# Preparing for Malware that Uses Covert Communication Channels: The Case of Tor-based Android Malware

Fragkiskos – Emmanouil Kioupakis[1] and Dr. Emmanouil Serrelis[2]
[1]Business Data Analytics, AMC Metropolitan College, [2]AMC Metropolitan College
74, Sorou st. 151 25, Amaroussio, Greece
fkioupakis@amcstudent.edu.gr
eserrelis@metropolitan.edu.gr

## ABSTRACT

The usage of the Tor network, has introduced a new malware paradigm that could also threaten mobile security and privacy. In the recent past months there have been only two cases of Tor-enabled malware spotted in the wild. In both cases, the malware used Tor for secondary operations and was not built around Tor from the ground up, limiting the potential impact. A more sophisticated malware that could use Tor as its core communication channel would have increased its impact and potentially the period of its illegal operations while making its detection significantly harder. At the same time, no commercially available mechanism has been found embedded in any anti-malware software that can detect Tor connection initiation at its source. This fact has identified this specific type of malware as a significant threat in both personal and business environments. This paper aims to set the principles and provide a proof of concept at design level of a new Tor-based Android malware, along with a related detection mechanism. This effort can prove that the risk can be significant and that such malware can be an actual threat, aiming to drive researchers and anti-malware vendors to include this type of malware in their research for anti-malware techniques.

## KEYWORDS

Mobile Malware, Android, Tor Network, Malware Detection, Privacy, Anti-Malware.

## 1. INTRODUCTION

The rate of usage of Android OS based devices is growing rapidly [1]. At a similar rate, the usage of such devices for security-critical tasks such as online banking transactions and accessing corporate networks, is increased, making them valuable targets for malware practices [2]. To date, significant or large scale attacks involving Android devices have failed though security is proven to be a critical aspect for both individuals and corporate users [3]. Users make use of such devices for a number of tasks ranging from day to day communications to accessing sensitive corporate data. Those factors, in combination with the insufficient security management from the users and the lack of security within the relevant applications (e.g. implementing transaction authentication - TANs - for online banking transactions), make those devices a valuable target for malicious users regardless of their motives. Up until recently, the Android OS model have managed to avert significant attacks from malware mostly due to the lack of attack vectors that contribute to the distribution of infections, and due to the low sophistication of malware applications [3]. The technology named Tor, besides protecting the freedom of speech, has also given the ability to malicious users to proceed to various acts – mostly illegal – with various motives [4],[5]. Recently it was used as a component of malware applications, creating in that manner a new category of threats that requires a new perspective of defense mechanisms for detecting and mitigating those new threats.

## 2. RELATED WORK

To date there have been only two confirmed cases of malware that use Tor as their covert communication channel component. In the first case [6], the malware called “Simplocker” is a ransomware type of malware that encrypts certain type of data located on the smartphone device. The difference between this particular ransomware and others is that “Simplocker” uses a Command & Control (C&C) server operating on an .onion domain and Tor network is used to establish communication with it. The purpose of this communication is for the malware to acquire a list of filename extensions, in order to encrypt the corresponding files on the device. However, this implementation does not exploit the full potential of Tor as a covert communication channel, limiting the potential impact of the malware. With minor changes in the malware payload code, the malware could potentially upload the device’s files to the C&C server and then delete them from the device, making the recovery of the files much harder, either upon infection or later as a means of escalating the pressure against the victim and in order to pay the ransom.
In the second case [7], the malware named Backdoor.AndroidOS.Torec.a is reported to be the first Android Trojan to use Tor. This particular malware constitutes of the “Orbot” Tor Client repackaged with a malicious payload. Upon infection, the malware establishes communication with a C&C server that operates on an .onion domain of Tor network.
The purpose of this communication is to transfer data collected as JavaScript Object Notation (JSON) objects from the Android device to the C&C server. This data include the phone number, the country, the IMEI number, the phone model and the OS version. The malware also includes a class used for intercepting SMSs, sending data regarding Unstructured Supplementary Service Data (USSD) requests to the C&C server, as well as to send a list of all installed applications on the device. The intercepting SMS function is also used to issue commands from the C&C server to the infected device.
To date no commercially available detection mechanisms have been found that are capable of detecting the initiation of Tor connections at its source (i.e. at the Android device level). Those mechanisms (e.g. firewalls) are only capable of explicitly forbidding Tor traffic from passing through them and are usually found only in enterprise environments.

