“We Know What You Did This Summer” Android Banking Trojan Exposing its Sins in The Cloud

By Carlos Castillo and Siegfried Rasthofer

ABSTRACT

Backend-As-A-Service (BaaS) solutions are a very convenient way for developers to connect their apps easily with a cloud storage. There are different BaaS solutions on the market, offered by various vendors such as Amazon, Google or Facebook. All of them provide simple APIs for common tasks such as managing database records or files. Adding a few library classes and writing three or four lines of code is sufficient to integrate cloud storage into the app. While usually such solutions are created for well-intentioned developers, very recently we have spotted two Android malware families that make use of BaaS solutions as well, Facebook's in this case. Using Facebook's BaaS solution the malware stores stolen data, delivers commands executed remotely on the infected device and performs SMS banking fraud. Malware authors apparently are unaware, however, of how to set up a BaaS solution securely, which gave us the possibility to easily obtain access to all data they store. This gave interesting insights into their C&C (Command and Control) communication protocol and all sensitive data they stole, including requesting the current balance of credit cards associated with the device, and attempting to perform payments and fraudulent transfer of funds via SMS messages during June and July 2015. To extract the necessary data from malicious applications automatically, we developed an automatic exploit generator that extracts credentials from the app, even if they are obfuscated, and provides access to the respective BaaS backend.

INTRODUCTION

Nowadays, most mobile app are connected to the Internet. The connectivity is needed, among other reasons, to increase the availability of a broad variety of data across different devices and platforms. Cloud storage is becoming a “must” for mobile application projects; preventing data loss, due to hardware failure, and enabling users access to their data across a variety of devices are some of the reasons for this migration to “the cloud”. However, storing and managing all the data remotely can be costly. Along with the planned app development, additional engineering, testing, and IT resources (with specific knowledge of databases, server-side languages, app server operational support) are required to develop and maintain the backend.

As a response to the increasing need of a solution to store and manage data for mobile apps, Internet companies like Amazon, Google and Facebook started to offer fully-maintained, ready-to-use and easy-to-implement Backend solutions, formally known as Backend-as-a-service (BaaS), to provide secure data

storage and management for mobile and web applications. Besides data storage, they also offer various additional features such as push notification mechanisms, user management or integrating social media into the application. The principle is always the same, the BaaS provider offers pre-defined backend solutions and BaaS SDKs to the developer. These SDKs, available for different platforms such as Android, iOS, JavaScript, etc., are added into the Client's code (e.g. Android application) and offer the developer easy-to-use APIs (e.g. two lines of code for writing data into the database)

However, recently it was found that, even when the BaaS providers offer security features to protect the data stored in their infrastructure, the default implementation and configuration of those services is insecure and could allow unauthorized access of the data "securely" stored in the cloud.

Sensitive information exposed by apps insecurely using BaaS

In March 2015, Siegfried Rasthofer from Technische Universität Darmstadt and Eric Bodden from Fraunhofer SIT, with assistance from Intel Security, investigated three major BaaS providers (Facebook Parse, CloudMine and Amazon AWS) and found 56 million sets of unprotected data records[1] by scanning about 2 million apps collected from various sources, such as Google Play. The researchers were able to access cloud databases from different legitimate apps and found sensitive information like full names, e-mail addresses, passwords, photos, money transactions and even health records that could be used to perform identity thief, to send e-mail spam, to distribute malware etc.

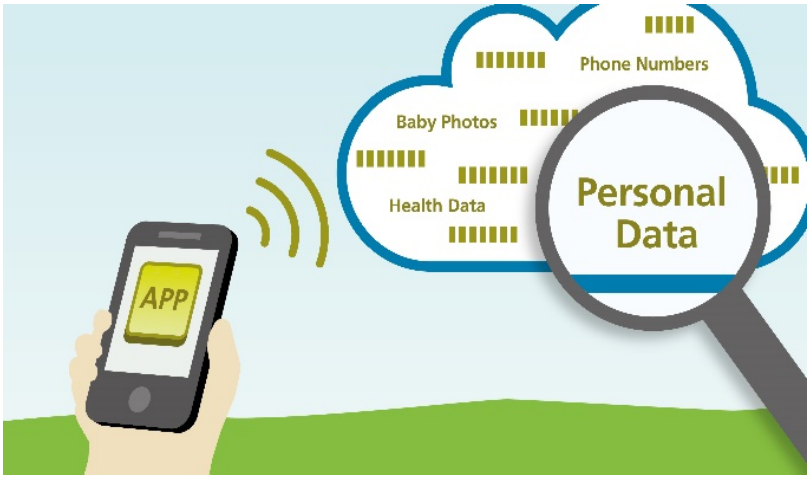

Figure 1. Sensitive information in BaaS[2]

The access was possible since app developers did not securely protect the user's data in the backend. In order to connect to the BaaS database ("cloud"), the developer has to create an ID/key pair in the BaaS backend, which has to be added into the app's code. Amazon's documentation[3] states that the ID /"secret-key" pair are used for identifying the application. Usually, identifying a user only requires a

[1] https://www.sit.fraunhofer.de/en/news-events/latest/press-releases/details/news-article/technische-universitaet-darmstadt-und-fraunhofer-sit-datenleck-in-apps-bedroht-millionen-von-nutzer/
[2] https://www.sit.fraunhofer.de/fileadmin/bilder/presse/appdatathreat_pressebild.jpg
[3] http://docs.aws.amazon.com/

single ID, where an ID/password pair is used for authentication. However, BaaS providers use an ID/"secret-key" pair for identifying the application, which is odd and can be the reason why various developers did not understand that the identification-pair is not an authentication mechanism. Instead, the developer should also authenticate the user if the application stores user-related information in the cloud and protect the access to personal data with proper authorization mechanisms like the Access-Control-Lists (ACLs).

Unfortunately, this was not the case in the majority of the applications that used BaaS mechanisms. Instead, the researchers even found cases where the developers tried to obfuscate the ID/"secret"-key pair with various obfuscation techniques. This shows once more that the developers did not understand that they have to add additional security mechanisms on top of the default application-identification mechanisms to protect the user's data.

To determine the dimension of the data leakage, the researchers developed a framework that fully and automatically scans applications for potential BaaS data leakage (more details about the framework can be read in their publication[4]). In the initial research, only legitimate apps were included but they are not the only ones that manage and store sensitive and private information. Android malware also steals and leaks private and sensitive information (phone contacts, incoming/outgoing SMS messages, geo-location, banking information) that could be stored insecurely in BaaS.

Android/OpFake and Facebook Parse

With the help of the researcher's framework, we scanned 294,817 malware applications in July 2015 and found 16 apps with Facebook's Parse BaaS instances, 9 of them with confirmed access to cloud database tables (NewTasks, SmsReceiver and TaskManager) which implies that BaaS was also used as a Command and Control server. In total we found 5 Facebook Parse accounts exposed which were used by two different, but related, Android Banking Trojan families: Android/OpFake and Android/Marry.

In order to understand how these threats make use of the BaaS services, and what type of information is stored in the cloud, we decompiled and statically analyzed one variant of the malware family Android/OpFake. The app, most-likely distributed via Smishing attacks, pretends to be an "Installer" (Установка) for the instant-messaging Russian app Chat for Friends[5] (Чат для друзей. ДругВокруг):

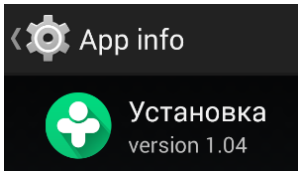

Figure 2. Android/OpFake pretending to be the app Chat for Friends

When the app is executed, a fake message is shown to the user saying that the app will be downloaded and installed on the device:

[4] https://www.blackhat.com/docs/eu-15/materials/eu-15-Rasthofer-In-Security-Of-Backend-As-A-Service-wp.pdf
[5] https://play.google.com/store/apps/details?id=drug.vokrug

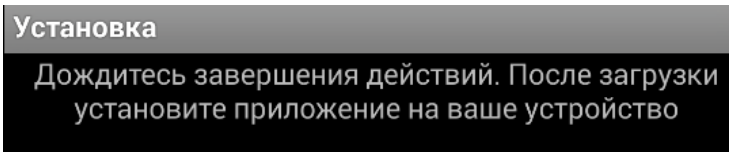

Figure 3. Android/OpFake fake download message

However, instead of downloading the Chat for Friends app, the malware hides the icon on the home screen and starts a service in the background that subscribe the device to Parse Push notifications, leaks device information to Parse and traditional Command and Control server and schedules a system alarm. The information that is stored in the Parse (Facebook's BaaS) database is mainly intercepted SMS messages and a Command and Control functionality. The tables are called "NewTasks", "SMSReceiver" and "TaskManager" (more details in the next section). Figure 4 shows a workflow of the main components of the Android/OpFake family that makes use of the BaaS solution.

