Secure Coding - Team 7- Phase 5

Magnus Jahnen, Thomas Krex, Elias Tatros

January 25, 2015

Part I

Executive Summary

Contents

Part II

Time Tracking Table

Chapter 1

Time Tracking Table

Time Tracking		
Name	**Time**	**Description**
Magnus Jahnen	15h	Reverse engineer batch parser (team 8)
	2h	Reverse engineer batch parser (own)
	5h	Reverse engineer Java-SCS (team 8)
	2h	Reverse engineer Java-SCS (own)
	10h	Meetings
	5h	Report & Presentation
Thomas Krex	3h	Self introduction into test environment and target application
	5h	Searching for Vulnerabilities in PHP&JavaScript Code and of Team 7
	11h	Testing own app according owasp checklist
	6h	Exploiting Proccessing Time and Account guessing vulnerabilities
	10h	Meetings
	6h	Documentation
Elias Tatros	2h	Static Analysis decompiled Java and PHP
	4h	Finding Encryption Flaws in Java/PHP
	5h	Analysis of application memory
	15h	Planning, Implementation and Testing of Memory Scanner
	6h	Working on Report (Sections on Key Weakness, Memory Scanner, Static IV)
	10h	Meetings

Part III

Application Architecture

Part IV

Security Measures

Chapter 2

Security Measures

2.1 PDF Password Protection

The TAN List of our customers are password protected. That ensures that only customer himself can access the transaction codes. To porotect the pdfs we used thethird party library FDPI.

2.2 HTTPS

Our webservice uses the HTTPS protocoll to encrypt all transmitted data to and from our clients. This prevents attackers from reading sensitive data via man-in-the-middle attacks.

2.3 Strong password policy

We enforce the user to choose a string password to minimize the possible threat based on brute force attacks.

2.4 Secure Session Management

The session cookie of an user is only transmitted via HTTPS, can not be accessed via javascript and expires after 30 minutes. So attackers cannot bypass the authentication schema by stealing cookies of logged in users.

2.5 Prepared Statements for SQL database

We protected our web service against sql injections by using prepared statements for database requests.

2.6 CSRF Token

2.7 Protection of C Parser against Heap/Buffer Overflows

To our knowledge there exist no possibilities for heap or stack overflows, since we tried to avoid allocating space on the heap and used secure functions for copying buffers. Also we add the compiler flag *-fPIE* and *-fstack-protector-all* in gcc to randomize address locations and a canary, making it harder for an attacker to cause buffer overflows.

2.8 Secure TAN Generation

To generate secure transaction codes for single or batch transfers, we use the hash algorithm sha256 in combination with a current timestamp as seeed. Time syncrhonization is done via a NTP server. The timestamp is concatenated with information of one transfer, including the destination account, the amount that will be transferred and the personal PIN of the user. A hash of this information is than trimmed to 15 digits. Since the timestamp is part of the TAN, one transaction code is only valid for at maximum 2 minutes, because the webservice only generate two tans for verifcation. One with the current timestamp and one with a timestamp one minute in future. Therefore transaction codes which an attacker got access to in some way, are useless in less than 2 minutes. That minimizes the risk of unauthorized transactions.

2.9 Secure Password Recovery

The Password Recovery for users is done in 3 steps. The user requests a new password at the website. Then he receives an email with n personalized URL. By clicking on it , the user gets to the page where he have to select a security question and the corresponding that was defined at the registration. The answer to one of this question was given at the registration. Now he can enter his new password.

2.10 Obfuscation of SCS

The Byte Code of the java application SCS.jar was obfuscated with ProGuard in order to complicate the reverse engineering. In Listing 2.1, the corresponding ProGuard config is shown.

Listing 2.1: Configuration File of ProGuard

```
-injars /Volumes/MacintoshHD/Users/mep/Downloads/scs/scs.jar
-outjars /Volumes/MacintoshHD/Users/mep/Downloads/scs/scs_out.jar
```

```
-libraryjars
    /Library/Java/JavaVirtualMachines/jdk1.7.0_71.jdk/Contents/Home/jre/lib/rt.jar

-libraryjars
    /Library/Java/JavaVirtualMachines/jdk1.7.0_71.jdk/Contents/Home/jre/lib/jfxrt.jar

-dontshrink
-dontwarn com.javafx.**
-dontwarn org.apache.**

-keepattributes '*Annotation*'

-adaptresourcefilecontents **.fxml
-keepclassmembernames class * {
       @javafx.fxml.FXML *;
}

-keepclasseswithmembers public class com.javafx.main.Main, scs.Main {
       public *; public static *;
}


# Keep - Applications. Keep all application classes, along with their
    'main'
# methods.
-keepclasseswithmembers public class * {
       public static void main(java.lang.String[]);
}
```

2.11 Secure Handling of uploaded files

Uploaded Files for batch transaction are handled in a secure manner to avoid damage of uploaded malicious code. This includes the usage of the default PHP temporary folder. All files are stored in /tmp and get a randomly chosen name. This has two effects. First of all files are not stored at the webroot and can not be access via an unsecure webserver. Secondly, renaming files before passing them to the c parser, prevents malicous code in the filename to be executed. Another precoution we took, is to delete this files right after parsing them, so that malicous code can't be executed later on.

