Secure Coding Phase 3

Team 7

Magnus Jahnen, Thomas Krex, Elias Tatros

Use Cases

- Fixed
  - Multiple transfers in batch file
  - Balance + Initialisation through employee
  - Name
- New
  - Password Recovery
  - Encrypted PDF with TANs
  - SCS

Live Demo

Account #6664117044

#	Source	Src-Name	Destination	Dst-Name	Amount	Valid	Description	Time
1	*6664117044*	First User	7911353165	Second User	**1000**	yes	blah	2014-12-01 12:09:33
2	*6664117044*	First User	7911353165	Second User	**20000**	yes	blah	2014-12-01 12:10:18
3	*6664117044*	First User	7911353165	Second User	**15000**	yes	too much	2014-12-01 12:14:02
4	*6664117044*	First User	7911353165	Second User	**14000**	yes	too much	2014-12-01 12:14:49
5	7911353165	Second User	*6664117044*	First User	**50000**	yes	test	2014-12-01 12:28:46

Time Tracking Table

Name	Task	Time
Magnus Jahnen	Adapt batch parser	5
	Java SCS	5
	PW Recovery	3
	Presentation	2
Thomas Krex	Encrypted PDF TANs	5
	PHP SCS	3
	Download SCS	4
	Various Fixes	3
Elias Tatros	Fix Vulnerabilities	6
	Added balance	2
	Various Fixes	4
	Presentation	3

General Information

- URL: https://<ip>/
- Employees:
  - tattioff42+admin@gmail.com:Password0
- Users:
  - tattioff42+user1@gmail.com:Password1
  - tattioff42+user2@gmail.com:Password2

Register

- Goal: Get user account at the online bank
- Actors: Client / Employee
- Pre-conditions: you have not registered yet with your email address
- Main Course of Execution: Enter a valid email, your name, the same strong password twice, select your status. If your status is ‘Client’ select your TAN method, then click on register
- Alternate Courses: none
- Exceptions: invalid email-address, account with email-address already exists, password not strong enough, password confirmation not equal to password, Name invalid
- Post-Conditions: Actor has a user account & a initial bank account, client gets an *Registration Successful* email with his PIN, which can be used for opening the TAN PDF file or for generating TANs with the SCS depending on the TAN method chosen
- Data formats used: String for email, name, password (at least 8 characters, one upper case, one lower case and one number), status and TAN method represented by bit

Login

- Goal: Get Access to the personal bank data
- Actors: Client / Employee
- Pre-conditions: Client / Employee is registered in the system and registration was approved by Employee
- Main Course of Execution: Enter email and password and click on Login
- Alternate Courses: none
- Exceptions: Invalid Email/Password or account is not activated yet
- Post-Conditions: Client is redirected to his main page with all his bank account/ Employee is redirected to a overview of new registration and transfer wich has to be approved
- Data formats used: String for both email and password (at least 8 characters, one upper case, one lower case and one number);

Logout

- Goal: Logout from website
- Actors: Client / Employee
- Pre-conditions: Client / Employee is currently logged in
- Main Course of Execution: Click in logout in the top right corner
- Alternate Courses: Let your session be expired (timeout)
- Exceptions: none
- Post-Conditions: Actor is now logged out and has no access to the system
- Data formats used: none

View account details: Client

- Goal: View details of an account (Balance, available funds)
- Actors: Client
- Pre-conditions: Client is logged in, selected a bank account
- Main Course of Execution: Click on Account in the top left corner, and select an account
- Alternate Courses: Login with your credentials, after that you will land on that page per default
- Exceptions: no bank account selected yet
- Post-Conditions: Client sees his account balance
- Data formats used: Double for balance and available funds

View transaction history: Client

- Goal: View transaction history
- Actors: Client
- Pre-conditions: Client is logged in, selected a bank account
- Main Course of Execution: Click on History in the top left corner
- Alternate Courses: none
- Exceptions: no bank account selected yet
- Post-Conditions: Client sees the transaction history of his selected bank account including source account number and name, destination account number and name, amount, status and time
- Data formats used: String, Double, Boolean