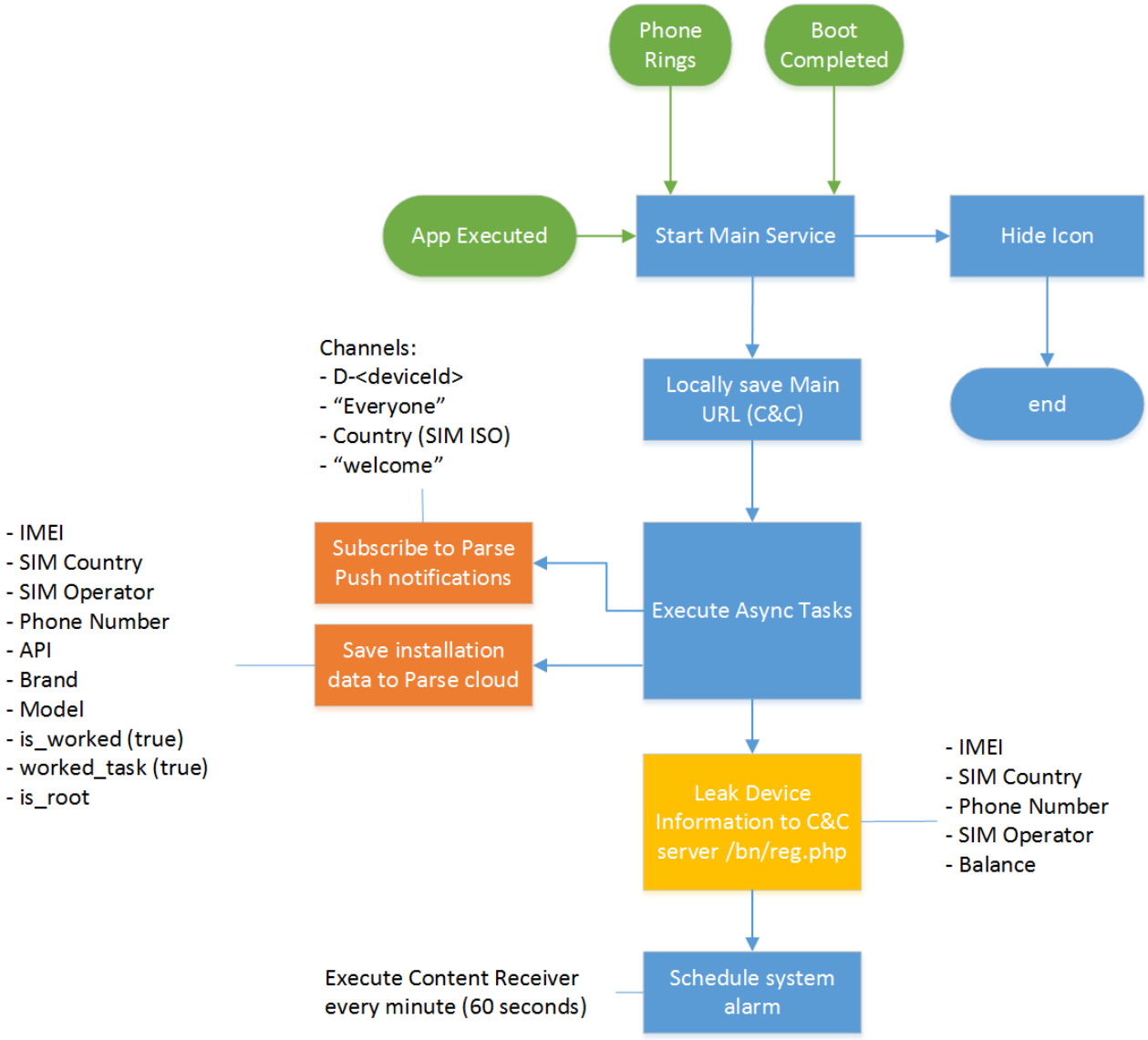

Figure 4. Android/OpFake initial behavior

Figures 4 to 7 have the following type of components:

- Green shape: Events in the system that trigger the execution of specific code like a system alarm, the app opened by the user or an incoming call.
- Blue shape: Actions performed by the malware itself like hiding the icon on the home screen or executing a specific task in the infected device.
- Yellow shape: Malware functionality related to communication between the malware and the remote Command and Control server.
- Orange shape: Interaction between the malware and Facebook Parse BaaS.

Figure 4 shows that a service is started in the background when specific events occur. Among other actions, the service subscribes the device to several channels to receive push notifications with new tasks to be executed. Interestingly, the malware makes use of Parse push notification mechanism and did not implement its own mechanism. The tasks can be sent to:

- A specific device using the device identifier (IMEI or android_id).
- Devices from a specific country.
- All devices (Everyone channel).
- Devices subscribed to the channel “welcome”.

Figure 5 shows that the system alarm in Figure 4 will check if there are new commands to be executed both in the traditional (remote) C&C server (yellow shape) and in the NewTasks table in Facebook Parse (orange shape).

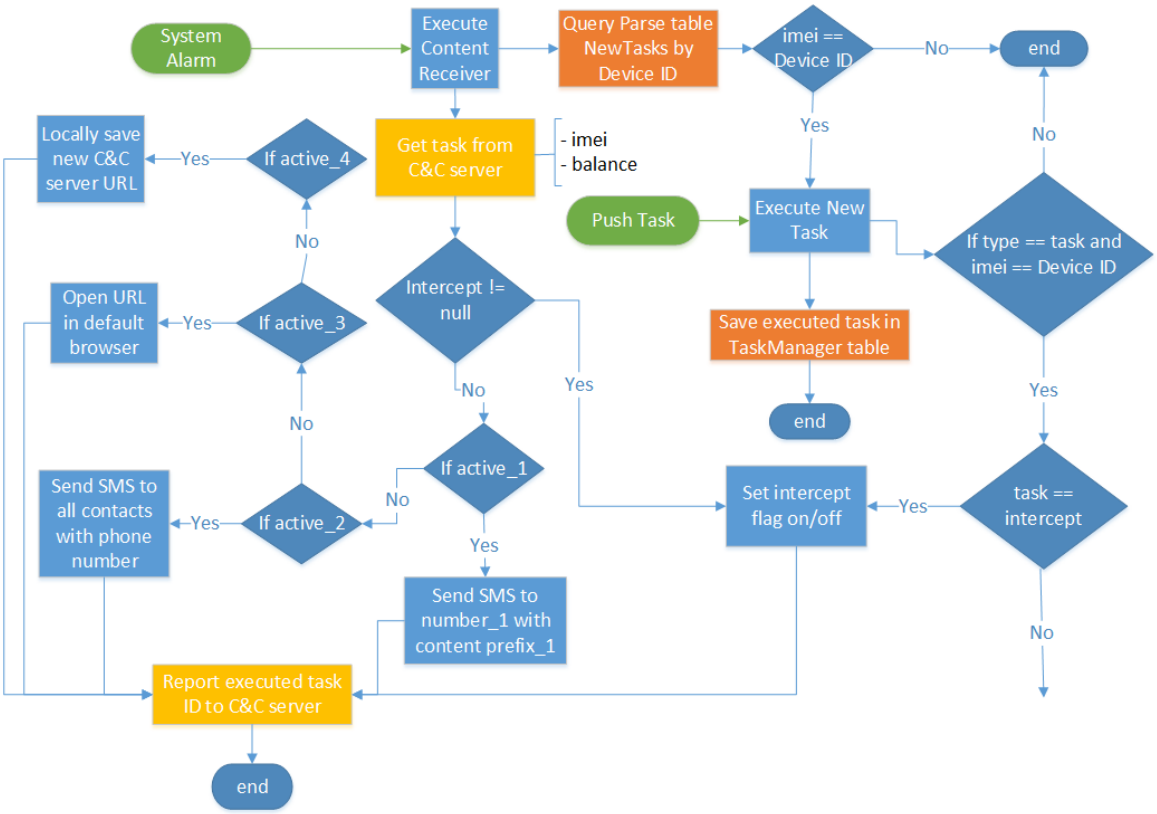

Figure 5. Android/OpFake behavior when the system alarm is fired

The remote (traditional) C&C server commands are the following:

- Send an SMS message to a specific number and with a specific content provided by the remote server.
- Update the C&C server URL.
- Open a URL in the default browser.
- Send a SMS message to all the contacts in the device with a phone number.

