StrongPity APT Group Deploys Android Malware for the First Time

trendmicro.com/en_us/research/21/g/strongpity-apt-group-deploys-android-malware-for-the-first-time.html

July 21, 2021

We recently conducted an investigation into a malicious Android malware sample, which we believe can be attributed to the StrongPity APT group, that was posted on the Syrian e-Gov website. To the best of our knowledge, this is the first time that the group has been publicly observed using malicious Android applications as part of its attacks.

We first learned about the sample from a thread shared on the MalwareHunterTeam Twitter. Based on the discussion thread, we learned that the shared sample is a trojanized version of the Syrian e-gov Android application that would steal contact lists and collect files with specific file extensions from its victim's device.

One response from the thread pointed out that the malicious APK was likely distributed using a "watering-hole"-like technique: the attacker allegedly had compromised the official Syrian E-Gov website and replaced the official Android application file with a trojanized version of the original app. Due to the suspicious nature of this activity, we decided to investigate further.

This blog entry will discuss the group’s attack tactics, techniques, and procedures (TTPs) in relation to the Android malware and why these activities can be attributed to this threat actor. Furthermore, we will also dive into the threat actor’s development progress and identify several other malicious Android malware samples produced by StrongPity. Finally, we will briefly discuss related malware variants, including the second version of the Android trojan, which appears to be a work in progress and includes several testing features.

Initial investigation

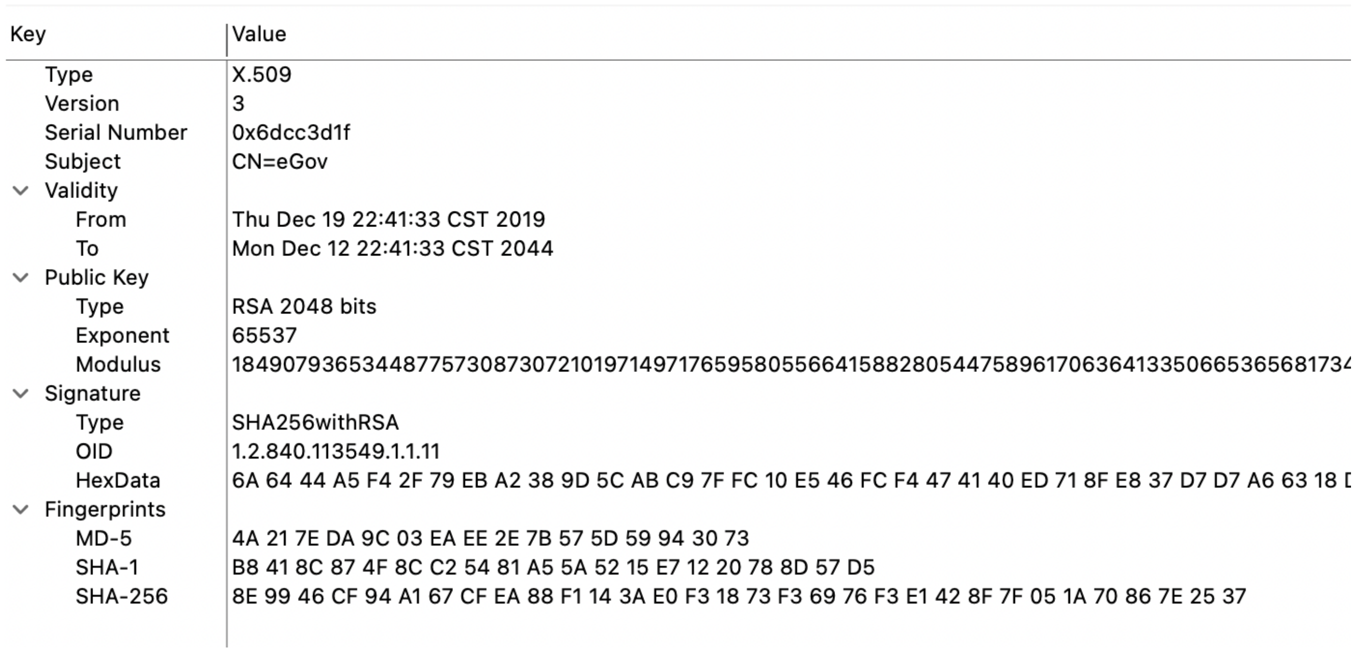
The first thing we did was check the URL where the malicious APK file was hosted (https://egov[.]sy/mobile/egov[.]apk). The version of the application that is downloadable from the site at the time of writing is a clean version of the Syrian e-gov Android application that is different from the malicious application previously discussed on Twitter. This means that, at one point, the malicious version of the application was deleted from the site.

At least six other samples with the same application name ("بوابتي") and matching package names (com.egov.app.*) can be identified on VirusTotal. We verified all of these samples and concluded that all of them are benign. These benign versions of the application were created during the period from February 2020 until March 2021. We believe all of them are official apps from the Syrian E-Gov website.

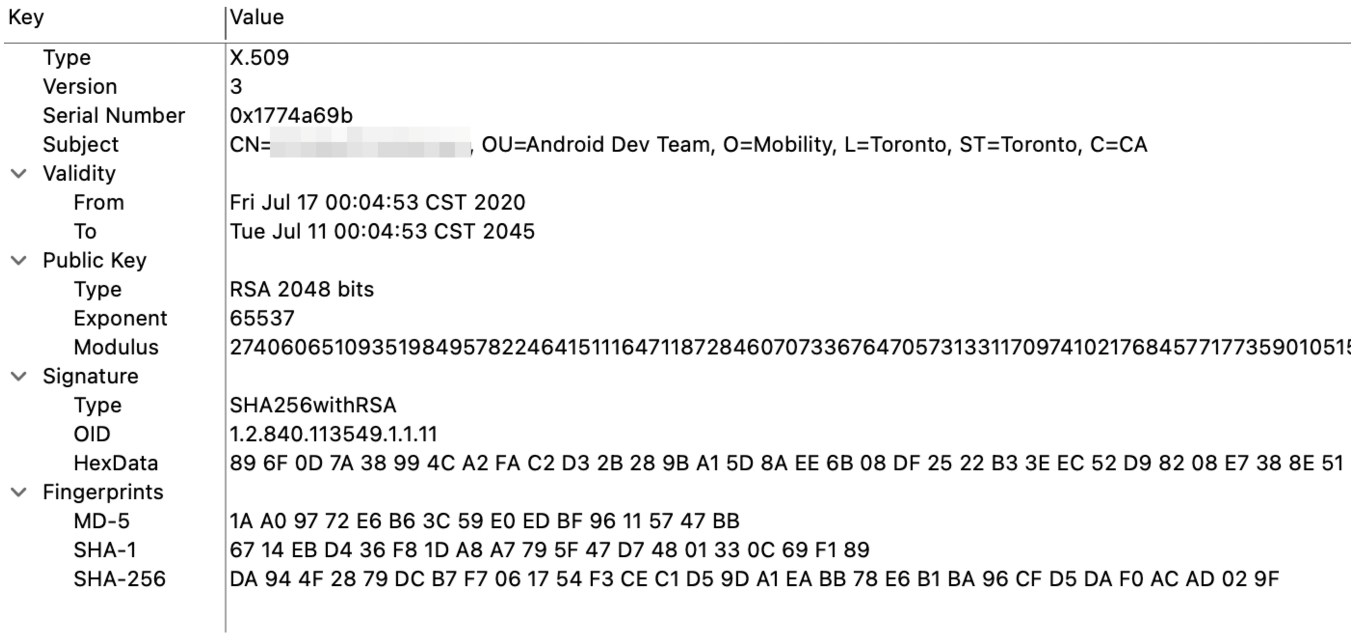
The malicious sample, mentioned in the Twitter thread, is available on VirusTotal and as of the time of writing, has several positive detections. Although some antivirus vendors detect the identified malicious sample as Bahamut, we doubted the accuracy of this attribution to the Bahamut APT group. Further investigation revealed several artifacts that could possibly link the malicious sample to the StrongPity APT group.