## 3. DESIGNING A NEW TOR-BASED ANDROID MALWARE

### 3.1. Requirements

In order to demonstrate the feasibility of the new Tor-based Android malware paradigm, a number of design and implementation requirements have to be met. Dealing with a great variety and complexity of these requirements could achieve the maximum preparedness from the anti-malware point of view.
Initially, since the infection requires physical access to the device, the time frame required for the infection is considered to be critical. This time frame should be at most equal to the average time that users leave their devices unattended. In this design, the time targeted is less that one minute which is empirically considered less than the one described above. During that time, the attacker will be connected through https to the web server that hosts the malware and install it. To speed that process up, the host’s URL can be written on an NFC tag that can be used with NFC enabled devices in order to avoid the manual access to the device’s browser and the typing of the URL. Another approach would be to use a misleading QR code that could appear as a legitimate source (e.g. Clothing sales link) and instead lead to the aforementioned malware.
Upon infection, the victim should not be in the position to detect the operation of the malware.

For this reason, the malware operations should not interfere with the rest of the device's operations when the victim uses the device. For example when navigating to a menu there should not be any delays or glitches or any other behavior that might make the user suspicious. Hence, while the user browses through the contact list or the call log, there should not be any abnormal activity such as time delays or screen movement glitches, even if the malware is intercepting that list at the same time.

For the same reasons, the malware should not appear in the list of installed applications, nor in the list of running applications. These two requirements are can only be implemented through OS-based exploits and cannot be based on permission exploitation like most of the requirements. Such exploits may differ from an OS version to another. In any case, the malware should be compatible with the most widely used Android version which, according to Google, is version 4.1.x (Jelly Bean) [8].

The malware should be sending the extracted data to the C&C server in specified time intervals, when the victim's smartphone device is connected to the Internet. Though, to ensure the integrity of the malware operation, the extracted data should be sent to the C&C server after a predetermined time threshold regardless of whether the victim's device is connected to the Internet or not. The data to be extracted by the malware include victim's geolocation, call log, contacts list and SMSs list.

As far as the network traffic is concerned, the malware should be using the Tor anonymity network in order to secretly exchange data and control commands between the infected device and the C&C server. In addition, the tracing of the C&C server that the malware uses, should be made very hard due to the use of Tor. Furthermore, the malware should easily operate in enterprise environments dealing with advanced network blocking mechanisms that enterprise security solutions have implemented. For that reason, the Tor traffic of the malware should not be different from any other Tor traffic, hence making it difficult for anti-malware mechanisms to block malware Tor traffic while letting regular Tor traffic to pass through.

**Table 1** Malware requirements categorized according to three different perspectives

| | |
|---|---|
| **Operational** | **Fast Deployment:** Time relevant to the period that a device is typically left unattended by its user: target time < 1 minute<br><br>**Sending Data:** Sending the extracted data in regular intervals when the victim is connected to the Internet<br><br>**Integrity:** Creating a hidden connection to send the extracted data after a defined threshold even if the victim is not connected to the Internet<br><br>**Interoperability:** Compatibility with as many OS versions and devices as possible |
| **Functional** | **Transparency:** The victim does not notice the malware operation when it is running<br><br>**Anti-tampering Mechanisms**: Hardening of manipulating data used by the malware so the victim cannot feed it with false data upon detecting it's presence<br><br>**Interception:** Exfiltration of data needed by the malware (Geolocation, Call Log, Contacts List, SMSs List) |
| **Covertness** | **Presence:** The malware does not show as an installed application or as a running process<br><br>**Activity:** The malware does not trigger any notifications when it is running<br><br>**Traffic:** Usage of Tor network to send data so that the traffic is hidden from network security appliances and software, especially in enterprise deployments<br><br>**Rogue Connection:** Creating the rogue connection based on certain events, e.g. when the device is in idle (screen turned off) |

The malware should include controls to determine when is the most appropriate time to create a covert connection to the C&C server, based on specific events. An example of such event is when the device is in idle mode (e.g.