Once the task delivered by the remote server is successfully executed by the infected device, the task ID is reported back to the C&C server.

In addition to the traditional C&C communication, the malware also used the BaaS data storage (database) for retrieving new commands. If there are new commands in the NewTasks table (top-right of Figure 5) the malware first checks the device identifier. Once confirmed, it checks which command shall be executed next. Figure 6 shows all the commands that can be executed if they are found in the NewTasks table.

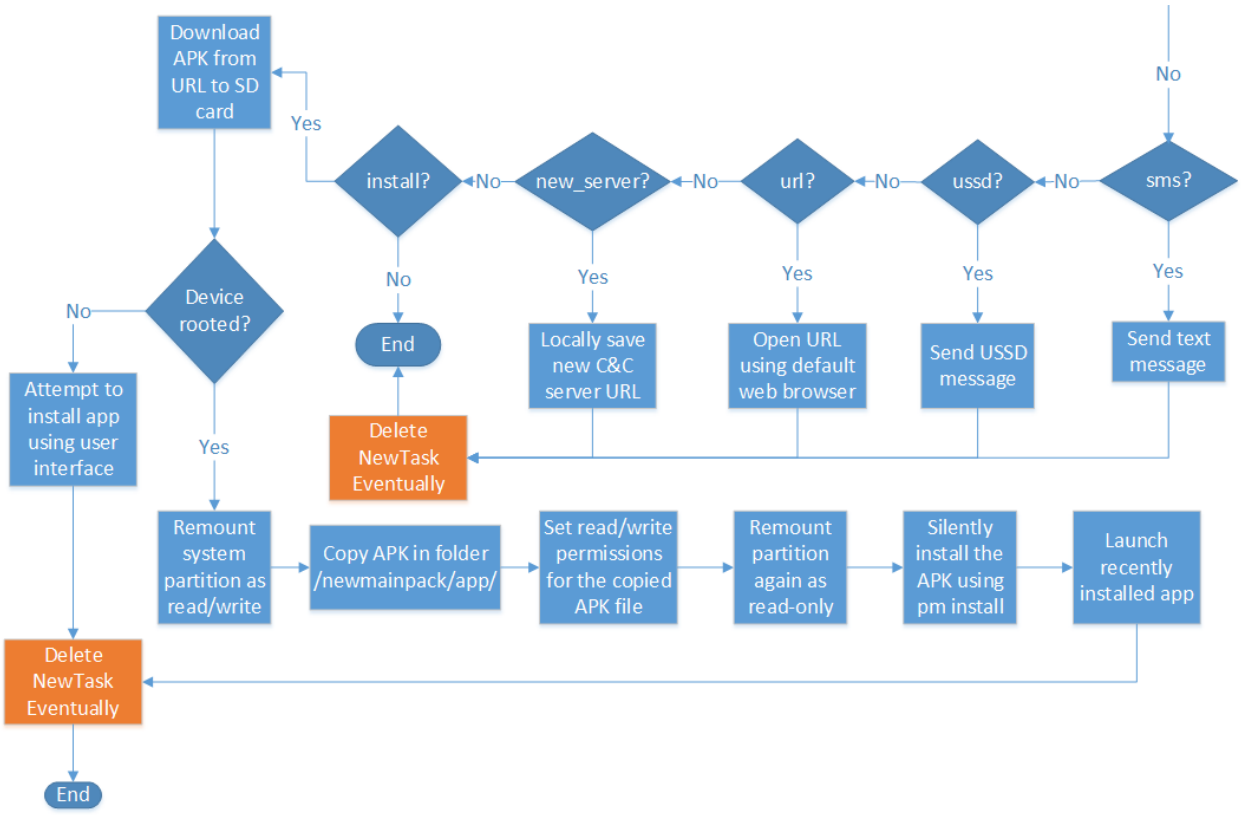

Figure 6. OpFake behavior when a NewTask record is executed

Once the command delivered via Parse cloud is executed, Figure 5 shows that the task is saved in the TaskManager table to later update the record with the response of the task (if any).

Figure 5 and 6 shows that Android/OpFake is able to execute any of the following commands present in the NewTasks table or sent in a Push notification to any of the channels in Figure 4:

- intercept: Sets a flag (on/off) that later will be used to decide if an incoming SMS message should be leaked to a remote C&C server.
- sms: Send a text message. The content (destination and content) is present in the NewTasks record.
- ussd: Send a USSD message using the URI "tel:"
- url: Open the URL provided by the NewTasks record using the default web browser.
- new_server: Locally save the new C&C server URL
- install: Download an APK file from the URL provided by the NewTasks record to the SD Card. If the device is already rooted, the malware will use the admin privileges to silently install the APK as a system app using the "pm install" command. If the device is not rooted, the malware will attempt to trick the user to install the app using the user interface.

Once the task is consumed, the record will be deleted in the NewTasks table in order to avoid the re-execution of the command. Figure 7 shows that in case of an incoming SMS message, Android/OpFake will:

- Save the message in the Parse SmsReceiver table
- Send the message data to the Parse Push channel "T"
- If the intercept flag is activated, the malware will leak the message (and the balance if the SMS comes from a company like MegaFon) to a remote C&C server.

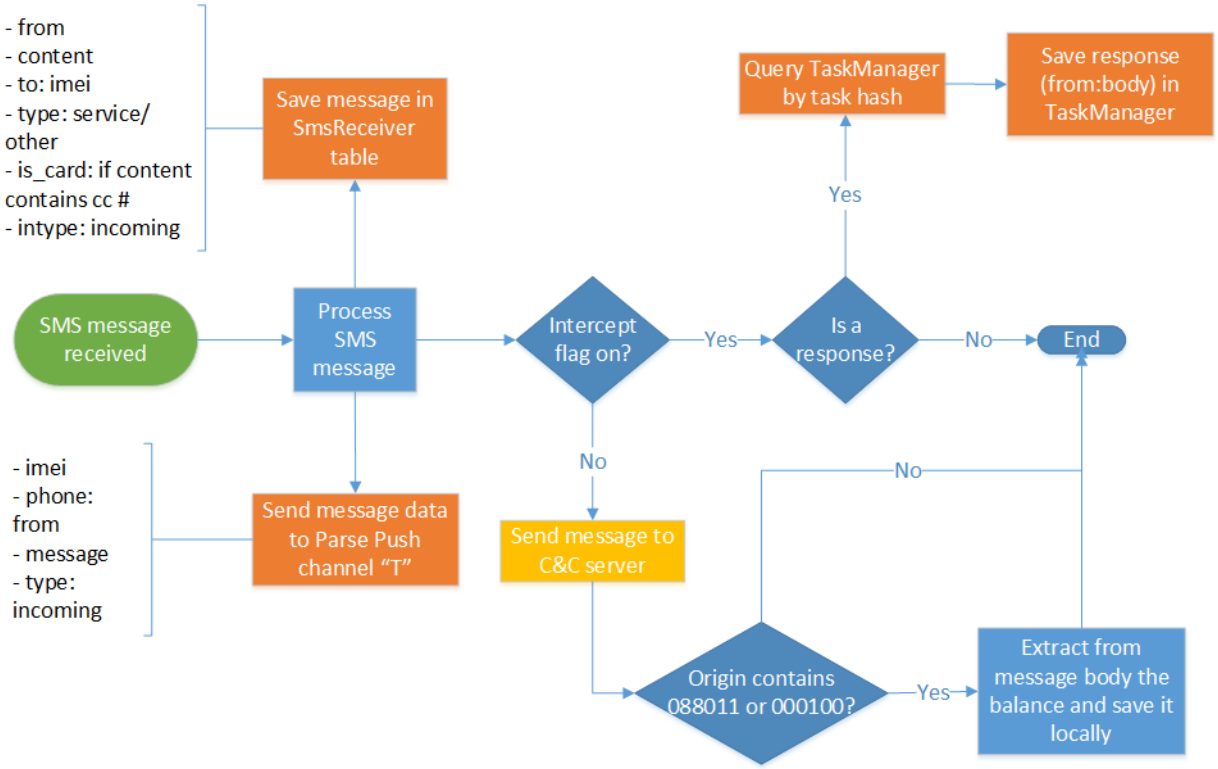

Figure 7. Android/OpFake behavior when an SMS is received

Figure 7 also shows that Android/Opfake checks if the SMS is a response to a SMS previously sent using the NewTask table. If that is the case, the content will be saved in the "response" field.