Analysis of the malicious sample

The malicious version of the application (fd1aac87399ad22234c503d8adb2ae9f0d950b6edf4456b1515a30100b5656a7) was created on May 2021(The timestamps within the file point to 2021-05-03 as the creation date, while the file was uploaded to VirusTotal on May 24, 2021). This application is signed with a different certificate and was produced by repackaging the original app from the Syrian government. All of the original applications, on the other hand, are signed with another certificate.



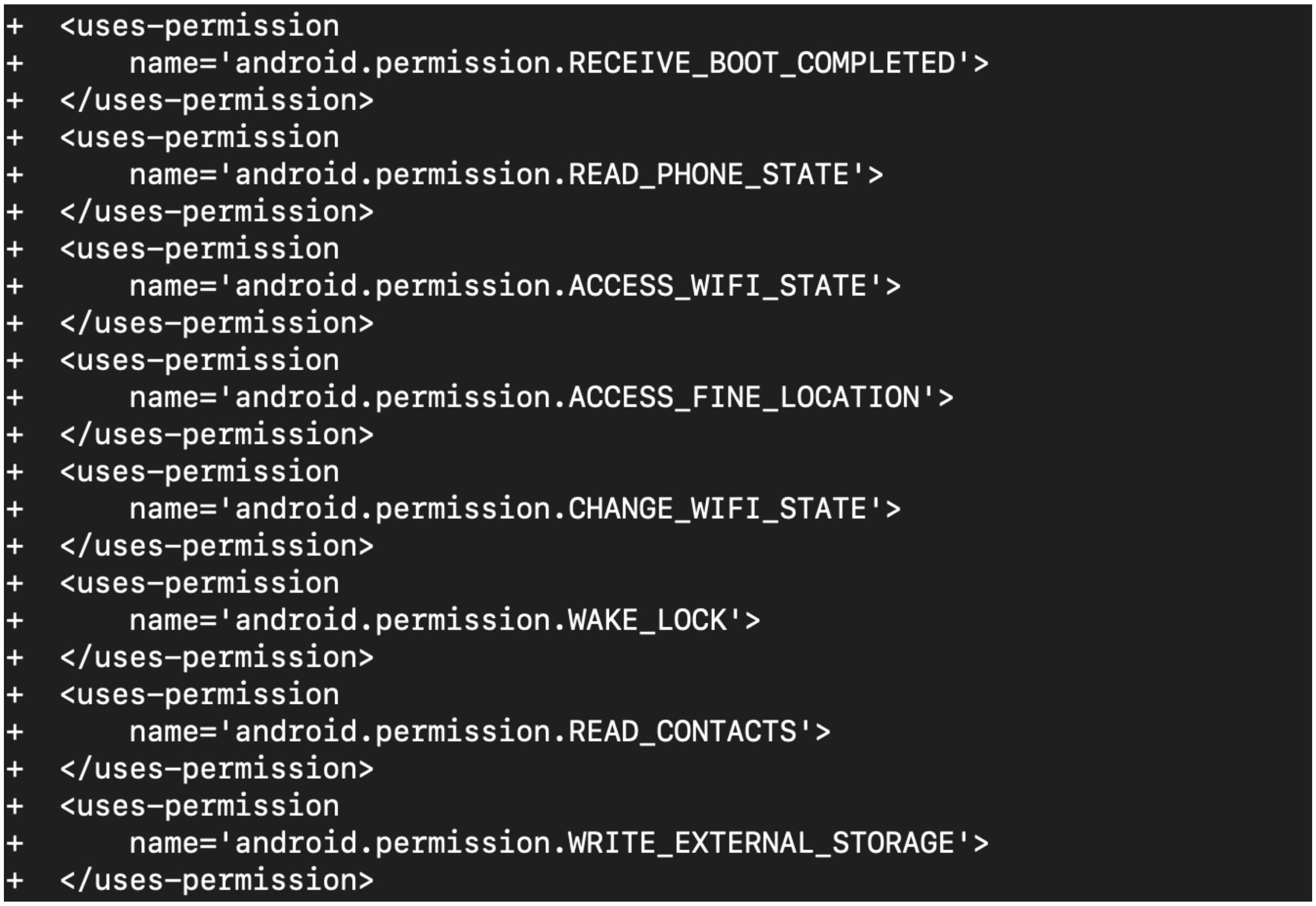
Key	Value
Type	X.509
Version	3
Serial Number	0x6dcc3d1f
Subject	CN=eGov
Validity	
From	Thu Dec 19 22:41:33 CST 2019
To	Mon Dec 12 22:41:33 CST 2044
Public Key	
Type	RSA 2048 bits
Exponent	65537
Modulus	184907936534487757308730721019714971765958055664158828054475896170636413350665365681734
Signature	
Type	SHA256withRSA
OID	1.2.840.113549.1.1.11
HexData	6A 64 44 A5 F4 2F 79 EB A2 38 9D 5C AB C9 7F FC 10 E5 46 FC F4 47 41 40 ED 71 8F E8 37 D7 D7 A6 63 18
Fingerprints	
MD-5	4A 21 7E DA 9C 03 EA EE 2E 7B 57 5D 59 94 30 73
SHA-1	B8 41 8C 87 4F 8C C2 54 81 A5 5A 52 15 E7 12 20 78 8D 57 D5
SHA-256	8E 99 46 CF 94 A1 67 CF EA 88 F1 14 3A E0 F3 18 73 F3 69 76 F3 E1 42 8F 7F 05 1A 70 86 7E 25 37



Key	Value
Type	X.509
Version	3
Serial Number	0x1774a69b
Subject	CN=[redacted], OU=Android Dev Team, O=Mobility, L=Toronto, ST=Toronto, C=CA
Validity	
From	Fri Jul 17 00:04:53 CST 2020
To	Tue Jul 11 00:04:53 CST 2045
Public Key	
Type	RSA 2048 bits
Exponent	65537
Modulus	27406065109351984957822464151116471187284607073367647057313311709741021768457717735901051
Signature	
Type	SHA256withRSA
OID	1.2.840.113549.1.1.11
HexData	89 6F 0D 7A 38 99 4C A2 FA C2 D3 2B 28 9B A1 5D 8A EE 6B 08 DF 25 22 B3 3E EC 52 D9 82 08 E7 38 8E 51
Fingerprints	
MD-5	1A A0 97 72 E6 B6 3C 59 E0 ED BF 96 11 57 47 BB
SHA-1	67 14 EB D4 36 F8 1D A8 A7 79 5F 47 D7 48 01 33 0C 69 F1 89
SHA-256	DA 94 4F 28 79 DC B7 F7 06 17 54 F3 CE C1 D5 9D A1 EA BB 78 E6 B1 BA 96 CF D5 DA F0 AC AD 02 9F

Figure 1. Comparison of the certificates used to sign the original (top) and the malicious applications(bottom)

The malicious application has the AndroidManifest.xml modified to include references to additional classes and request additional permissions on the device (seen in Figure 3).

