ISBN-13: 978-1593271442, pp. 25-40*

[6] Kevin Beaver, *Hacking for Dummies, 2004, ISBN-13: 978-0764557842, pp. 80-133*