ANDROID BANKING TROJAN PARSE TABLES

Based on the static analysis of Android/OpFake, we were able to understand the purpose and the data stored on each Parse table:

- NewTasks: Stores new commands awaiting to be consumed by each infected device. Once the command is executed, the record is deleted.
- SmsReceiver: Contains all the intercepted incoming SMS messages received by each infected device:
    - from: origin of the text message (phone number / company name)
    - intype: incoming/outgoing message
    - to: device ID of the infected device
    - is_card: true/false if the message contains a credit card number
    - type: "service" if the origin is a company (e.g. MegaFon) or "other if is another phone number (personal message)
- TaskManager: Stores all the executed tasks plus the response if the incoming SMS message is a response of a previously executed task (like requesting the balance of a specific credit card – more details in the next section).

In total, the malware developers used 5 diverent Parse accounts that contained a schema as described above. 4 of them were used by Android/OpFake and only one, Account D, was used by Android/Marry. In the case of NewTasks, as we learned from the static analysis, once the task is executed, the command-execution record is deleted from the table.

Analyzing the creation date of each record in that table (Figure 8), we found that there are almost no command-execution records until June 25 (except for Account E on June 16), which probably means that all the commands created at that time were successfully executed by the infected devices (or no new commands were created). After June 25, we found several records in all accounts, which suggests that none of them were executed because the records were not deleted. The malware was probably removed from the victim's device. Figure 8 also shows that the impact to the victims could have been greater if all the pending commands since June 25 (and on June 16) were executed by the infected devices.

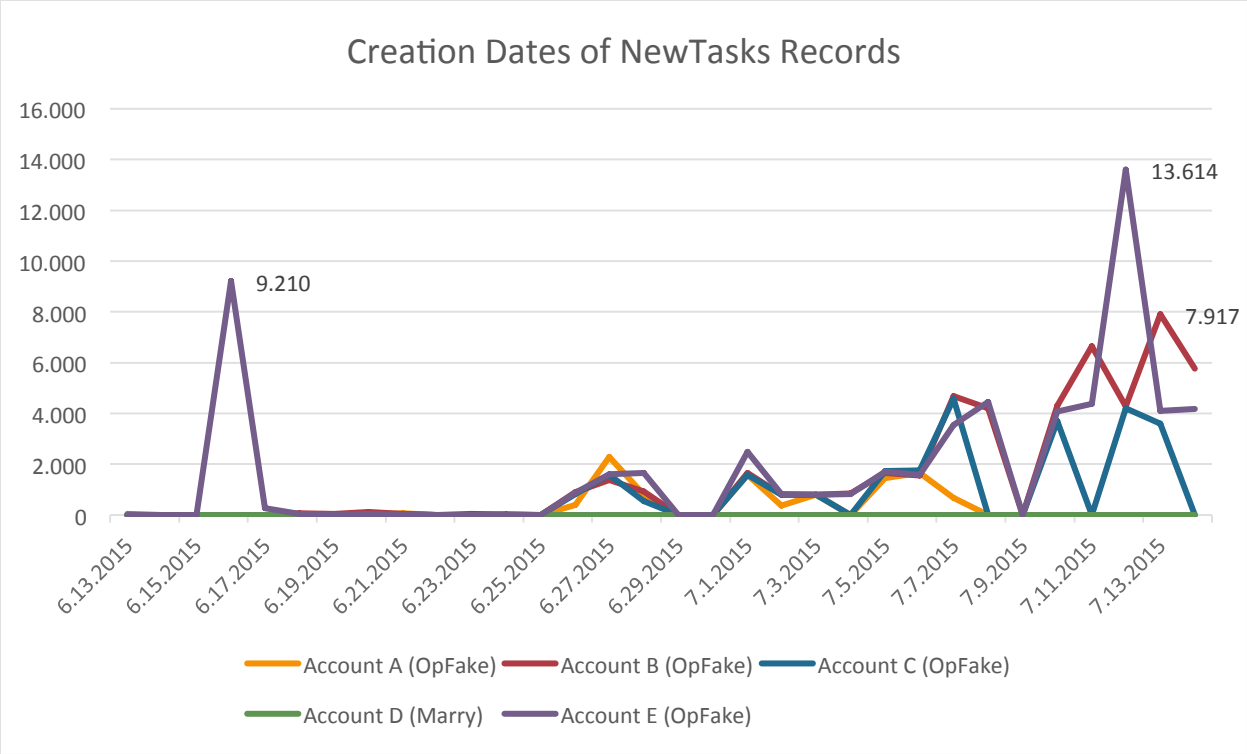

Figure 8. Creation dates of NewTasks records

Figure 9 shows that the most popular command that was pending to be executed is “sms” with almost 50,000/60,000 records in Account B and E (Android/OpFake) respectively. Also Figure 9 shows that Account D (Android/Marry) had few records probably because most of the infected devices were active consuming tasks at the time of the analysis.

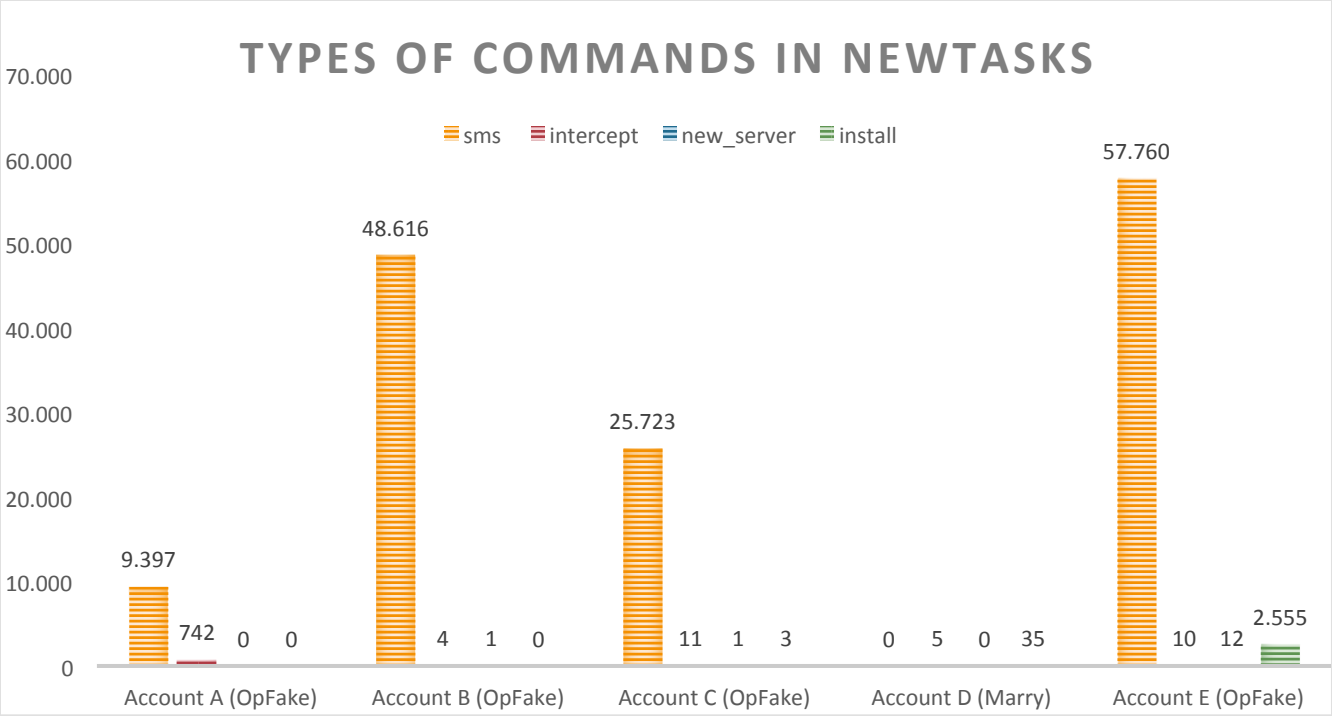

Figure 9. Type of commands in NewTasks

In addition to thousands of "sms" commands shown in Figure 9, here is some examples of commands delivered to the victims but probably never executed (or at least never deleted from the NewTask table):

- new_server:
  - hxxp://newwelcome00.ru
  - hxxp://newelcome00.ru
- install:
  - Android/OpFake delivering Android/Marry:
    - hxxp://newwelcome00.ru/appru.apk (marry.adobe.net.threadsync).
    - hxxp://newwelcome00.ru/app.apk (marry.adobe.net.nightbuid).
  - hxxp://notingen.ru/Player.apk (com.adobe.net)
  - hxxp://швждаыдлпждв

On the other hand, in the case of the SmsReceiver table, Figure 10 shows that Account E (Android/OpFake) was the most active account intercepting and stealing ~60,000 incoming SMS messages followed by Account B with ~41,000 records.