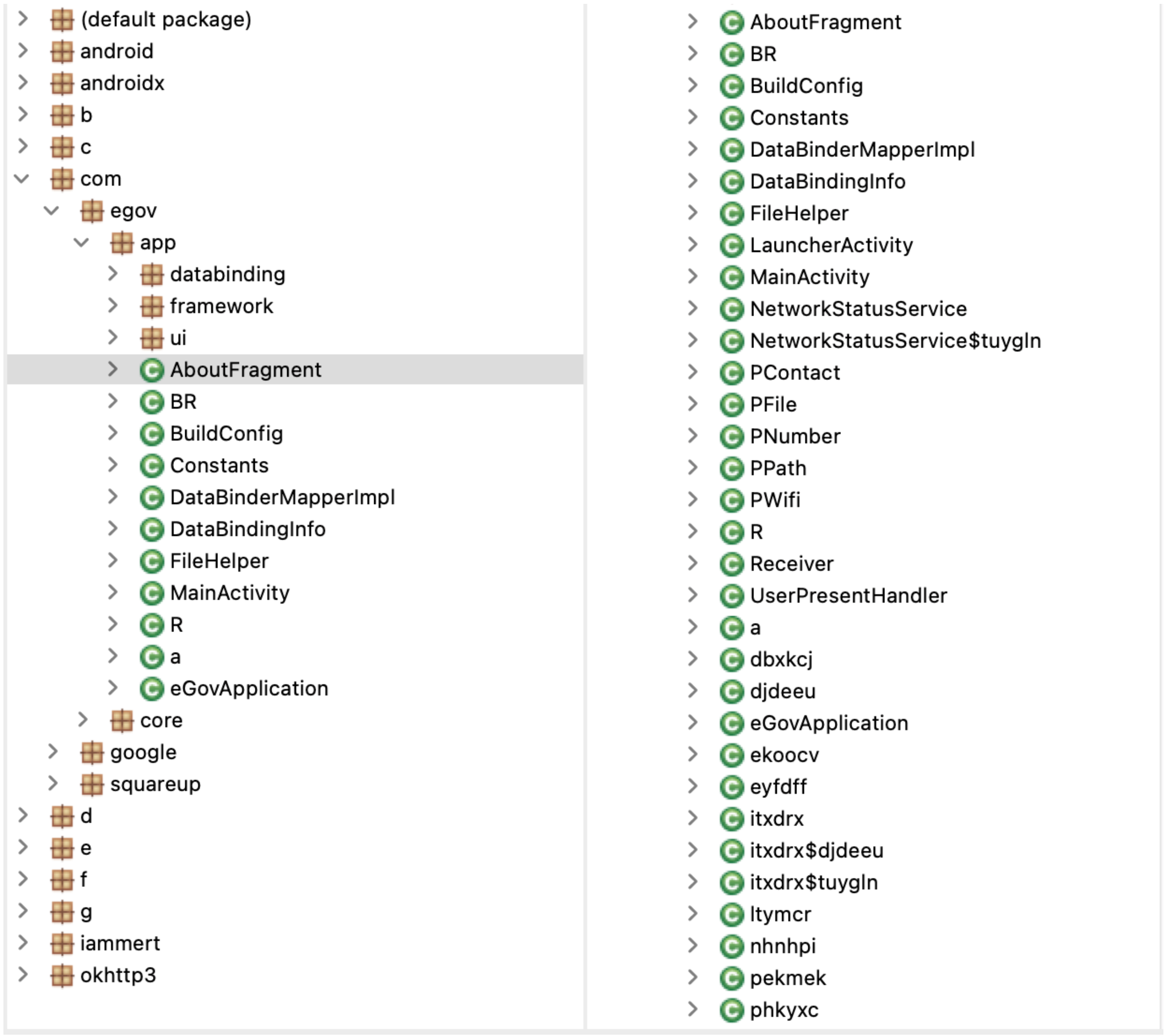


```
+  <uses-permission
+      name='android.permission.RECEIVE_BOOT_COMPLETED'>
+  </uses-permission>
+  <uses-permission
+      name='android.permission.READ_PHONE_STATE'>
+  </uses-permission>
+  <uses-permission
+      name='android.permission.ACCESS_WIFI_STATE'>
+  </uses-permission>
+  <uses-permission
+      name='android.permission.ACCESS_FINE_LOCATION'>
+  </uses-permission>
+  <uses-permission
+      name='android.permission.CHANGE_WIFI_STATE'>
+  </uses-permission>
+  <uses-permission
+      name='android.permission.WAKE_LOCK'>
+  </uses-permission>
+  <uses-permission
+      name='android.permission.READ_CONTACTS'>
+  </uses-permission>
+  <uses-permission
+      name='android.permission.WRITE_EXTERNAL_STORAGE'>
+  </uses-permission>
```

Figure 2. The modified AndroidManifest.xml of the malicious app

Overview of the inserted code

The threat actor added the following classes to this application; some of these classes (com.egov.app.NetworkStatusService, com.egov.app.Receiver) are referenced in the modified AndroidManifest.xml.



```
> (default package)
> android
> androidx
> b
> c
v com
    v egov
        v app
            > databinding
            > framework
            > ui
            > AboutFragment
            > BR
            > BuildConfig
            > Constants
            > DataBinderMapperImpl
            > DataBindingInfo
            > FileHelper
            > MainActivity
            > R
            > a
            > eGovApplication
        > core
    > google
    > squareup
> d
> e
> f
> g
> iammert
> okhttp3
```

```
> AboutFragment
> BR
> BuildConfig
> Constants
> DataBinderMapperImpl
> DataBindingInfo
> FileHelper
> LauncherActivity
> MainActivity
> NetworkStatusService
> NetworkStatusService$tuygln
> PContact
> PFile
> PNumber
> PPath
> PWifi
> R
> Receiver
> UserPresentHandler
> a
> dbxkcj
> djdeeu
> eGovApplication
> ekoocv
> eyfdff
> itxdrx
> itxdrx$djdeeu
> itxdrx$tuygln
> ltymcr
> nhnhpi
> pekmek
> phkyxc
```

Figure 3. The classes added to the malicious app are shown in the right part of the image. The original classes are shown in the left.
Figure 3 shows that many of the other classes have randomly generated class and method names. This naming pattern was likely produced by a software obfuscation tool.

Malicious code initialization

Two major additional components were added to the malicious version of the application: a service and a receiver. The receiver starts the malicious service. The malicious service is declared as an Android Service, which is an application component that can perform long-running tasks in the background.

This malicious service is declared with the class name "com.egov.app.NetworkStatusService", and is started by the Receiver class.

Our analysis of the Receiver configuration and code found multiple methods for starting the malicious NetworkStatusService service.