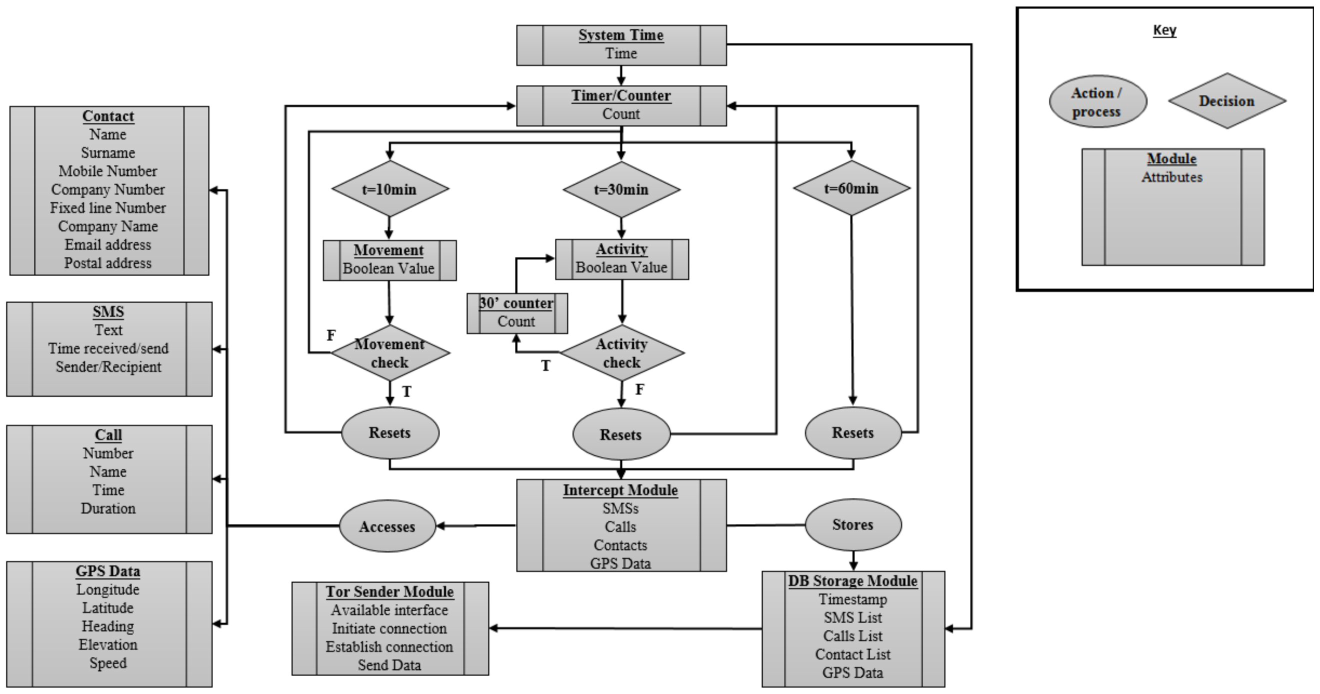

**Figure 1** Architectural diagram of the malware modules

the screen is turned off). This would also achieve the enhancement of the covertness of the operation. For the same reason, when the malware is active it should not trigger any notifications to the user.

To further enhance the robustness of the malware operations, there should be anti-tampering mechanisms implemented, in order to make more difficult for the security analysts to manipulate the data entered to the malware. More specifically, this requirement aims to ensure that no false data are fed to the malware ensuring that the attacker receives only useful data from the device in case of detection.

The requirements needed for designing the malware can be categorized into three broad categories, namely Operational, Functional and Covertness as presented in Table 1. Operational requirements refer to *"identify the essential capabilities, associated requirements, performance measures, and the process or series of actions to be taken in effecting the results that are desired in order to address mission area deficiencies, evolving applications or threats, emerging technologies, or system cost improvements"* [9]. Functional requirements refer to those that “*define the fundamental actions that must take place in the software in accepting and processing the inputs and in processing and generating the outputs*” [10], in our case those needed for the malicious activities. Covertness requirements refer to those needed to ensure the secret operation of the malware.

## 3.2. Module Design

For designing each subsystem (module) and operation of the malware, an Architectural diagram has been made and is presented in Figure 1.

The malware’s operation is highly dependent on proper timing. In order to achieve that, it uses the system time in combination with an internal counter. During malware initialization the internal counter is set to zero. Initialisation could occur either upon infection or upon turning on the device after infection. Ten minutes after initialisation – calculated by the difference between the system time and the internal counter – the malware checks if the victim is on the move. If that condition is true, then the counter resets and the process proceeds to the intercept module.

This process is performed by the malware in order to intercept the victim's geographical location (geolocation) along with the changes made to the call log, the contacts list and SMS list.

