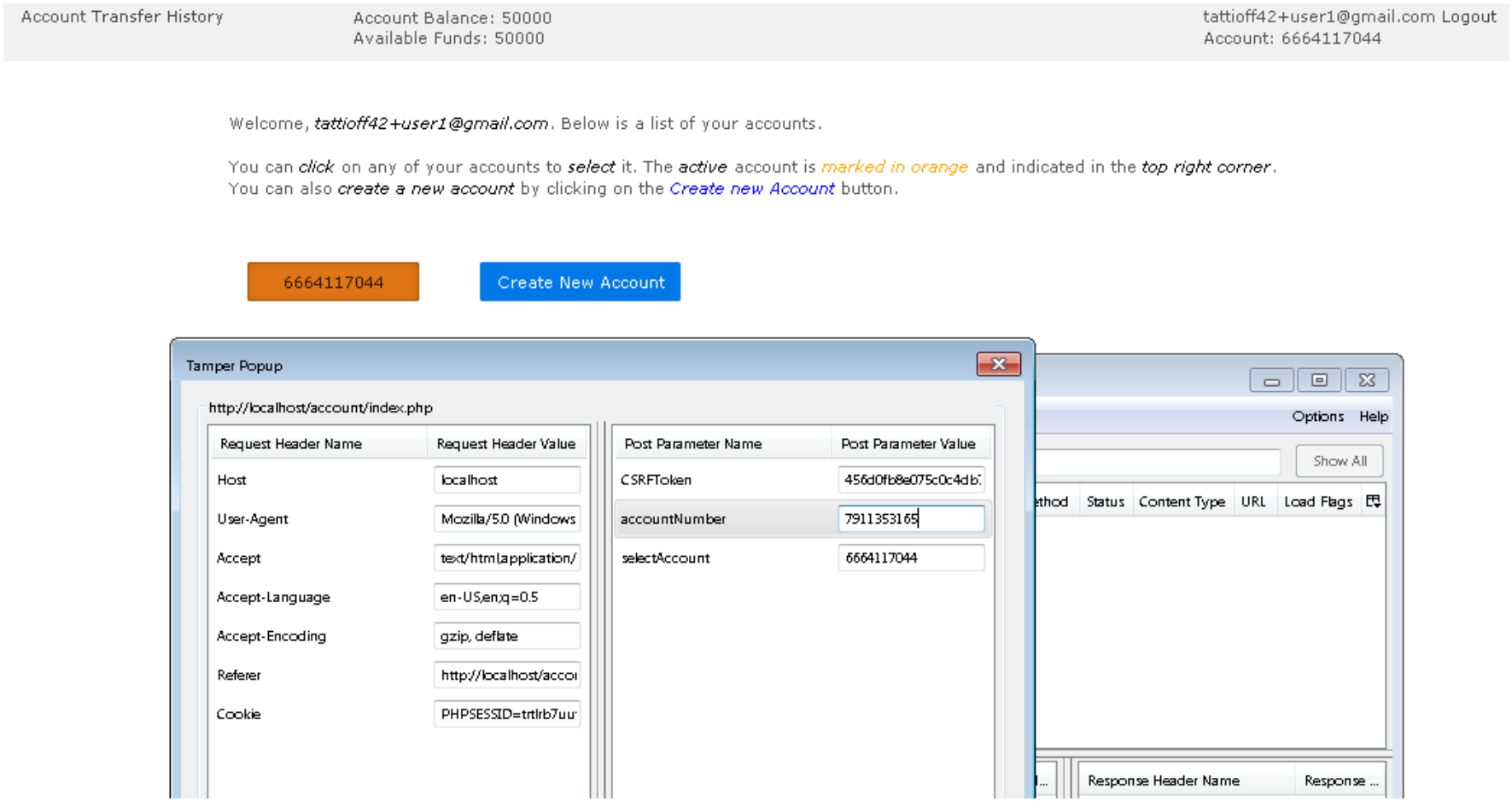
Vulnerability 1: Description

- 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

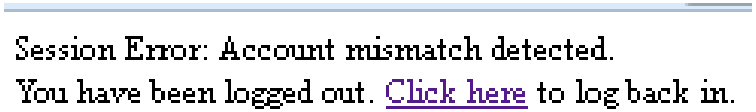
Vulnerability 1: Showcase



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:



  - 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:

```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 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.)

```
9  /* Generate Form Token (valid for this session) */
10 if (!isset($_SESSION['CSRFToken'])) {
11     $_SESSION['CSRFToken'] = generateFormToken();
12 }
```

  - 2.)

```
54         /* Check presence & validity of CSRF Token */
55         if (isset( $_POST['CSRFToken']) && validateFormToken($_POST['CSRFToken'])) {
56             /* Perform Account Creation */
57             $accNumber = randomDigits( 10 );
58             $user->addAccount( $accNumber );
59         } else {
60             $_SESSION['error'] = "CSRF Token Invalid.";
61         }
```

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:
```