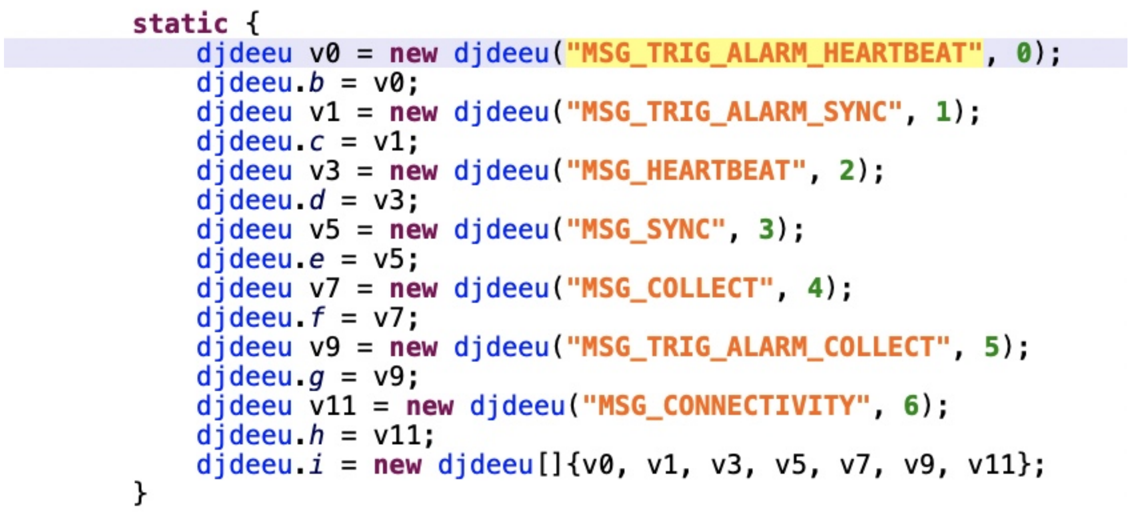
- The service is started when the device connectivity is changed. The djdeeu class registers a broadcast receiver for CONNECTIVITY_CHANGE.
- The service can be started from launcher activity or other registered receivers.
- The service can be started using the "Alarm" mechanism.

Once the malicious NetworkStatusService service is started, it executes its malicious functionality via a set of message handlers, which are responsible for handling specific messages.

Architecture of the backdoor

The sample uses the "Handler" mechanism to dispatch messages that trigger malicious behavior. A custom enum structure is used to define the message types.

It defines seven message types, shown in Figure 4.



```
static {
    djdeeu v0 = new djdeeu("MSG_TRIG_ALARM_HEARTBEAT", 0);
    djdeeu.b = v0;
    djdeeu v1 = new djdeeu("MSG_TRIG_ALARM_SYNC", 1);
    djdeeu.c = v1;
    djdeeu v3 = new djdeeu("MSG_HEARTBEAT", 2);
    djdeeu.d = v3;
    djdeeu v5 = new djdeeu("MSG_SYNC", 3);
    djdeeu.e = v5;
    djdeeu v7 = new djdeeu("MSG_COLLECT", 4);
    djdeeu.f = v7;
    djdeeu v9 = new djdeeu("MSG_TRIG_ALARM_COLLECT", 5);
    djdeeu.g = v9;
    djdeeu v11 = new djdeeu("MSG_CONNECTIVITY", 6);
    djdeeu.h = v11;
    djdeeu.i = new djdeeu[]{v0, v1, v3, v5, v7, v9, v11};
}
```

Figure 4. Code showing the seven defined messages.

Each of the messages trigger a different behavior through the handler. The following is a quick summary of the purpose and behavior of each specific message:

MSG_TRIG_ALARM_HEARTBEAT

When this message is received, a periodic task for heartbeat message is set.

MSG_TRIG_ALARM_SYNC

When this message is received, a periodic task for sync message is set.

MSG_HEARTBEAT

This message triggers the heartbeat function, which sends a request to the command-and-control (C&C) server and receives a response with an encrypted payload.

The encrypted payload is first saved into the directory <DIR>/.android/water.zip, after which the file water.zip is decrypted and the decrypted payload written to <DIR>/.android/e.zip.

Next, the file e.zip is decompressed into <DIR>/.android and the file with the name "config.properties" is accessed.

Finally, this file is read and parsed. These properties are extracted and written as configuration settings to local shared preference, allowing the malware to change its behavior according to the configuration.

MSG_SYNC

Sync is a repeated behavior. It uploads files, which were collected on infected devices, with a periodicity of 3000 seconds.

The handler for MSG_SYNC executes the following functionality:

First, it enumerates all files under <DIR>/.android/.lib2. It then creates a zip file with the name <uniqueId>.zip (note that the unique ID is not a real device ID, the malware just calculates a custom unique ID based on the device ID value), and writes the files into the compressed file.

Finally, it upload the zip file to the C&C server and deletes all files under <DIR>/.android/.lib2, as well as the compressed file <uniqueId>.zip.

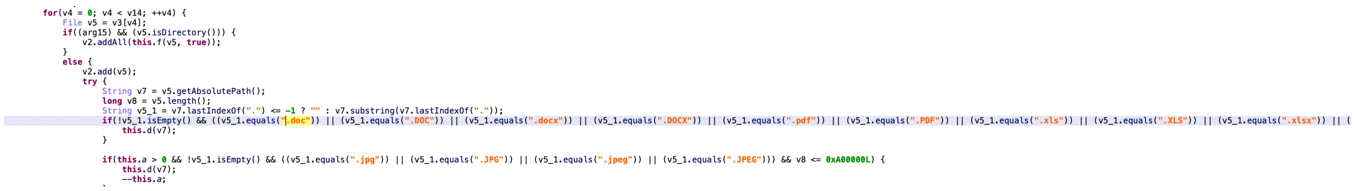
MSG_COLLECT

The handler for this message collects data from the victim's device. First, it collects contact information, followed by information regarding available Wi-Fi networks.

It then searches through the device files and harvests all files that match pre-defined file extensions:

- .asc
- .dgs
- .doc
- .docx
- .edf

- .gpg
- .jpeg
- .jpg
- .key
- .m2r
- .meo
- .pdf
- .pgp
- .pir
- .pkr
- .pub
- .rjv
- .rms
- .sem
- .sit
- .skr
- .sys
- .xls
- .xlsx



```
for(v4 = 0; v4 < v14; ++v4) {
    File v5 = v3[v4];
    if((arg15) && (v5.isDirectory())) {
        v2.addAll(this.f(v5, true));
    }
    else {
        v2.add(v5);
        try {
            String v7 = v5.getAbsolutePath();
            long v8 = v5.length();
            String v5_1 = v7.lastIndexOf(".") <= -1 ? "" : v7.substring(v7.lastIndexOf("."));
            if(!v5_1.isEmpty() && ((v5_1.equals(".doc")) || (v5_1.equals(".DOC")) || (v5_1.equals(".docx")) || (v5_1.equals(".DOCX")) || (v5_1.equals(".pdf")) || (v5_1.equals(".PDF")) || (v5_1.equals(".xls")) || (v5_1.equals(".XLS")) || (v5_1.equals(".xlsx")) || (
                this.d(v7);
            }

            if(this.a > 0 && !v5_1.isEmpty() && ((v5_1.equals(".jpg")) || (v5_1.equals(".JPG")) || (v5_1.equals(".jpeg")) || (v5_1.equals(".JPEG"))) && v8 <= 0xA00000L) {
                this.d(v7);
                --this.a;
```