If the application exits the initial ten-minute loop, this indicates that the victim is either not moving or he/she was moving and now has stopped. In this case, at a thirty-minute mark, indicated again by the internal counter, the application enters a second loop which checks whether the victim is using the device or not. If that condition is false, the application proceeds to the intercept module and resets the counter so the process begins again. If that condition is true, which indicates that the victim is using the device at the thirty-minute mark, the application enters a secondary loop which also includes a secondary internal counter. That secondary internal counter awaits for another thirty minutes for the victim to stop using the device in order to proceed to the intercept module.

If the victim continues to use the device for the thirty minutes counted by the secondary internal counter, the application exits that loop and enters the last loop which at the sixty-minute mark, resets the primary internal counter and proceeds to the intercept module with no further checks. This process aims to send the extracted data to the attacker at most within an hour regardless of conditions ensuring that these will be ultimately communicated no matter the usage circumstances. This requirement is related to the robustness of the malware.

Following any of the above cases, the application then proceeds to the intercept module, seizing the geolocation data, the call log, the contacts list and the SMSs list. Next, the data are received by the storing subsystem (storage module) which handles the storage of the intercepted data to the database included in the Android OS. This process is implemented so that all data can be manipulated as a single entity. Also when storing the data, a timestamp will also be included (according to the system time) in order of the attacker to be aware of the exact time of each data intercept cycle. In this manner, the attacker would also be aware if any of the data intercept cycles was delayed, skipped or otherwise altered.

Subsequently, the entity created by the storage module will be forwarded to the Tor subsystem (Tor module) which in turn is responsible for the connection initiation and session establishment with the C&C server. Upon that, the intercepted data will be sent to the C&C server and that completes that data interception cycle.

## 4. DESIGNING A NEW ANTI-MALWARE FOR TOR-BASED ANDROID MALWARE

### 4.1. Requirements

There is a number of design and implementation requirements that need to be met to demonstrate the feasibility of the related anti-malware for Tor-based Android malware such as the one specified above. As of August 2014, no commercial detection mechanisms have been found that are capable of detecting Tor-connection initiation at its source (i.e. within Android devices).

Various static and dynamic detection methods can be used to detect the malware behavior of such malicious applications [11] as shown in Figure 2.

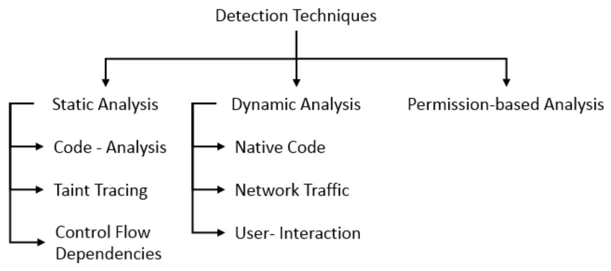

**Figure 2** Malware detection techniques categorized by type

Of these, static and permission-based analysis are not appropriate due to the offline nature of their implementations. Since a real-time detection implementation is required, dynamic analysis is the only appropriate method for detecting such a malware. The suggested detection methods can be classified, according to Figure 2, as network traffic analysis, since it detects connections with specific destination ports as described in the corresponding section below. Moreover, a different detection approach is introduced, which includes the detection of a string, transmitted during a Tor connection initiation process which cannot be classified as pure network traffic analysis methodology. To add to this, a native code analysis would fail to detect such a malware since the core code itself is not malicious and since the malicious processes are typically based on permission exploitation. However, regarding offline implementations, other approaches could be followed upon detection and extraction of the malware. Static code analysis would reveal to an experienced researcher the true nature of the application and the way it intercepts and sends sensitive personal data. Also, permission-based analysis would reveal the exploitation of permissions which leads to the malicious behavior of the application.

One of the most critical parts of the anti-malware is the one that will detect the initiation process of a Tor connection. Upon detection of such activity, a decision-making subsystem will determine whether the application requesting the Tor connection would be granted permission to establish this connection or not. The decision making process takes into account whether the Android process requesting the initiation of a Tor connection is generated by signed application. If this is not the case, no permission would be granted and the connection would be automatically blocked. Still, even when the application is signed, the anti-malware will prompt the user whether he/she permits the application to initiate a Tor connection or not. The anti-malware could include a user and/or vendor customizable application whitelist that should include all applications authorized to use Tor, in order not to block unsigned applications that the vendor or user trusts and not to prompt the user for signed applications that have been authorized in the past by the user.