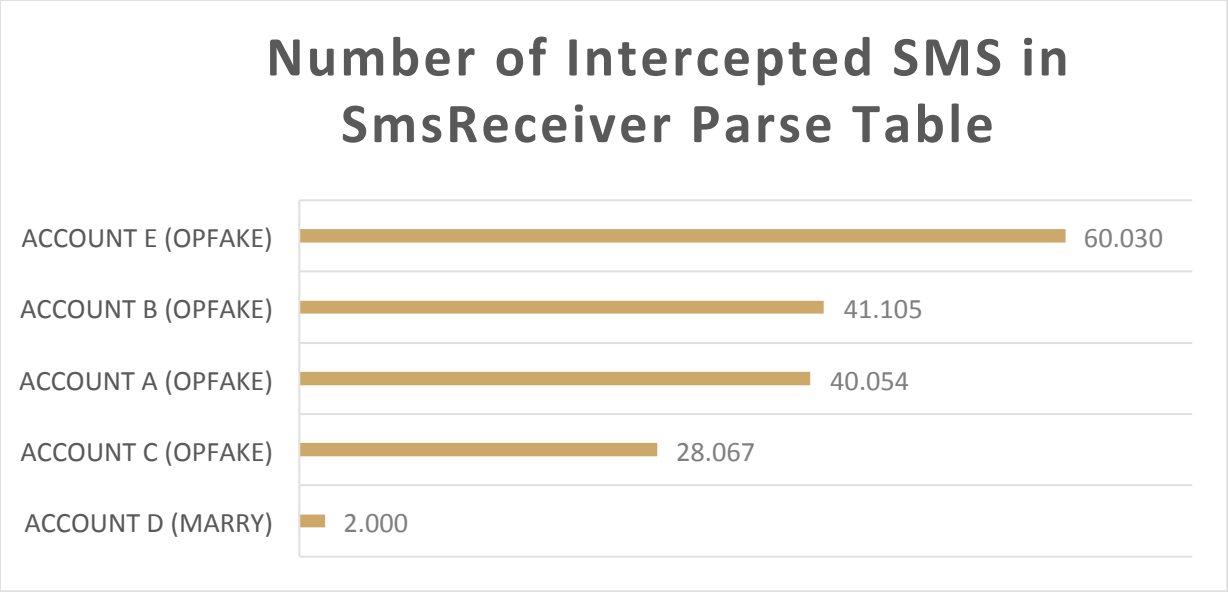

Figure 10. Number of intercepted SMS messages in SmsReceiver Parse table

Figure 10 also shows that Android/OpFake gathered almost 170,000 SMS messages from infected devices, most of them personal messages. This demonstrates that victims were not only impacted financially but their privacy was also invaded by the cybercriminals.

Checking the field is_card in the SmsReceiver table, we were also able to find out how many credit cards were obtained in incoming SMS messages by account (Figure 11).

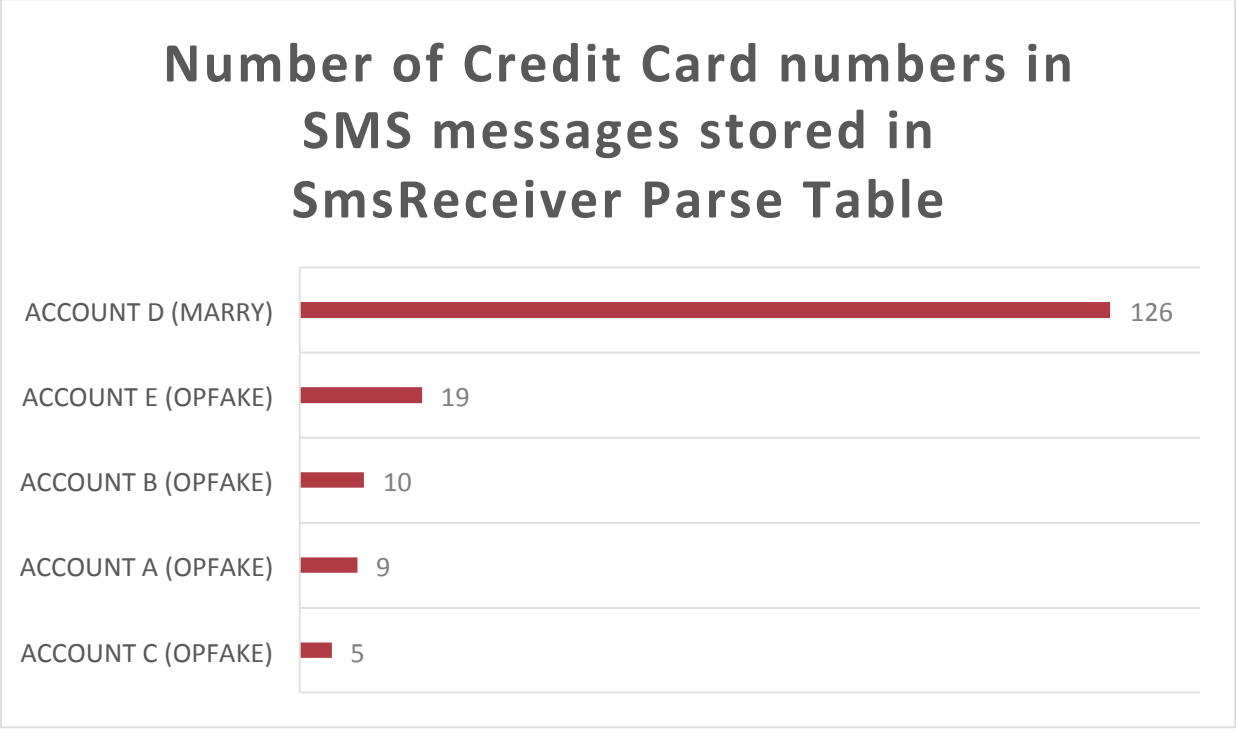

Figure 11. Number of credit card numbers stored in SmsReceiver parse table

As for the dates when the SMS messages were intercepted, Figure 12 shows that all the accounts were most active between June 16 and June 24, intercepting more than 15,000 SMS messages.