Figure 5. A snippet of the file harvesting code

MSG_TRIG_ALARM_COLLECT

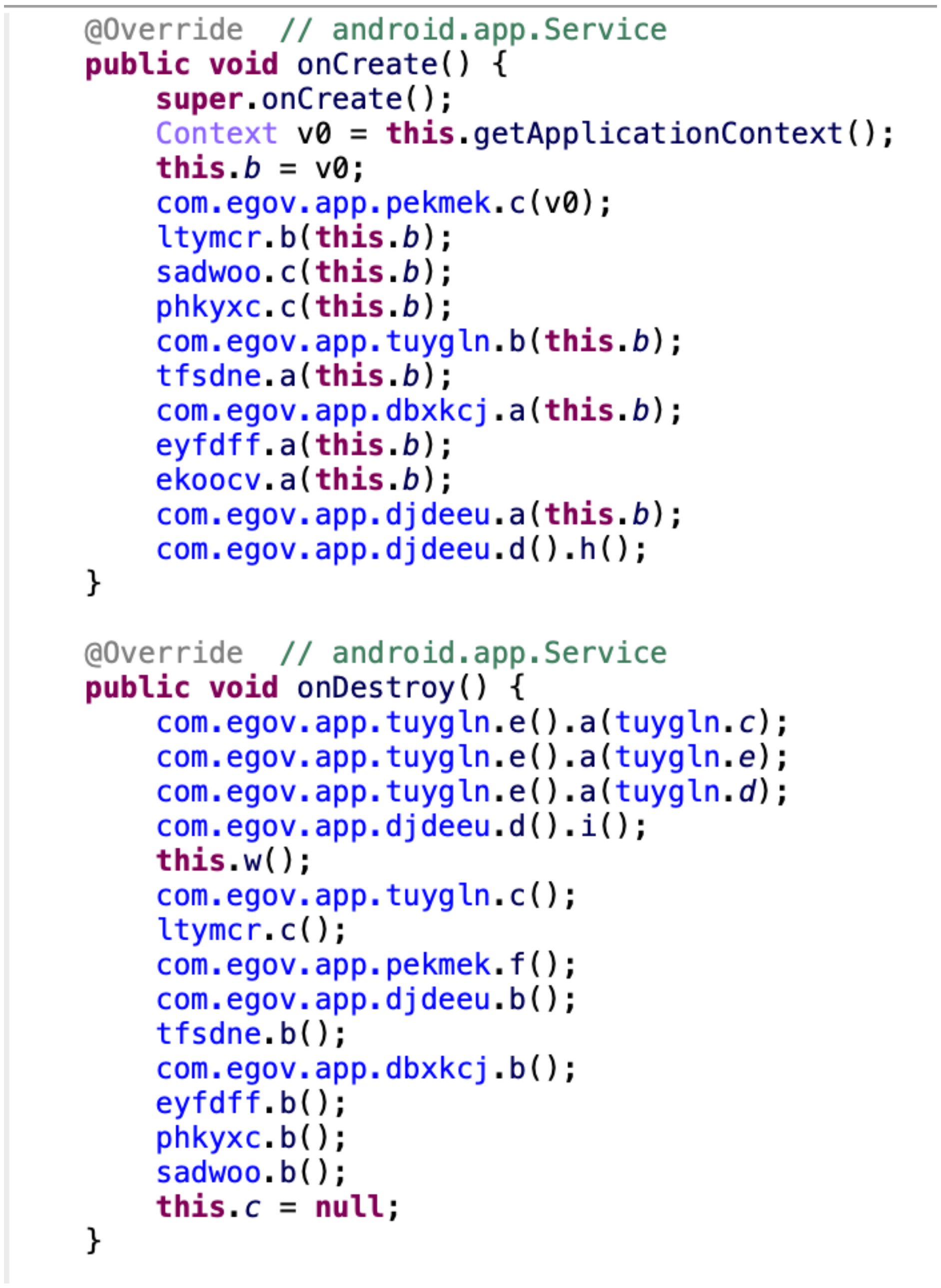
When this message is received, a periodic task for the “collect message” handler is set.

MSG_CONNECTIVITY

This message sends all the mentioned messages one by one.

Modular Functionality of the backdoor

This sample uses highly modular components to create a flexible architecture for component loading and unloading. The functions onCreate and onDestroy show a common approach to loading and unloading a component.



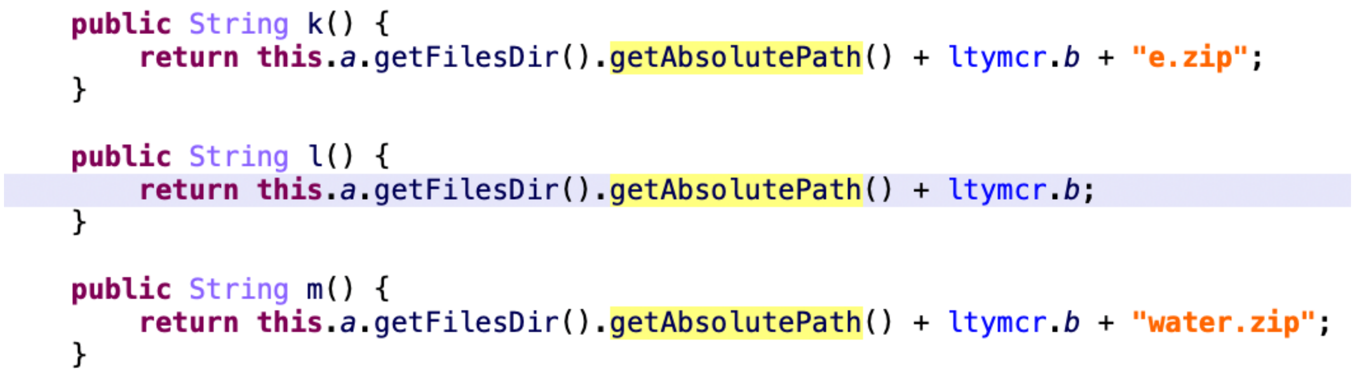
```java
@Override  // android.app.Service
public void onCreate() {
    super.onCreate();
    Context v0 = this.getApplicationContext();
    this.b = v0;
    com.egov.app.pekmek.c(v0);
    ltymcr.b(this.b);
    sadwoo.c(this.b);
    phkyxc.c(this.b);
    com.egov.app.tuygln.b(this.b);
    tfsdne.a(this.b);
    com.egov.app.dbxkcj.a(this.b);
    eyfdff.a(this.b);
    ekoocv.a(this.b);
    com.egov.app.djdeeu.a(this.b);
    com.egov.app.djdeeu.d().h();
}

@Override  // android.app.Service
public void onDestroy() {
    com.egov.app.tuygln.e().a(tuygln.c);
    com.egov.app.tuygln.e().a(tuygln.e);
    com.egov.app.tuygln.e().a(tuygln.d);
    com.egov.app.djdeeu.d().i();
    this.w();
    com.egov.app.tuygln.c();
    ltymcr.c();
    com.egov.app.pekmek.f();
    com.egov.app.djdeeu.b();
    tfsdne.b();
    com.egov.app.dbxkcj.b();
    eyfdff.b();
    phkyxc.b();
    sadwoo.b();
    this.c = null;
}
```

Figure 6. Code snippet showing the onCreate and onDestroy functions

The following are the components were used in the sample:

- pekmek(Crypto Manager): Uses AES to decrypt and encrypt files and strings.
- ltymcr(Helper Class): Contains many utility functions, such as a function to calculate unique id, parse config file, write/read shared preference, and define encryption keys.



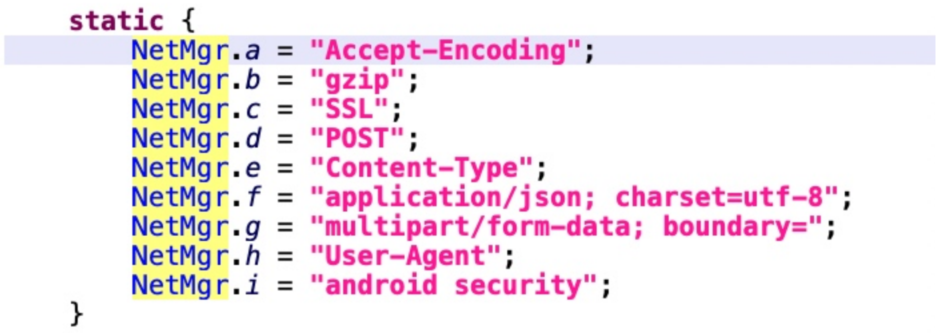
```java
public String k() {
    return this.a.getFilesDir().getAbsolutePath() + ltymcr.b + "e.zip";
}

public String l() {
    return this.a.getFilesDir().getAbsolutePath() + ltymcr.b;
}

public String m() {
    return this.a.getFilesDir().getAbsolutePath() + ltymcr.b + "water.zip";
}
```

Figure 7. Code snippet showing the ltymcr(Helper Class) component

- sadwoo: A component used as PowerWakeLock.
- phkyxc: A component used as WifiWakeLock.
- tfsdne: This is a wrapper used for C&C communication such as heartbeat and sync.
- itxdrx(Net Manager): A component responsible for handling HTTP protocol communication.