**Table 2** Anti-malware requirements categorized according to four different perspectives

| | |
|---|---|
| Operational | **Seamless experience:** The anti-malware requires minimum feedback from the user<br><br>**Device impact:** The anti-malware does not drain the battery or does not make heavy use or resources<br><br>**On-event action:** The anti-malware acts only when needed, i.e. upon detection of Tor traffic<br><br>**Interoperability:** The anti-malware is compatible with as many Android OS versions as possible<br><br>**Non-conflicting:** The anti-malware does not cause any conflicts with other anti-malware applications |
| Functional | **Traffic Detection:** Detecting Tor traffic upon connection initiation even from trusted applications<br><br>**Traffic Blocking:** Blocking Tor traffic by default for non-signed applications<br><br>**Traffic Authorization:** Prompt user to grant or deny Tor connections from signed applications |
| Performance | **False alarms rate:** The anti-malware should have low rate of false alarms, target < 5%<br><br>**Effectiveness rate:** The anti-malware should have high rate of successful detection, target > 95% |
| Version-specific | **Service-enabled version:** The anti-malware runs in the background, dynamically scanning all applications that the user runs<br><br>**Standalone version:** The anti-malware runs once on demand checking the applications that are running at that time |

The anti-malware operation should be transparent to the user. Only notifications regarding the blocking of an application that tries to initiate a Tor connection and the prompt for asking permission for signed applications should be generated. Limiting the notifications towards the user would minimize user annoyance caused by unnecessary information, while also making more efficient usage of the system resources. In fact, the only time that this anti-malware would be taking any action would be when detecting a Tor connection being initiated.

For further enhancing the efficient usage of system resources, the anti-malware should be in a constant idle-standby state and exiting that only when taking action is required (i.e. a Tor connection initiation by an application). The scanning subsystem should remain active even when the device is in an idle state (i.e. screen turned off), since a malware could be initiating the Tor connection at exactly that state to increase its covertness.

The anti-malware should aim not to exhaust the battery or make heavy use of system resources by deactivating specific modules based on various events. An example of this includes deactivating all components when the device is in “flight mode”. Subsequently, the module for traffic scanning could be divided in separate sub-modules equal to the number of different interfaces found in various devices in order to deactivate a certain sub-module when the corresponding interface on the device is also deactivated.

Furthermore, the anti-malware operation should not cause any conflicts with other anti-malware software running on the same device and should be compatible with as many Android OS versions as possible, starting with the most widely used version which, according to Google, is v.4.1.x (Jelly Bean) [8].

Also, the anti-malware could come in two different versions, one that would be running constantly as a background service, providing real-time protection and a standalone version that would be used on-demand, scanning running applications when the user suspects that there is a Tor-enabled malware application installed on his/hers device.

Finally, as far as performance and reliability is concerned, the anti-malware should not give a false alarm rate of more than 5% while not failing to detect Tor connection initiations in a rate greater than 5% as well. These percentages are close to the average industry performance and they are a direct measure of reliability [12].

The requirements needed for designing the anti-malware can be categorized into four broad categories namely Operational, Functional, Performance related and Version-specific, as presented in Table 2. These requirements refer to those needed as an application in general, those needed for the anti-malware functions, those related with application performance and those for the different versions of the anti-malware respectively.

### 4.2. Module Design

For designing each subsystem (module) and operation of the anti-malware, an Architectural diagram is created and presented in Figure 3.

The anti-malware implements a listener/handler module which communicates with the network layer which in turn communicates with the physical interfaces that are present on the device. The listener/handler module acts only when it monitors the traffic and detects any Tor traffic being initiated on one of the interfaces.

Upon detection, the module checks which application is requesting to initiate the Tor connection and checks if that application is included in the application whitelist.

During inspection, if the application is indeed included in the whitelist, the listener/handler module takes no further action. In case the application requesting to initiate the Tor connection is not in the whitelist, then the anti-malware application proceeds to the next inspection step.

Any application that requests to initiate a Tor connection that is not included in the safe-list of applications authorized by the user, will be inspected to determine whether that application