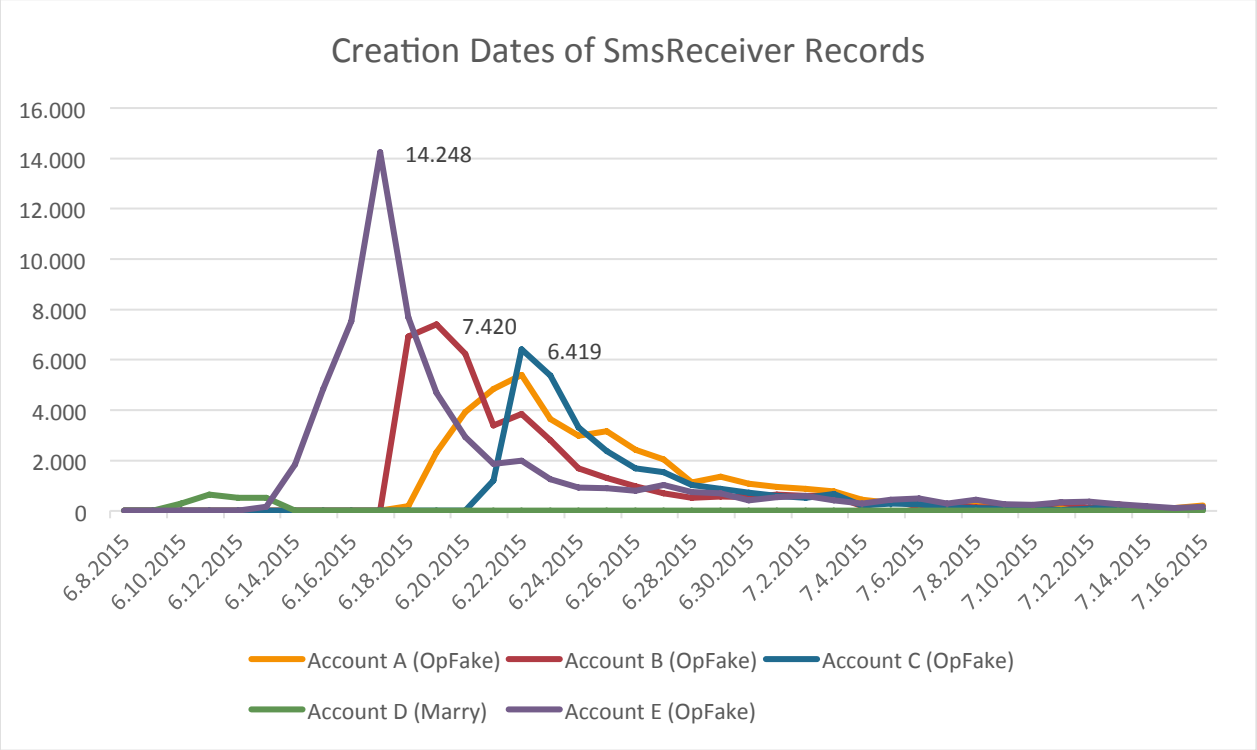

Figure 12. Creation dates of SmsReceiver records

Finally, in the case of the TaskManager table that contains the tasks executed with the corresponding response (if any), Figure 13 shows that Account D (Android/Marry) was by far the most successful in the execution of tasks.

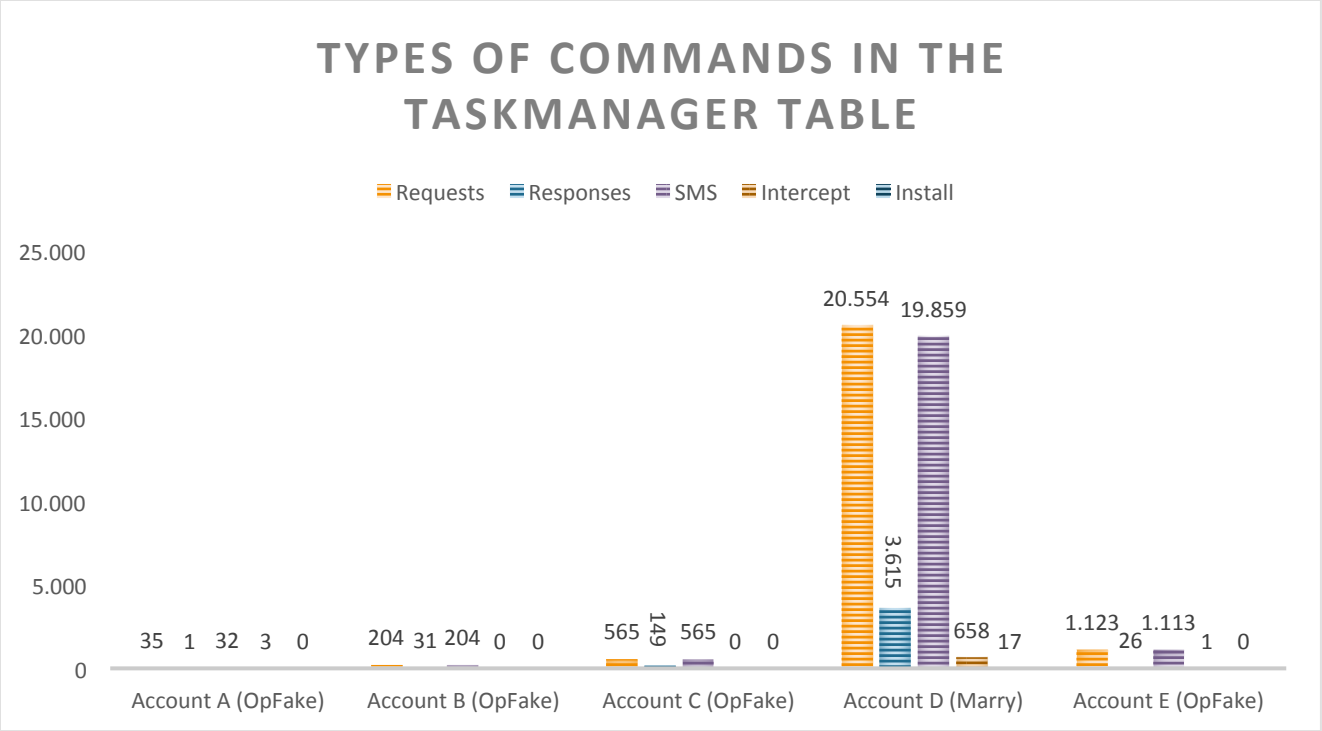

Figure 13. Type of commands in TaskManager table

In conjunction with number of commands present in NewTasks in Figure 9, Figure 13 confirms that Android/Marry was very active at the time that we accessed the exposed accounts and that more than 20,000 commands were successfully executed, most of them SMS tasks primarily for financial fraud.

ANDROID/MARRY

Android/Marry is very similar to Android/OpFake in the sense that both malware families pretend to be "installers" of legitimate apps but, once they are executed, the main icon of the app (shown in Figure 14) is hidden from the home screen and the code execution starts in the background without user consent. In the case of Android /Marry, the malware pretends to be "Flash Player".

Figure 14. Android/Marry icon

As soon as the user opens the app, the icon disappears from the home screen and the malware opens Google Play pointing to the Google Translate app in order to trigger a phishing attack to capture credit card details as shown in Figure 15.

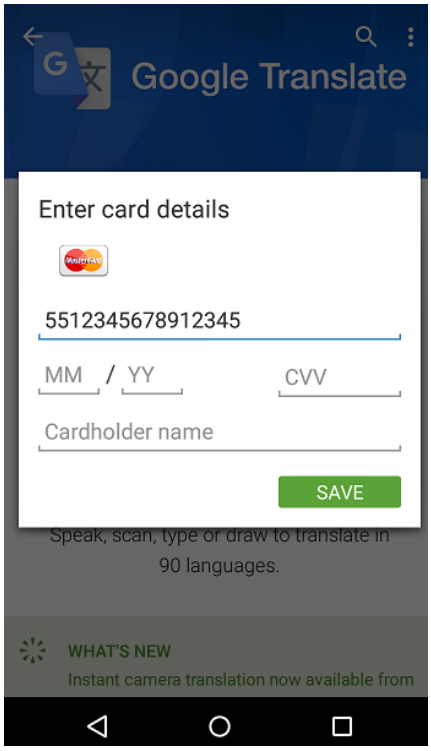

Figure 15. Phishing attack to obtain credit card information

At the same time, several services are started to run in the background as described in Figure 16.