View transaction history: Employee

- Goal: View transaction history and account details of clients
- Actors: Employee
- Pre-conditions: Employee is registered and approved
- Main Course of Execution: Click on Users in the navigation panel, select user from listing
- Alternate Courses: none
- Exceptions: none
- Post-Conditions: Employee sees list of client accounts and their transactions
- Data formats used: Double for Account balance and Available Funds

Download SCS

- Goal: Download the SCS TAN generator to be able to generate TANs
- Actors: Client
- Pre-conditions: Client is registered, chose SCS as TAN method at register page, logged in and is on the account details page
- Main Course of Execution: Click on the *Download SCS* button, download the zip file to your desired directory and unzip it
- Alternate Courses: None
- Exceptions: None
- Post-Conditions: The SCS is now on the local computer and can be used (Java 7 + JavaFX needed to execute the *scs.jar*), the PIN is also located in the zip file
- Data formats used: Zip file as download

Generate TAN (Single transaction)

- Goal: Generate TAN to transfer money
- Actors: Client
- Pre-conditions: Client is registered, chose SCS as TAN method at register page, logged in and has downloaded the SCS generator
- Main Course of Execution: Start the scs.jar (Java 7 + JavaFX needed), enter the PIN you received via email, the destination account number and the amount to transfer, then click *Generate TAN*
- Alternate Courses: None
- Exceptions: Destination, amount or PIN invalid
- Post-Conditions: The TAN was generated, you can now click *Copy TAN* to copy it to the clipboard and paste it into HTML form
- Data formats used: String(10) for destination, String(6) for PIN, Double for amount, String(15) for TAN

Generate TAN (Batch transaction)

- Goal: Generate TAN to transfer money
- Actors: Client
- Pre-conditions: Client is registered, chose SCS as TAN method at register page, logged in and has downloaded the SCS generator
- Main Course of Execution: Start the scs.jar (Java 7 + JavaFX needed), enter the PIN you received via email, click on the batch tab and specify the batch file via clicking on the search button, then click *Generate TAN*
- Alternate Courses: None
- Exceptions: PIN or file invalid (File cannot be opened, File contents are invalid)
- Post-Conditions: The TAN was generated, you can now click *Copy TAN* to copy it to the clipboard and paste it into the batch file
- Data formats used: String(10) for destination, String(6) for PIN, Double for amount, String(15) for TAN

Transfer money via HTML form

- Goal: Transfer Money
- Actors: Client
- Pre-conditions: Client is logged in, selected an account and clicked on Transfer
- Main Course of Execution: Enter destination bank account, the amount and the right TAN (either SCS or PDF) and click on submit
- Alternate Courses: None
- Exceptions: Entered non-existing bank account, wrong format of amount, wrong TAN, account balance / available funds too low
- Post-Conditions: Transfer was committed , message of completion is displayed, transfer will appear in history
- Data formats used: String(10) for bank account, double for amount and String(15) for TAN, INT for TAN number

Transfer Money via batch file

- Goal: Transfer Money via a text file
- Actors: Client
- Pre-conditions: Client is logged in, selected an account and clicked on Transfer ->“upload file“
- Main Course of Execution: Fill out all required information, upload file via html form
- Alternate Courses: none
- Exceptions: Entered non-existing bank account, wrong format of amount, wrong TAN, file upload failed, account balance / available funds to low
- Post-Conditions: Transfer was committed , message of completion is displayed, transfer will appear in history
- Data formats used: File with: String(10) for bank account, double for amount and String(15) for TAN, Int for TAN number

Approve registration of Client/Employee

- Goal: Approve registration of Client / Employee
- Actors: Employee
- Pre-conditions: Employee is logged in, approvable registrations are available
- Main Course of Execution: Click on „Approve“, check the desired registrations, enter a valid balance in the text field and click on „Approve“
- Alternate Courses: None
- Exceptions: None
- Post-Conditions: Client / Employee can now access the system
- Data formats used: Double for the initial account balance