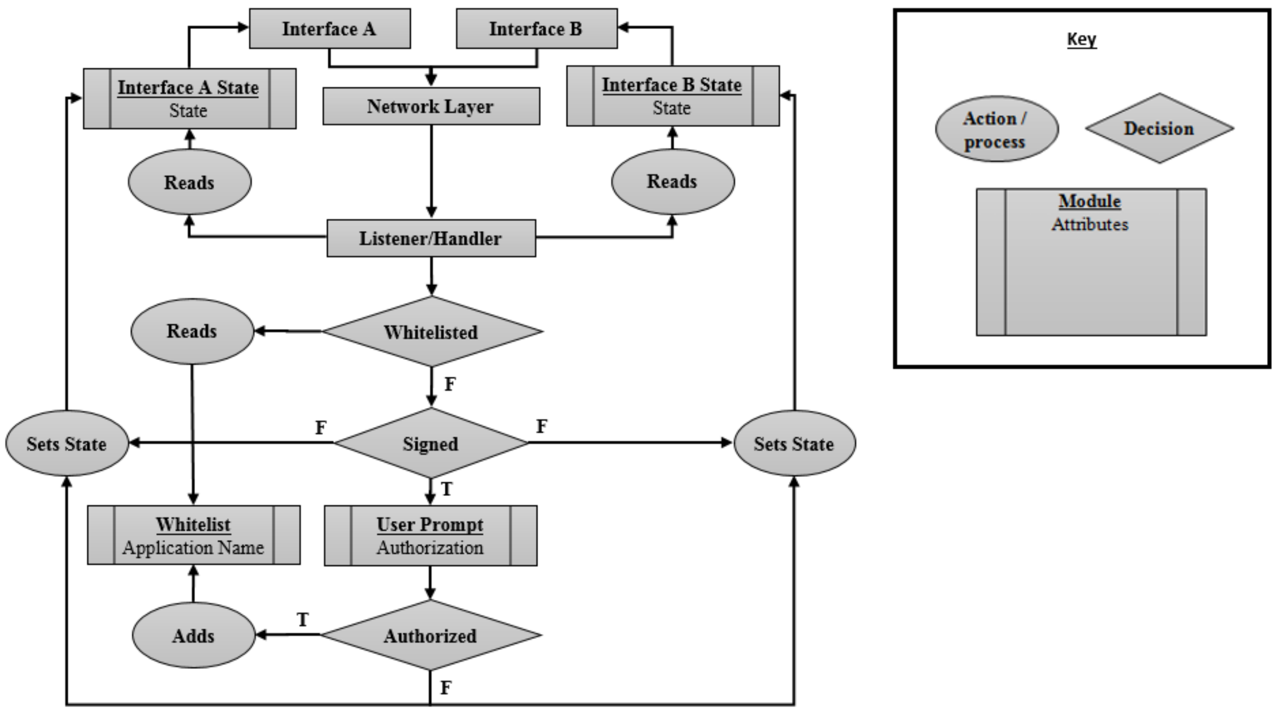

**Figure 3** Architectural diagram of the anti-malware modules

is signed or not. In case the application is indeed signed, the anti-malware application will prompt the user asking to authorize that application to initiate a Tor connection. If the user authorizes that application, then it will be added to the safe-list of applications in order not to prompt the user again in the future for the same application and the anti-malware application will take no further action, allowing that application to proceed with initiating the Tor connection. In case that the application requesting to initiate the Tor connection is either not signed or the user does not authorize it when prompted, the anti-malware application will deactivate the interface that would be used for the Tor connection in order of the user to investigate that application and uninstall it if needed.

Furthermore, the anti-malware listener/handler module will be constantly monitoring the state of each interface and deactivate the corresponding scanning sub-module if an interface is deactivated in order to make use of system resources more efficiently.

## 5 IMPLEMENTATION ELEMENTS

### 5.1. Implementing the Malware

Initially as described above and as shown in Figure 1, the malware is highly dependent on the timing structures. For those timing structures the class android.os.SystemClock is used. This class includes three methods for providing system time. These are: System.currentTimeMillis(), uptimeMillis() and elapsedRealtime() [13]. For implementing the timing structure, the third method will is preferred, since the first one provides the "real world" time which is prone to changes by the user, the cellular network or other factors. The second method will also be avoided since it stops counting when the device goes in idle (e.g. screen turned off, CPU idling). In contrast to the two previous methods, the elapsedRealtime() starts counting upon OS booting and continues to count even when the device is idling and until the device is switched off. That method will provide the malware operations with a trustworthy timing source while the internal counter, initially set at zero, will count the difference with that method's value. The internal counter is acting as a timing parameter that can be reset on demand.

Due to the multiple types of data that the malware intercepts, the intercept module uses various classes and methods from the corresponding APIs. To intercept the SMSs list, the classes named Telephony.Sms.Inbox and Telephony.Sms.Sent will be used. The relevant classes named Telephony.Sms.Conversations, Telephony.Sms.Draft and also the class Telephony.Sms.Outbox should not be used, since the first one restricts the way SMSs are categorized. Moreover, the last two classes do not provide any valuable data since those SMSs will either be moved to the "Sent" folder or will never be sent.