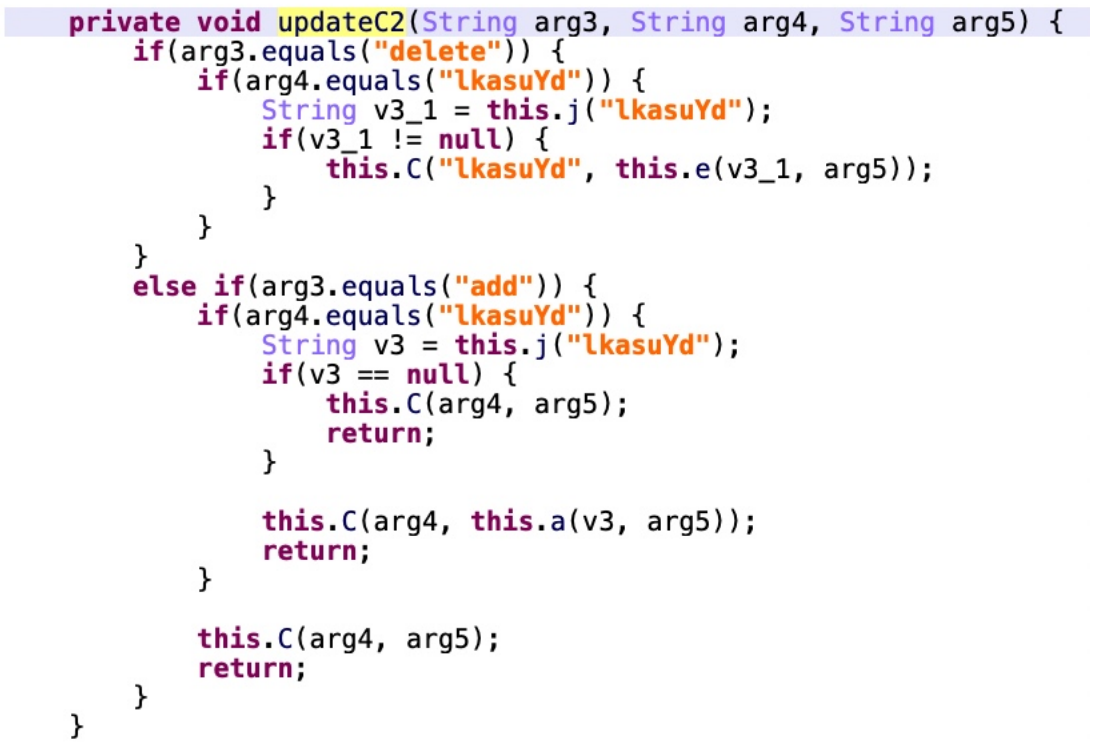


```
    static {
        NetMgr.a = "Accept-Encoding";
        NetMgr.b = "gzip";
        NetMgr.c = "SSL";
        NetMgr.d = "POST";
        NetMgr.e = "Content-Type";
        NetMgr.f = "application/json; charset=utf-8";
        NetMgr.g = "multipart/form-data; boundary=";
        NetMgr.h = "User-Agent";
        NetMgr.i = "android security";
    }
```

Figure 8. Code from the itxdrx component

nhnhpi: The component responsible for managing the C&C server.

This component includes definition of an initial C&C server. The initial value can be overridden. The StrongPity backdoor has the ability to update (including deleting and adding) its C&C server address via configuration updates from the "heartbeat" command.



```
    private void updateC2(String arg3, String arg4, String arg5) {
        if(arg3.equals("delete")) {
            if(arg4.equals("lkasuYd")) {
                String v3_1 = this.j("lkasuYd");
                if(v3_1 != null) {
                    this.C("lkasuYd", this.e(v3_1, arg5));
                }
            }
        }
        else if(arg3.equals("add")) {
            if(arg4.equals("lkasuYd")) {
                String v3 = this.j("lkasuYd");
                if(v3 == null) {
                    this.C(arg4, arg5);
                    return;
                }

                this.C(arg4, this.a(v3, arg5));
                return;
            }

            this.C(arg4, arg5);
            return;
        }
    }
```

Figure 9. The code that handles the addition and deletion of C&C servers

Investigation and attribution

When we learned how the threat actor repackages benign applications into trojanized variants, we decided to search for similar samples on VirusTotal. We searched for other applications that were repackaged in a similar method and included similar malicious components.

Similar malicious samples

We found several other samples that were produced by the same threat actor. We determined these samples to be similar because all of them (except for the last sample) were also repackaged from normal applications and had similar malicious code inserted.

Date of submission	SHA256	Identified C&C servers	Additional Detail

August 2, 2019	374d92f553c28e9dad1aa7f5d334a07dede1e5ad19c3766efde74290d0c49afb	upeg-system-app[.]com	Likely repacked fro
June 8, 2020	be9214a5804632004f7fd5b90fbac3e23f44bb7f0a252b8277dd7e9d8b8a52f3	networktopologymaps[.]com	Likely repacked fro 596257ef017b02b
June 8, 2020	a9378a5469319faffc48f3aa70f5b352d5acb7d361c5177a9aac90d9c58bb628	networktopologymaps[.]com	Likely repacked fro from the Yemeni w
June 13, 2021	596257ef017b02ba6961869d78a2317500a45f00c76682a22bbdbd3391857b5d	upeg-system-app[.]com	Likely repacked fro
January 1, 2021	75dc2829abb951ff970debfba9f66d4d7c6b7c48a823a911dd5874f74ac63d7b	upn-sec3-msd[.]com	This is likely a test sample also conta samples. This sho APK versions of th