Approve transfer larger than 10.000€

- Goal: Approve transfer larger than 10.000€
- Actors: Employee
- Pre-conditions: Employee is logged in, approvable transactions are available
- Main Course of Execution: Click on „Approve“, check the desired transactions and click on „Approve“
- Alternate Courses: None
- Exceptions: None
- Post-Conditions: Transfer is now approved
- Data formats used: None

Password Recovery

- Goal: Get a new password if the current password was lost
- Actors: Employee / Client
- Pre-conditions: Employee / Client is registered but not logged in
- Main Course of Execution: Click on the landing (login page) on *Forgot Password?* Enter your email address and click on *Send new Password* Check your emails and click on the link in the recovery email. After that you can login with the new password
- Alternate Courses: None
- Exceptions: Recovery id is invalid
- Post-Conditions: Employee / Client can now login with the new password specified in the email
- Data formats used: String for email, recovery id and password

Vulnerability 1: Description

- On the User‘s Index page a hidden field, containing the users account numbers was used to switch between different accounts

- If you knew the 10-digit account number of a different user, you could alter the post request (e.g. Via Tamper Data) to include this number

- You could then technically switch to that users account, although no transactions were possible

Vulnerability 1: Showcase

Account Transfer History | Account Balance: 50000 | Available Funds: 50000 | tattioff42+user1@gmail.com Logout | Account: 6664117044

Welcome, *tattioff42+user1@gmail.com*. Below is a list of your accounts.

You can *click* on any of your accounts to *select* it. The *active* account is *marked in orange* and indicated in the *top right corner*.
You can also *create a new account* by clicking on the *Create new Account* button.

6664117044 Create New Account

Tamper Popup

http://localhost/account/index.php

Request Header Name	Request Header Value
Host	localhost
User-Agent	Mozilla/5.0 (Windows
Accept	text/html,application/
Accept-Language	en-US,en;q=0.5
Accept-Encoding	gzip, deflate
Referer	http://localhost/acco
Cookie	PHPSESSID=trtlrb7uu

Post Parameter Name	Post Parameter Value
CSRFToken	456d0fb8e075c0c4db
accountNumber	7911353165
selectAccount	6664117044

Options Help

Show All

thod | Status | Content Type | URL | Load Flags

Response Header Name | Response ...

Vulnerability 1: Problem/Solution

- The Problem:
  - Our application did not sufficiently verify the post data, assuming the user would not tamper with the data and submit valid account numbers of other users.

- The Solution:

Session Error: Account mismatch detected.
You have been logged out. Click here to log back in.

  - Our application now strictly verifies all post data. This includes verifying that a given account belongs to the user that is currently logged into a given Session.
  - If application detects tampered data (e.g. account that does not belong to the logged in user) it raises a session error (can be logged) and ends the session.

Vulnerability 1: Specifics

- Important adjusted Files/Lines:
  - account/index.php (line 72++)
  - account/transfer.php (line 48++)
  - account/history.php (line 37++)
  - account/upload.php (line 54++)
- Code:

```php
/* Validate that given account number belongs to this user */
$accounts = $user->getAccounts();
if( in_array( $selectedAccount, $accounts ) ) {
    $_SESSION['selectedAccount'] = $selectedAccount;
} else {
    /* Possible malicious activity: Account does not belong to user
     * Raise Session Error and close the session
     */
    $_SESSION['error'] =  "Account mismatch detected.";
}
```

Vulnerability 2: Description

- The source code and binary of the C-program were available on /var/www in a public directory. These files also contained database credentials for the root user.

Vulnerability 2: Problem/Solution

- The Problem:
  - Source code and binary in public directory
  - Root DB user credentials available in source file

- The Solution:
  - We moved the binary to /bin/ and changed the permissions to execution only by apache. We deleted the source code.
  - We created a new database user that has only access to the mybank db and limited his permissions to the operations required by our application.