Part V

Fixes

Chapter 3

Fix of HTTP Strict Transport Security

3.1 Affected Files

/etc/apache2/httpd.conf: line 15

3.2 Description

We added an HTTP Scrict Transport Security Header to teh config of our web server. Therefore the server notifies the client's browser that all traffic has to be exchanged via HTTPS. To do so the added the following line to the Virtual Host Config:

Header add Strict-Transport-Security "max-age=15768000"

Chapter 4

Fix for Bypassing Session Management Schema and Cookies attributes

4.1 Affected Files

- /etc/apache2/httpd.conf lines: 2++

4.2 description

In order to protect the session cookie and its attributes against attackers the added the following settings to httpd.conf:

```
php_value session.cookie_httponly true
php_value session.cookie_secure true
php_value session.cookie_lifetime 1800
```

The httponly flag was already set in phase 3 and prevents that the cookie can be acessed by javascript. The http_secure flag ensures that cookies are only sent via HTTPS.
Futhermore the attribute cookie_lifetime defines the expire data for a cookie. A short lifetime decreases the chances for an attacker to sucessfully use a foreign session cookie to authenticate with the web service.

Chapter 5

Fix Process Timing

5.1 Affected Files

- webroot/pw_recovery.php: 137++

5.2 Description

Our App was leaking information by its process timing. To be more precise, the vulnerability was placed in the password recovery service. The User can enter his/her email address and will retrieve an email with an new password. The problem was the time, the system took to send that email. If entering an email which isn't registered in the system, the process takes round about 30 ms instead of 3 seconds. This means that an attacker can guess regitered email addresses my observing the process time. To fix this problem there is more than one solution. You could start an asynchronous task that is sending the mail while the web service reponds to the client. For that you have to use external libraries that would increase the complexity of the system. Another way would be to send a curl request to an internal web service. This basically would work but it requires a static host name to do so. So we decided to chose a third solution. We inserted a sleep command that will wait a randomly picked interval between two and four seconds if the email address is not known to the system. This prevents an attacker of easily guessing registered email adresses.

Listing 5.1: Random Timeout in pw_recovery.php

```
if($user->email) {
// if we found the mail it is a valid user
$user->sendPwRecoveryMail();
}
else {
// timeout , so it's not that easy to guess existing accounts
    via the processing time.
```

```
$timeout = rand(2,4);
sleep($timeout);
}
```

Chapter 6

Fix of TAN Generation

6.1 Affected Files

- c_user.php : line 324++
- helper.php : line 216

6.2 Description

Our app offers the possibility to use a SmartCardSimulator to generate transaction codes for single or batch transactions. This code can then be entered at our website or in the batch transaction batch file. To verify the TAN, the webservice is using the same computations as the SCS. This includes the hashing of a string containing the current time as seed, the destination account, the amount and the pin of the user. This String was hashed with md5. This was a vulnerability because md5 is not secure in terms of collision attacks. So we changed the hashing algorithm to sha256 which generates a 256 bit long hash instead of 128 bit. To receive the same results as the java code and the c code for parsing batch files, we had to modify the hashing, shown in Listing 6.1. The hash function returns a string with a 128 bit hexadecimal number. 16 of this 32 characters are pairwaised concatenated and transformed into decimal numbers. Because this process ingores possible signed numbers, each decimal number greater than 127 and to be substracted by 256. Of these rsulting numbers the absolute values of the first 16 numbers is concatenated and trimmed to 15 characters.

Listing 6.1: Generation of a TAN using sha256

```
function generateTanWithSeed($seed,$pin,$destination,$amount){

$plaintext = $seed.$pin.$destination.$amount.$seed;
```

```
//$hash = $this->generateMD5Hash($plaintext);
$hash = hash('sha256',$plaintext);
$hash_array = array();
for ($i=0;$i<16;$i += 2) {
$tmp = substr($hash, $i,1).substr($hash, $i+1,1);
array_push($hash_array, hexdec($tmp));
}
$hash_string = "";
for ($j=0;$j<16;$j++) {
$tmp = $hash_array[$j];
if ($tmp > 127) {
$tmp -= 256;
}
$hash_string .= strval(abs($tmp));
}
return substr($hash_string,0,15);
}
```

The next problem was the unavailbility of the time server which was used to synchronize the SCS and the webservice. Also it ensures that transaction codes for an transaction with the same amount and destinations differs over the time. This results in a stronger protecion against brutforcing the transaction codes. To fix this problem we simply changed the time server from that the time is requested, shown in Listing **??**

Listing 6.2: New TimeServer IP

```
function getUTCTime(){
$timeserver = "129.6.15.28";
```

6.3 Fix for Upload of Unexpected File Types

Our app allows the user to upload any kind file for the batch transaction. But in our point of view that is nota vulnerability. Uploaded Files are not stored in the webroot of the server and are renamed before they are stored. Furthermore uplaod files are directly deleted after being parsed by the c program. If an attacker woud have the possibility to to execute his code, the extension of the file would not make any difference. To sum it up, allowing uploading any file extension increases the comfort of the user and has no influence on the security of our web service.