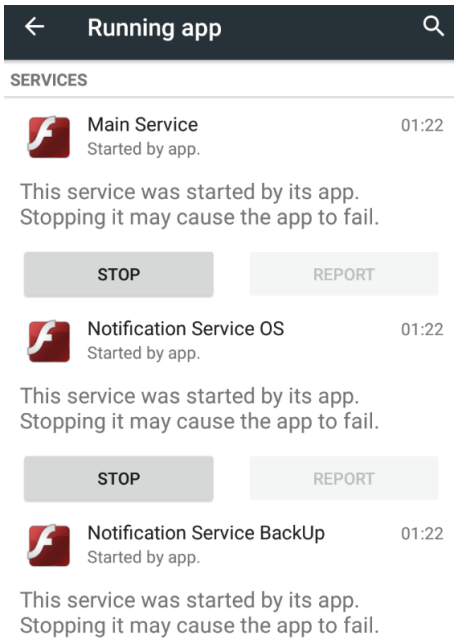

Figure 16. Android/Marry services running in the background

The purpose of the services are:

- Main Service:

    - Subscribe the device to Parse push notifications.
    - Saves device information (including if the device is rooted) into Parse's ParseInstallation table.
    - Leak device information like IMEI, IMSI, SIM Serial Number and phone number to a remote server.
    - Check every 30 seconds with the Parse backend (NewTasks table) if there are new tasks to be executed.
    - If credit card details have not been sent yet, check every 5 seconds if the process com.android.vending (Google Play App) is running to execute the phishing attack and obtain the financial information.
- Notification Service OS: Every 5 seconds check if there are SMS messages (number and content) pending to be sent to all the contacts on the device.
- Service BackUp: Upload the following files to a remote server:
    - /sdcard/[device_id].sms that contains SMS messages present in the device (date, origin, content and type which could be incoming or outgoing).
    - /sdcard/[device_id].con that contains the contact list (display name, phone number)
    - /sdcard/[device_id].apps that contains installed apps on the device (label, package name)

In addition to the functionality above, like Android/OpFake, Android/Marry is also able to:

- Download and install apps (silently if the device is already rooted)
- Update records in TaskManager table with responses from sms commands executed on the infected device
- Intercept incoming SMS messages and store them into the Parse table SmsReceiver
- Open a URL in the default browse
- Send a USSD message using the URI tel:
- Dynamically update the C&C server URL

Financial Fraud In Account D

Because Android/Marry was the malware family that successfully executed the largest number of tasks (due to the number of records in the TaskManager table), we focused our analysis on the commands and responses for Account D. Analyzing the destinations of the SMS tasks we uncovered the top 10 targeted companies by Android/Marry (Account D).

Figure 17 shows all companies (banks and mobile operators) that were targeted by the malware family. One possible reason of being selected to be targeted is the fact that these companies support financial transactions via SMS messages. For instance, if a normal mobile user wants to get information about all its different credit card accounts, the user just need to send an SMS message with the content "INFO" to the number 900 (Sberbank) and the bank will reply with all the requested information. Even concrete transaction-requests, like payments and transfers, are possible. This makes it very convenient for a cybercriminal, since they can "silently" transfer money from one account to another.

Figure 17 also shows that the most targeted company is one of the largest banks in Eastern Europe, Sberbank (Сбербанк), with 5,350 SMS messages sent to the number 900.

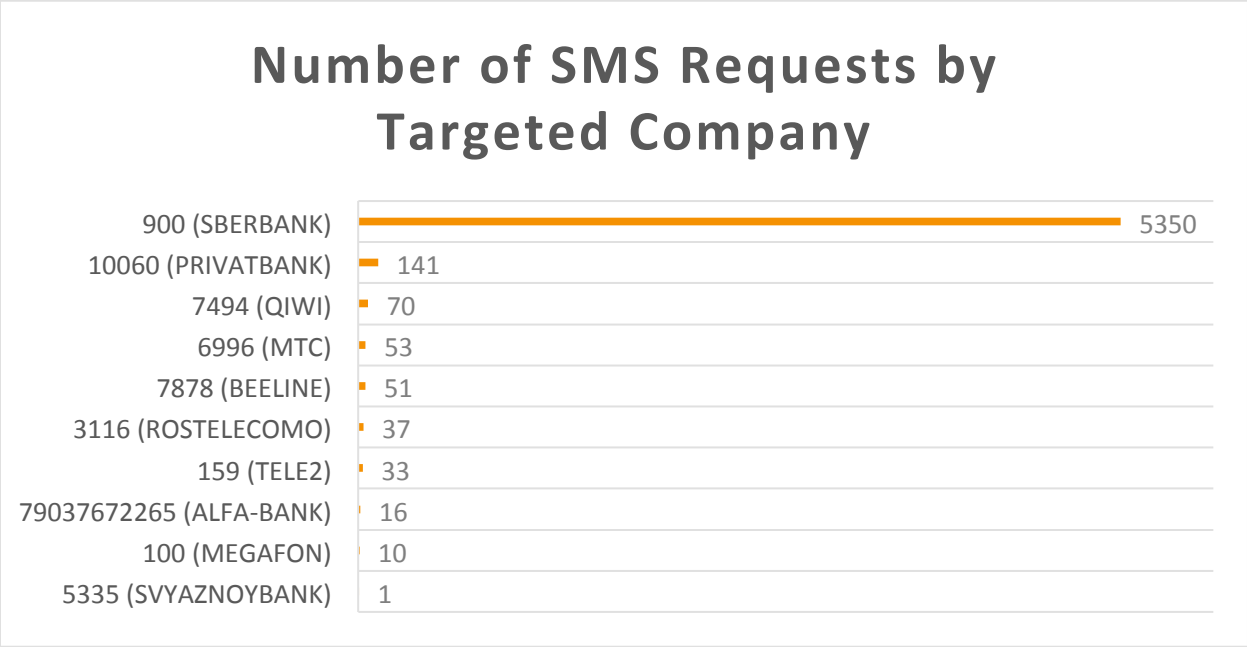

Figure 17. Number of SMS requests by targeted company

We extracted all SMS commands that were used by the cybercriminals that attacked victims with a Sberbank account. Table 1 provides more details about them.

Command	Format	Response
BALANCE/BALANS/баланс	BALANS **<4-last-digits>**	VISA1234 Balance: **<amount>**
INFO/СПРАВКА	СПРАВКА	List of connected cards: VISA1234(ON);
ПЕРЕВОД/PEREVOD/ПЕРЕВЕСТИ (Transfer)	ПЕРЕВОД <4digits_card_origin> <4digits_card_destination> or <phone_number_destination> <amount>	To transfer **<amount>** from card VISA1234 the recipient **<name>** must send the code **<code>** to the number 900
ZAPROS (Request)	ZAPROS <phone_number> <amount>	Request transfer for **<amount>** to your card VISA4321 has been sent. After confirmation by the sender **<name>** the money will go to your account.
TEL/PLATEZ/PHONE/POPOLNI/PLATI (Pay mobile account)	TEL <phone_number> <amount>	To pay with card VISA1234 phone **<company>** **<phone_number>** the amount **<amount>** send the code **<code>** to number 900.

Table 1. SMS Banking Commands

From the information that we have retrieved from the BaaS accounts, we found a common pattern cybercriminals used to steal money from the victim. Financial fraud usually starts by sending an SMS

message with the keyword “INFO” to the number 900 by retrieving this task from the NewTasks table. This step is necessary to collect information about all credit card accounts that support banking transactions via SMS. For instance, if the bank replies with something like VISA1234 (ON), the cybercriminal knows that the VISA1234 account is enabled for banking transactions via SMS. As a next step, the cybercriminals proceeded with checks on the balance of each SMS-supported credit card account. If there is money available, the cybercriminals tried to steal money from the victims account (e.g. through the “Transfer” command (in Russian) or “TEL/PLATI” command). The malware did this by adding a new record into the NewTasks table, which will be executed after a while.

However, in the case of actual transactions, the bank will reply with a code that the user should send to confirm the transaction as an additional security measure. The cybercriminal checks the TaskManager table to get the code and creates a new record in NewTasks to send the confirmation code to the number 900. Figure 19 demonstrates an overview of the usage of executed SMS commands. The BALANCE request is the most executed command followed by INFO.