For intercepting the recent call log, just the class CallLog.Calls is sufficient while intercepting the contacts list the class ContactsContract.Data can be used.

Using the android.location API, a number of malware functions can be implemented. With the onLocationChanged method, it can be determined when the victim is on the move, which is necessary for implementing the ten minute mark loop. Especially for the intercept module, the getAltitude, getBearing, getLatitude, getLongitude and getSpeed methods of that API will be used in order to determine the geolocation and the related data. Determining the device's speed can also determine whether the victim is on foot or using some other means of transport which can prove valuable for the aims of such a malware implementation.

As far as the intercepted data storage module is concerned, the SQLiteOpenHelper is chosen. This makes use of the database that the Android OS incorporates for applications to use. For that particular module, common database functions will be used such as using the getWritableDatabase method for creating and opening a database to store the intercepted data and the insert method to insert rows.

## 5.2. Implementing the Anti-malware

The core component of the anti-malware is the listener/handler module which includes a packet sniffing subsystem and a decision making subsystem. Since the Android OS vendor (i.e. Google) does not natively allow access to the network layer, the packet sniffing subsystem can be implemented by incorporating a third party open-source packet sniffing software. For the decision making subsystem the Tor connection detection mechanism will be implemented by detecting traffic using destination ports 9001-9004, 9030-9033 and 9100 along with a string that is transmitted during Tor connection initiation [14]. As mentioned above, the anti-malware will proceed to the next step only upon detection of a Tor connection initiation request where it will query the safe-list of applications to determine whether the application requesting the Tor connection is included or not. To implement the safe-list of applications, the SQLiteOpenHelper class will be used which includes the getWritableDatabase for creating the database for the safe-list and the insert method for adding applications (i.e. records) to the whitelist.

For applications that are not in the whitelist, the anti-malware will check whether the application requesting to initiate a Tor connection is singed or not. To implement that checking structure, the getPackageInfo method will be used. This is included in the PackageManager class.

As described in the corresponding sections above, when the anti-malware detects a Tor traffic connection being initiated and is neither on the safe-list of applications nor granted by the user when prompted, the anti-malware will deactivate the corresponding interface so the user can inspect that application and remove it if needed. To implement the structure which will be deactivating the interfaces, the WifiManager and the ConnectivityManager classes will be used. Those classes include the setWifiEnabled and stopUsingNetworkFeature methods for deactivating the Wi-Fi and GPRS/3G/4G interface respectively.

As mentioned above, the listener/handler module will be monitoring the state of the interface in order to deactivate the corresponding sub-modules when needed. To implement the interface state monitoring structure, the method getWifiState of the WifiManager class and the getNetworkInfo method of the class ConnectivityManager will be used for monitoring the Wi-Fi and GPRS/3G/4G interface respectively.

## 6 FEASIBILITY

The design principles for the malware application are based on widespread techniques used often by application developers. Most of the operations described above are based on permission exploitation, which makes the malware compatible with multiple versions of Android OS. The two greatest challenges for materializing such a malware are the Tor component and the propagation/infection methods. The challenge regarding the Tor component includes incorporating open-source Tor modules with the rest of the code. Nevertheless, that has been already done, as mentioned in sections above, by incorporating the whole Tor Orbot application code along with the malicious payload. Some malware functions require the use of specific OS-based exploits for better covertness but a number of them are already known. Regarding the propagation/infection methods, the challenges that arise depend on the aims of the attacker as the malware can be used as a targeted spyware similar to the concept of an Advanced Persistent Threat (APT) or as a more widely distributed malware. In the APT case, the type of intercepted data can be characterized as valuable and can be used in many different ways. For enhancing the propagation/infection methods, various exploits for remote exploitation can also be used. In all cases, once a device is infected, it should be able to operate for a significant amount of time before being detected, especially when it comes to average skilled users. All issues above prove this specific type of malware achieves a high degree of feasibility. The presented information also proves that the threat is plausible; hence there should be a high degree of alertness among information security professionals as well as related anti-malware vendors.

As far as the anti-malware is concerned, most of the design principles used above also use common application developments methods to be implemented. Again the main challenge that arises regards the incorporation of third party open-source code for the packet sniffing structured. This is an essential requirement since the OS vendor does not allow native access to the network layer, although certain groups of researchers may have access to different resources such as the source code of the detection mechanisms that various mobile security software vendors own and use. Besides that, and after incorporating such as mechanism with the rest of the code, an anti-malware application similar to the one specified and designed above, could easily be the first line of defense against threats that use covert communication channels, such as Tor.