Table 1. Similar malicious samples found on VirusTotal

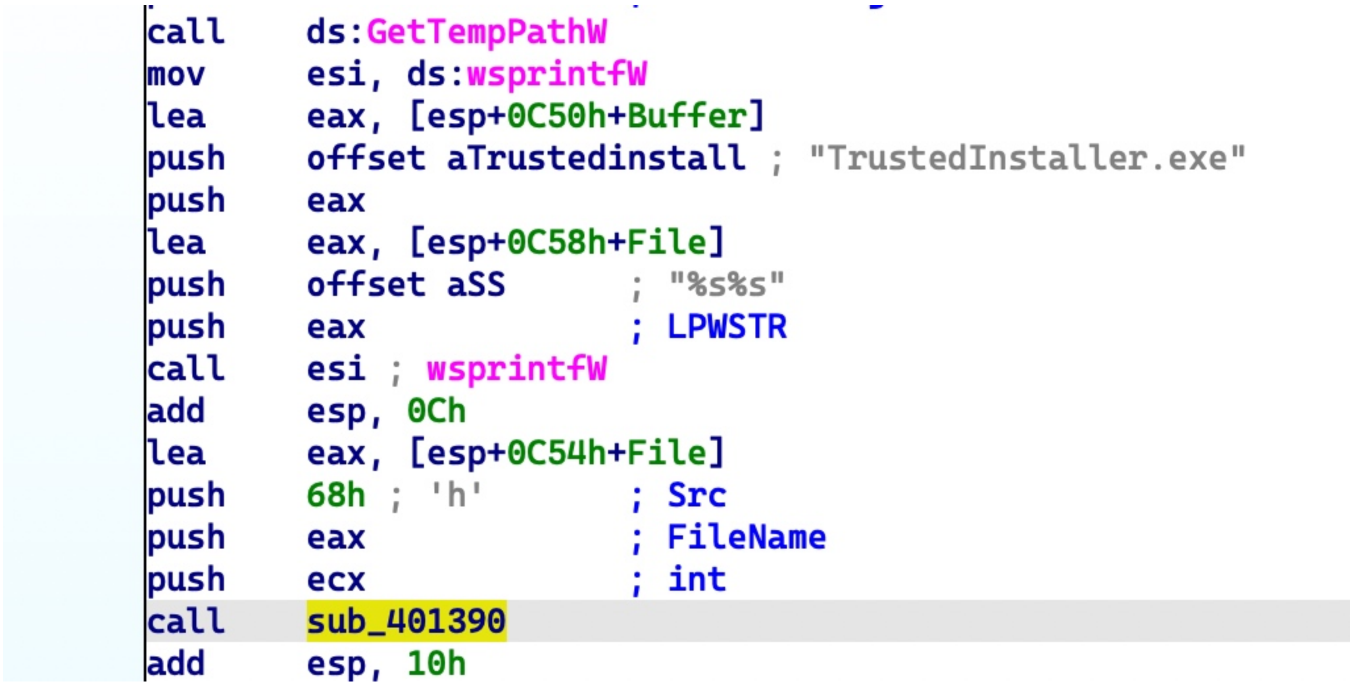
The sample 75dc2829abb951ff970debfba9f66d4d7c6b7c48a823a911dd5874f74ac63d7b serves as the key attribution factor and is the main link to the StrongPity threat actor, because it communicates with a C&C server that was previously identified by several research teams as infrastructure used by the group.

Tools, tactics, and procedures on Windows

There are no known public reports of StrongPity using malicious Android applications in their attacks at the time of writing. In order to strengthen our confidence in the accuracy of our attribution to StrongPity, we decided to further examine some of their samples that were used to target Microsoft Windows platforms and see if we could identify similar tools, tactics, and procedures (TTPs) in their actions.

Just as we have seen with the Android apps, the StrongPity group favors repacking benign installers to produce trojanized versions of these applications. Likewise, the main function of these backdoors is to search, harvest, and exfiltrate files from the victim's computers.

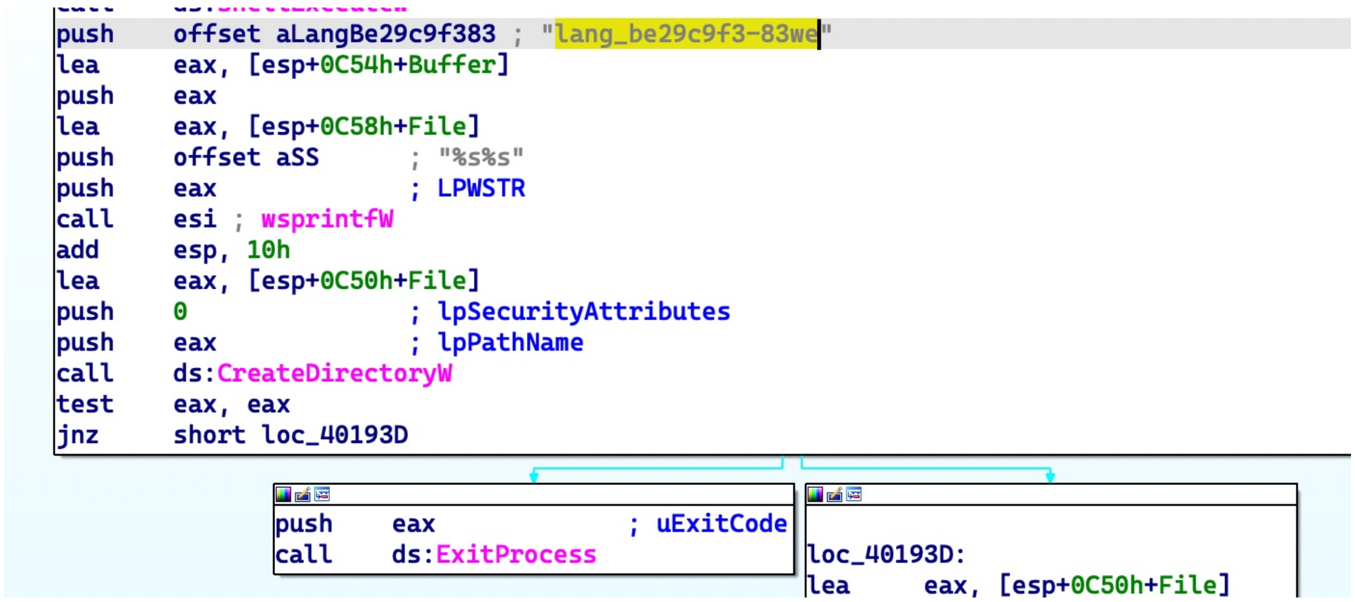
Take, for example, the following sample: 48f67be806b4e823280f03ee5512ffd58deb6f37ecc80842265d4e8d2ca30055. The sample first drops a file called "TrustedInstaller.exe" to <DIR>/AppData/Local/Temp and then executes it. This dropped file is a clean WinRAR installer.



```
call    ds:GetTempPathW
mov     esi, ds:wsprintfW
lea     eax, [esp+0C50h+Buffer]
push    offset aTrustedinstall ; "TrustedInstaller.exe"
push    eax
lea     eax, [esp+0C58h+File]
push    offset aSS      ; "%s%s"
push    eax             ; LPWSTR
call    esi ; wsprintfW
add     esp, 0Ch
lea     eax, [esp+0C54h+File]
push    68h ; 'h'       ; Src
push    eax             ; FileName
push    ecx             ; int
call    sub_401390
add     esp, 10h
```

Figure 10. Code used in the file "TrustedInstaller.exe"

It then creates <DIR>/AppData/Local/Temp/lang_be29c9f3-83we to drop malicious files and execute them.


```
push    offset aLangBe29c9f383 ; "lang_be29c9f3-83we"
lea     eax, [esp+0C54h+Buffer]
push    eax
lea     eax, [esp+0C58h+File]
push    offset aSS      ; "%s%s"
push    eax             ; LPWSTR
call    esi ; wsprintfW
add     esp, 10h
lea     eax, [esp+0C50h+File]
push    0               ; lpSecurityAttributes
push    eax             ; lpPathName
call    ds:CreateDirectoryW
test    eax, eax
jnz     short loc_40193D

push    eax             ; uExitCode
call    ds:ExitProcess

loc_40193D:
lea     eax, [esp+0C50h+File]
```
Figure 11. Code showing the creation of the directory

If we examine another StrongPity sample (12818a96211b7c47863b109be63e951075cf6a41652464a584dd2f26010f7535), the logic is similar — it drops a normal installer into the Temp directory and creates a directory for dropped malicious files.

Here are three notable similarities between the Windows sample and the Android sample:

1. They all disguised as normal apps by utilizing the original clean applications — the Android sample repacks the original one into a trojanized version, while the Windows sample uses a trojanized installer packed with the original program.