Vulnerability 3: Description

- PHP Cookie that is storing the Session ID is not using HTTP-Only flag. Potentially makes Session Hijacking easier, in combination with other attacks.
- Potential XSS possible using hidden fields that are used to store the selected account (see Vulnerability 1). Requires user to be logged in and submit special form with tampered post data on the attackers website.

Vulnerability 3: Problem/Solution

- The Problem:
  - Http-Only flag not set for PHP Cookie
  - Insufficient validation of post data for selected account field

- The Solution:
  - We forced the Http-Only flag to be set for all cookies served by our Server
  - As mentioned in the solution for Vulnerability 1, we strictly validate all post data, this includes the account number. Invalid data results in a session error and forced logout.

Vulnerability 3: Specifics 1/2

- Important adjusted Files/Lines:
  - account/index.php (line 72++)
  - account/transfer.php (line 48++)
  - account/history.php (line 37++)
  - account/upload.php (line 54++)
- Code:

```
/* Validate that given account number belongs to this user */
$accounts = $user->getAccounts();
if( in_array( $selectedAccount, $accounts ) ) {
    $_SESSION['selectedAccount'] = $selectedAccount;
} else {
    /* Possible malicious activity: Account does not belong to user
     * Raise Session Error and close the session
     */
    $_SESSION['error'] =  "Account mismatch detected.";
}
```

Vulnerability 3: Specifics 2/2

- Important adjusted Files:
  - Httpd.conf

- Code:

```
557  <IfModule php5_module>
558      php_value session.cookie_httponly true
559  </IfModule>
```

▼5 cookies

Name	PHPSESSID
Value	ahpeq5fpijkbinboera0b0qf47
Host	localhost
Path	/
Expires	At end of session
Secure	No
HttpOnly	Yes

Delete... Edit...

Vulnerability 4: Description

- As already hinted in Vulnerability 1 and 3 it is possible to perform a CSRF attack on the insufficiently validated account selection field.

- By baiting the user into submitting a special form (with malicious post data) on the attackers website a user that is currently logged into the banking system may execute malicious javascript in the context of the banking website.

Vulnerability 4: Problem/Solution

- The Problem:
  - Although our application only uses only post requests instead of gets, the application does not verify the source of post data.

- The Solution:
  - We attach an anti-CSRF token to all of our forms. This token is generated for each session and its existence & correctness is verified upon submission of all post requests. Since external websites have no knowledge of the token value, they can no longer successfuly submit post requests in a CSRF attack.

Vulnerability 4: Specifics

- Important adjusted Files/Lines:
  - All files that use a session and any forms.
  - All pages in account/*.php and employee/*.php
  - On the login and register page we generate & verify anti-CSRF tokens for each request

- Example (index.php):
  - 1.)

```php
/* Generate Form Token (valid for this session) */
if (!isset($_SESSION['CSRFToken'])) {
    $_SESSION['CSRFToken'] = generateFormToken();
}
```

  - 2.)

```php
/* Check presence & validity of CSRF Token */
if (isset( $_POST['CSRFToken']) && validateFormToken($_POST['CSRFToken'])) {
    /* Perform Account Creation */
    $accNumber = randomDigits( 10 );
    $user->addAccount( $accNumber );
} else {
    $_SESSION['error'] = "CSRF Token Invalid.";
}
```

Vulnerability 5: Discussion 1/2

- The user is required to enter a certain TAN. If an invalid TAN is entered the same TAN is requested.

- We do not agree that this is a security vulnerability. This is how most real world systems work. Requesting a new random TAN is not safer, it just introduces different security risks. For example if an attacker knows one TAN of the user, he just needs to enter false TANs until the TAN he knows is requested by the system. Requesting a random TAN does not make it significantly harder to brute force TANs either (only by a constant factor).

Vulnerability 5: Discussion 2/2

- We think a good solution would be to leave the system as it is, but only let the user try a single TAN three times. This would prevent brute-force attacks.