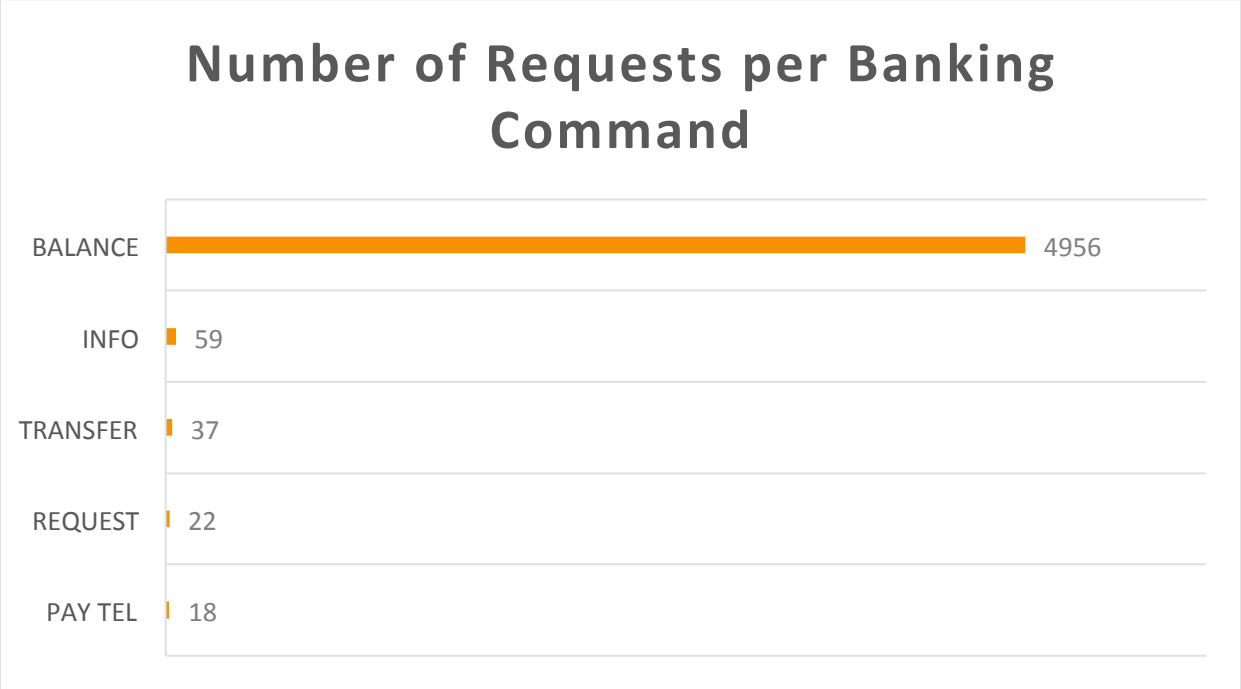

Figure 18. Number of requests per banking command

On the other hand, the responses that we found for Account D from Sberbank are shown in Table 2.

Type	Response
Balance	VISA1234 Balance: **<amount>**
Info	List of connected cards: VISA1234(ON);
Tel Asked	To pay with card VISA1234 phone **<company>** **<phone_number>** the amount **<amount>** send the code **<code>** to number 900.
Tel Processed	VISA1234 **<date>** **<time>** payment for services **<amount>** **<operator>** **<phone_number>** Balance: **<amount>**
Transfer Processed	MAES1234: Transfer **<amount>** to the card recipient **<name>** is processed
Transfer Accepted	VISA1234: <time> Amount **<amount>** from the sender **<name>** received. Balance: **<amount>**
Transfer Asked	To transfer **<amount>** from card VISA1234 the card recipient **<name>** should send the code **<code>** to number 900.

Table 2. SMS Banking Responses

We can group the type of responses by category:

- "Balance"/"INFO": Contains general information like credit cards linked to the banking account, which ones are enabled and the current balance for the active ones.
- "Asked": Responses that includes the confirmation code that should be sent by the user to confirm the transaction.
- "Processed": Contains confirmed fraudulent transactions.

Figure 19. Number of Sberbank responses in TaskManager

Figure 19 shows that Balance is the most popular response with 607 credit card balances successfully obtained. It is followed by INFO with the list of connected credit cards that belong to 123 banking accounts. In total, 141 fraudulent transactions (Pay Tel and Transfer) were performed during June and July 2015.

Number of Users Affected by the Android Banking Trojans

Because each record had a unique device identifier (IMEI or android_id) we were able to get the number of victims affected per table and per account. This information is represented in Figure 20.

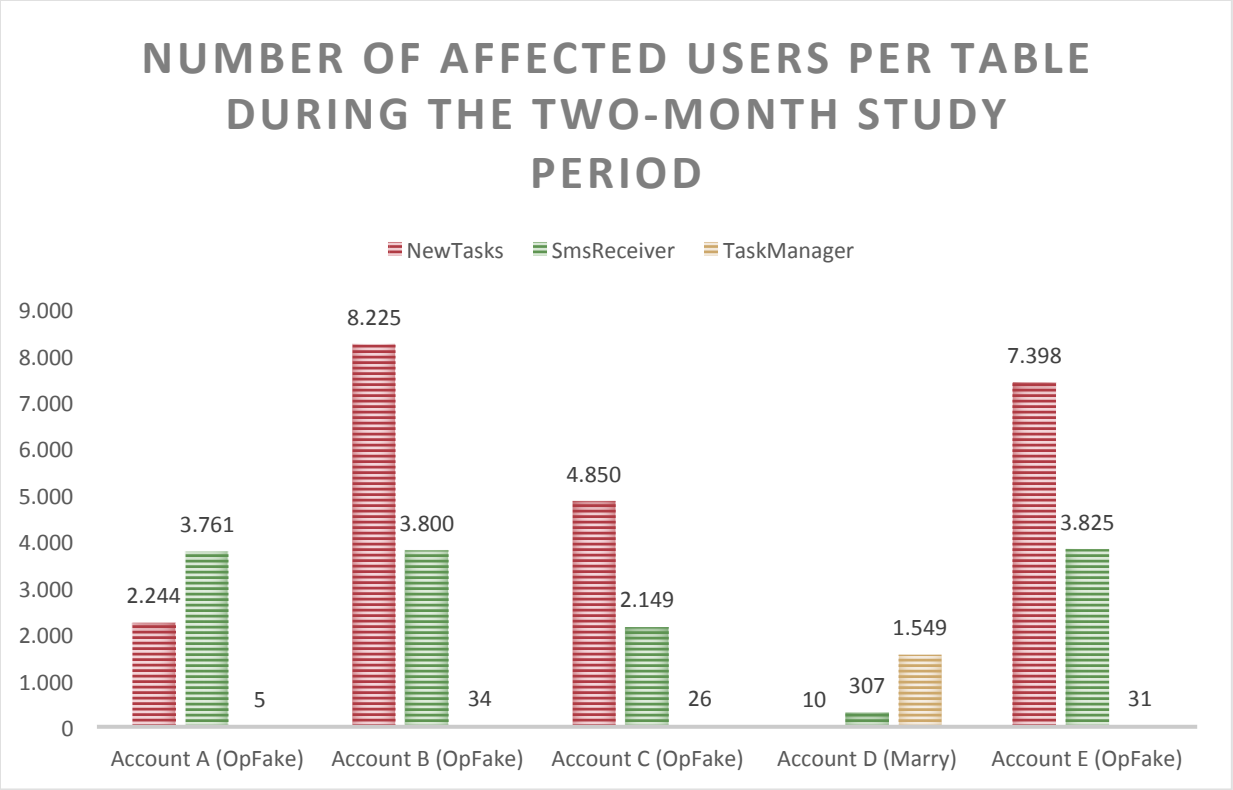

Figure 20. Number of affected users per table

Figure 20 shows three different facts: First, the number of users that could have been affected is much higher compared to the number of users that were actually affected. This can be seen if one compares the "NewTasks"-bar (recap that every successfully executed task will be removed from the "NewTasks" table) with the "TaskManager" (recap that all executed tasks will be added to the "TaskManager" table) bar. Secondly, we can see that Android/Marry affected more users performing fraudulent transactions via SMS messages (1549 TaskManager-tasks vs. 10 NewTask-tasks). Third, Android/OpFake was very successful in intercepting SMS messages based on the number of affected users in SmsReceiver table. In total, thousands of users, apparently most of them located in Eastern Europe countries, were affected by these two Android Banking Trojans.

Responsible Disclosure and Final Thoughts

On August 3, TU Darmstadt/Fraunhofer SIT reported the finding to Facebook and on August 6 Facebook blocked all exposed Parse accounts used by Android Banking Trojans. The data obtained from those accounts proves that Android Banking Trojans are a real threat currently affecting thousands of users and several companies like banks and mobile providers, especially in Easter Europe countries where financial fraud via SMS messages is actively being done via malicious Android apps.

Also the data shows that financial fraud via SMS messages is actually a real threat currently affecting hundreds of users. On the other hand, the analysis shows that malware creators are like legitimate developers in the sense that, as demonstrated by Android/OpFake and Android/Marry, they are focused on the functionality of the app rather than the security of the data collected or used by the malware.

In the case of non-malware apps, there is not much that the users could do to protect the data managed by legitimate apps because it is difficult for normal users to know if the app is using BaaS and if it is being implemented correctly without exposing their sensitive data. In order to reduce the exposure of personal data in BaaS solutions, one alternative could be the use of well-known apps that have been validated for security by a trustworthy third party. TU Darmstadt/Fraunhofer SIT were in contact with BaaS providers to reach out to the developers in order to inform them about their security issues. However, in the end it is their choice to fix their apps and follow the security guidelines of each BaaS provider.