## 7 CONCLUSIONS

### 7.1. Synopsis

The threat that Tor as a technology poses, is existent and in many cases underrated. This paper achieved to present a possible specification and design of a malware that uses Tor technology to disclose information from Android mobile devices. This effort can be seen as a proof-of-concept work, showing that defense mechanisms against technologies that can be used in a malicious manner against our private and enterprise security, such as Tor, prove to be inadequate. Appreciating the fact that today's information translates into power and competitive advantage, the mitigation of new threats and the development of appropriate and sufficient defense mechanisms is considered to be critical. This threat has already become a reality at least twice in the past and has been confirmed for a third time, through this paper. All these facts signify a new trend for malware threatening mobile devices. For the defined scope this may seem to only be applicable to one OS, the foundations have been placed for other OSs as well, aiming to contribute to create sufficient defense mechanisms against an already existent threat. The nature of open-source software allows and encourages the creation of new software products but likewise, new threats can arise using the same techniques and technologies. Therefore it is imperative that information security professionals need to mitigate sufficiently. The very same nature of open-source software that can pose threats, can also be proven a valuable ally against the challenges that arise when creating defense mechanisms that mitigate threats that threaten private and enterprise environments.

### 7.2. Future Work

Researchers taking the next step beyond the research described here, would implement a working example using the specification and the design principles analyzed above, hence creating an operational proof-of-concept that may drive a higher alertness and additional conclusions regarding the handling of such threats. Several optimizations may be included in both the malware and anti-malware application. Examples of such optimizations would include refinement of the way the anti-malware blocks the Tor connection upon detecting it and making use of OS-based exploits in order to increase the feasibility of the malware at a proof-of-concept basis. This research could also expand to create further awareness for security software vendors about this threat that has been active and currently developing. This could eventually lead to the inclusion of Tor-specific security mechanisms in their products.

## 8 REFERENCES

[1] Gartner, 2014. Gartner Says Annual Smartphone Sales Surpassed Sales of Feature Phones for the First Time in 2013. [Online] Available from: <http://www.gartner.com/newsroom/id/2665715> [Accessed November 16, 2014]

[2] ESET, 2013. Trends for 2013: Outstanding growth of mobile malware, ESET

[3] Rafael Fedler, Christian Banse, Christoph Krauss, and Volker Fusenig. 2012. Android OS Security: Risks and Limitations, Fraunhofer Research Institution AISEC

[4] Tor Project: Overview [Online] Available from: <https://www.torproject.org/about/overview.html.en> [Accessed November 16, 2014]

[5] Trendmicro, 2014. Defending Against Tor-Using Malware, Part 1. [Online] Available from: <http://blog.trendmicro.com/trendlabs-security-intelligence/defending-against-tor-using-malware-part-1/> [Accessed November 16, 2014]

[6] Robert Lipovsky, 2014. ESET Analyzes First Android File-Encrypting, TOR-enabled Ransomware. [Online] Available from: <http://www.welivesecurity.com/2014/06/04/simplocker/> [Accessed July 29, 2014]

[7] Roman Unuchek, 2014. The first Tor Trojan for Android. [Online] Available from: <http://securelist.com/blog/incidents/58528/the-first-tor-trojan-for-android/> [Accessed July 29, 2014]

[8] Google Inc., 2014. Dashboards. [Online] Available from: <https://developer.android.com/about/dashboards/index.html> [Accessed August 31, 2014]

[9] Alexander Kossiakoff and William N. Sweet, 2003, Systems Engineering Principles and Practices, Wiley and Sons

[10] IEEE, 1998. IEEE Recommended Practice for Software Requirements Specifications, IEEE

[11] Lovi Dua and Divya Bansal, 2014. Taxonomy: Mobile Malware Threats and Techniques, PEC University of Technology

[12] Avira vs avast vs AVG vs Panda Cloud vs Bitdefender vs MSE, review of best free anti-virus for Windows [4th Edition], 2014. [Online] Available from: <http://dottech.org/14151/windows-best-free-antivirus-antimalware-program-microsoft-security-essentials-vs-avira-vs-avast-vs-avg/> [Accessed November 16, 2014]

[13] Google Inc., 2014. System Clock. [Online] Available from: <http://developer.android.com/reference/android/os/SystemClock.html> [Accessed November 16, 2014]

[14] John Brozycki, 2008. Detecting and Preventing Anonymous Proxy Usage, SANS Institute