2. Both collect and exfiltrate files from the infected device.

3. Both are highly modular. The Windows sample has a standalone Exfiltration and File Search module, a feature that could also be seen in the latest test Android sample.

Possible connections to StrongPity

We found several clues that link the malicious Android samples with the StrongPity threat actor.

The sample 74582c3d920332117541a9bbc6b8995fbe7e1aff communicates with the URL https://www.upn-sec3-msd[.]com/ProxyServer/service/. The domain name "upn-sec3-msd[.]com" was mentioned in another StrongPity report.

The domain naming pattern and domain acquisition techniques are quite similar. For example, the domain names used by StrongPity in 2020 have a domain naming pattern similar to the domains used by the identified Android samples.

One of the domain names, networktopologymaps[.]com, was likely bought when registration at Gandi expired. The domain was acquired via the Porkbun network registrar.

This is similar to the domain hostoperationsystems[.]com, which was previously mentioned in the Talos report. This domain was also acquired via Porkbun and features a comparable domain naming pattern.

Another notable point of correlation to StrongPity is the list of file extensions, which we have seen in Android samples. A similar list of the file extensions for the files is presented in variants of the trojan for Windows systems. For example, one of the samples that we had examined earlier, gathers files with the following extensions:

- .7z
- .asc
- .dgs
- .doc
- .docx
- .gpg
- .pdf
- .pgp
- .ppt
- .pptx
- .rar
- .rjv
- .rms

- .rtf
- .sft
- .tc
- .txt
- .xls
- .xlsx

As we previously mentioned, there are no public reports of the StrongPity threat actor using malicious Android applications in the attack. However, we examined the trojan code-embedding techniques as well as the trojan functionality of the malicious code written by the same threat actor for Windows platforms, and we have identified some similar patterns. This leads us to believe that these could belong to the same threat actor.

StrongPity actively develops new malicious android apps

We believe that the StrongPity Threat actor is actively developing backdoors for Android. Based on the test sample that we have identified, we can see that the threat actor attempts several techniques to lure potential victims: repackaged applications, compromised websites, and fake variants of popular applications.

Based on the additional functionalities that we identified in the fake Samsung security service application (75dc2829abb951ff970debfba9f66d4d7c6b7c48a823a911dd5874f74ac63d7b), we think that among the APK files that we had identified, the repackaged applications are bundled with the first version of the Android trojan, while the fake application could be a work in progress for the next version of the tool.

In the second version, we observed the threat actor developed and included some additional components and as well as added support for more message types.

The following table shows the types that the threat actor has defined.

Message type	Details
MSG_ADD_MODULE	Add a new module
MSG_GET_MODULE	Get the module instance
MSG_DEL_MODULE	Delete module file under <DIR>/.android/.li/<module name>
MSG_DEL_APK	Delete the APK file under the download directory
MSG_START_MODULES	

Table 2. Message types defined by the threat actor

In this version, MSG_COLLECT is no longer present — we think they replaced it with MSG_START_MODULES, a message used to read all module names from the shared preference, and start/initialize them one by one.

We were not able to get access to these modules, but based on some of the code functionality that we observed, we believe that these modules are designed to collect data from the victim's devices and write the collected data into a local SQLite db data file. However, we were not able to find any of these modules in the wild.

There are also several other key differences between version 1 and version 2 of the trojan:

- The message Handler for heartbeat message in version 2 is now split into two messages: heartbeat and taken_config. Either of these messages can receive a response from the C&C server and decrypt the response to update the local configuration, similarly to the version 1.
- Version 2 uses different AES encryption keys: key("aaaanothingimpossiblebbb"), and AES IV("aaaanothingimpos")
- ScreenReceiver class is added to the second version of the trojan. The purpose of this Receiver is to start the malicious service via Screen_On and Screen_Off events.
- Version 2 has an ability to execute "su" command, if the device is rooted. The main usage of the root privilege here is that it could grant permissions silently. Such permissions include accessibility, notification and other. However, we did not find any evidence that the sample would attempt to root the device.
- Two components were added in version 2 for accessibility and notification.
- Version 2 uses SQLite to store collected data. Furthermore, it no longer uses ZIP.
- In Version 2, the extra modules used in "MSG_START_MODULES" are downloaded from the C&C server via either the heartbeat or taken_config message. It's possible that these modules are decompressed as part of the response into <DIR>/.android/.li and consequentially executed.

Conclusion

This investigation has provided evidence to attribute the Android malware sample, which was posted on the Syrian e-Gov website, to the StrongPity threat group. We were also able to identify additional Android trojan files and correlate these malicious Android applications with existing public reports based on their similarities to the threat actor's TTPs and network infrastructure they used.

Although there are no previously known malicious Android applications attributed to the StrongPity group, we strongly believe that the threat actor is in the process of actively developing new malicious components that can be used to target Android platforms.

We believe that the threat actor is exploring multiple ways of delivering the applications to potential victims, such as using fake apps and using compromised websites as watering holes to trick users into installing malicious applications. Typically, these websites would require its users to download the applications directly onto their devices. In order to do so, these users would be required to enable installation of the applications from "unknown sources" on their devices. This bypasses the "trust-chain" of the Android ecosystem and makes it easier for an attacker to deliver additional malicious components.

Indicators of Compromise (IOCs)

Files

SHA256	Description	Detection
fd1aac87399ad22234c503d8adb2ae9f0d950b6edf4456b1515a30100b5656a7	The trojanized version of the Syria eGov Application	AndroidOS_StrongPity.HRX
374d92f553c28e9dad1aa7f5d334a07dede1e5ad19c3766efde74290d0c49afb	Sample repackaged from Kingoroot	AndroidOS_StrongPity.HRX
a9378a5469319faffc48f3aa70f5b352d5acb7d361c5177a9aac90d9c58bb628	Sample repackaged from net.cybertik.wifi	AndroidOS_StrongPity.HRX
be9214a5804632004f7fd5b90fbac3e23f44bb7f0a252b8277dd7e9d8b8a52f3	Repackaged from Snaptube	AndroidOS_StrongPity.HRX
596257ef017b02ba6961869d78a2317500a45f00c76682a22bbdbd3391857b5d	Repackaged from Snaptube	AndroidOS_StrongPity.HRX
75dc2829abb951ff970debfba9f66d4d7c6b7c48a823a911dd5874f74ac63d7b	Fake Samsung Security Service sample	AndroidOS_StrongPity.HRX

Network C&C Infrastructure

SHA256	Domain	Detection
fd1aac87399ad22234c503d8adb2ae9f0d950b6edf4456b1515a30100b5656a7	Internetwideband[.]com	AndroidOS_StrongPity.HRX
374d92f553c28e9dad1aa7f5d334a07dede1e5ad19c3766efde74290d0c49afb	upeg-system-app[.]com	AndroidOS_StrongPity.HRX
a9378a5469319faffc48f3aa70f5b352d5acb7d361c5177a9aac90d9c58bb628	networktopologymaps[.]com	AndroidOS_StrongPity.HRX
be9214a5804632004f7fd5b90fbac3e23f44bb7f0a252b8277dd7e9d8b8a52f3	networktopologymaps[.]com	AndroidOS_StrongPity.HRX
596257ef017b02ba6961869d78a2317500a45f00c76682a22bbdbd3391857b5d	upeg-system-app[.]com	AndroidOS_StrongPity.HRX
75dc2829abb951ff970debfba9f66d4d7c6b7c48a823a911dd5874f74ac63d7b	upn-sec3-msd[.]com	AndroidOS_StrongPity.HRX