- After three unsuccsessful tries the account is frozen and the user needs to contact customer service.

- Since there is no customer service in our system, this solution is not implemented. However, we do believe that this is a good solution and used in many real world online banking applications. Our system can very easily be extended to feature this functionality.

Vulnerability 6: Description

- Our Server was not configured to use SSL.

- The use of SSL does not provide any benefits in the context of this project. However, we do agree that in a real world scenario a full SSL configuration using only secure algorithmns (e.g. no use of SHA-1 Certificates ;)) and featuring one or more (intermediate) certificates, signed by well known, trusted parties, is needed.

- Therefore we have implemented bare-bones SSL support with a self-signed certificate.

Vulnerability 6: Problem/Solution

- The Problem:
  - No SSL configuration available.
  - No connection via https.

- The Solution:
  - We have configured our Server for SSL.
  - We created a self-signed certificate using openssl.
  - Connecting via https is now possible, but users will need to accept the untrusted certificate (making the use of SSL a moot point).

Vulnerability 6: Specifics 1/2

- Important adjusted Files/Lines:
  - Httpd.conf
  - X.509 Certificate and PKCS key: mybank.crt/key

- Config (some entries omitted):

```
<VirtualHost localhost:443>
[.....]
SSLEngine on
SSLCertificateFile „/var/apache2/mybank.crt“
SSLCertificateKeyFile „/var/apache2/mybank.key“
</VirtualHost>
```

Vulnerability 6: Specifics 2/2

```
netsec@netsec-VM:~$ openssl x509 -in mybank.crt -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            d3:e9:fb:43:c2:ba:dd:58
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=DE, ST=Bayern, L=M\xC3\x83\xC2\xBCnchen, O=MyBank, OU=MyBank
T, CN=MyBank/emailAddress=admin@mybank-securecoding.com
        Validity
            Not Before: Nov 27 16:30:37 2014 GMT
            Not After : Nov 27 16:30:37 2015 GMT
        Subject: C=DE, ST=Bayern, L=M\xC3\x83\xC2\xBCnchen, O=MyBank, OU=MyBank
IT, CN=MyBank/emailAddress=admin@mybank-securecoding.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:ba:76:f3:37:40:83:5c:f2:6f:4b:ad:5e:30:1d:
                    a7:56:1f:cf:f2:2c:36:20:67:3b:fd:52:bb:c3:de:
                    3b:ec:31:13:85:cb:19:95:5f:5e:37:41:be:50:c8:
                    46:b3:aa:ea:75:a3:07:b3:c8:60:1c:f1:cd:aa:bd:
                    1a:c8:2a:77:9c:8b:55:0c:07:97:92:ab:8f:b1:3f:
                    97:ba:b7:22:cb:e1:e1:7d:66:a6:8b:53:48:f5:84:
                    3d:55:61:51:a4:3e:b4:03:13:8c:f0:bf:a8:3d:0f:
                    81:c5:d2:b9:19:da:6f:fd:74:b8:85:06:25:67:c8:
                    5d:e6:63:70:80:d1:e2:b3:ce:05:a8:63:6f:fa:33:
                    96:c6:e7:91:c1:eb:97:8b:99:5a:c5:fc:34:8a:85:
                    88:d4:dd:5a:68:15:e3:f3:82:0e:af:d3:62:b9:17:
                    56:47:d0:2e:82:f8:ab:d2:00:6a:57:67:47:f2:e2:
                    c6:fe:0d:e0:56:c7:96:65:01:da:27:0a:7e:f8:e1:
                    29:6d:72:81:c1:68:95:36:42:01:ee:d3:be:d3:2e:
                    42:bd:b7:63:c3:c5:12:77:52:a1:91:23:ca:40:46:
                    6a:af:7a:d1:85:03:93:07:bf:62:e1:27:9f:61:45:
                    6e:72:30:2e:79:ba:e8:51:ae:bb:a8:e8:c1:9d:64:
                    7a:a5
                Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Subject Key Identifier:
```