AMERICAN UNIVERSITY OF BEIRUT

DAGGER: DISTRIBUTED ARCHITECTURE FOR GRANULAR MITIGATION OF MOBILE BASED ATTACKS

by
KHALED ABDUL NASSER BAKHIT

A thesis
submitted in partial fulfillment of the requirements
for the degree of Master of Engineering
to the Department of Electrical and Computer Engineering
of the Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
February 2015

AMERICAN UNIVERSITY OF BEIRUT

DAGGER: DISTRIBUTED ARCHITECTURE FOR GRANULAR MITIGATION OF MOBILE BASED ATTACKS

by

KHALED ABDUL NASSER BAKHIT

Approved by:

Dr. Imad Elhajj, Associate Professor
Electrical and Computer Engineering — Advisor

Dr. Ali Chehab, Associate Professor
Electrical and Computer Engineering — Member of Committee

Dr. Ayman Kayssi, Professor
Electrical and Computer Engineering — Member of Committee

Date of thesis defense: February 5, 2015

AMERICAN UNIVERSITY OF BEIRUT

THESIS, DISSERTATION, PROJECT RELEASE FORM

Student Name: Bakhit Khaled Abdul Nasser
Last First Middle

☑ Master's Thesis ○ Master's Project ○ Doctoral Dissertation

☑ I authorize the American University of Beirut to: (a) reproduce hard or electronic copies of my thesis, dissertation, or project; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

☐ I authorize the American University of Beirut, **three years after the date of submitting my thesis, dissertation, or project,** to: (a) reproduce hard or electronic copies of it; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

Signature Feb 18, 2015
Date

This form is signed when submitting the thesis, dissertation, or project to the University Libraries

ACKNOWLEDGMENTS

I would like to thank my advisors, Prof. Imad H. Elhajj, Prof. Ali Chehab and Prof. Ayman Kayssi for their constant care and motivation throughout the time I spent at the American University of Beirut. Thank you for always finding the time to listen to my research related problems and giving me the right advice and direction in order to overcome the challenges I faced. Without your guidance, this work would not have seen the light.

I would also like to thank my family for all their continuous love, support and motivation. You are the reason I always strive for more and I hope I made you all proud.

A special thanks goes to my colleagues and friends: Salam Doumiati, Joseph Loutfi and Farah Saab. Thank you for all your support, especially during the AICCSA 2014 conference in Doha, Qatar, where a paper related to this thesis was presented. Additional gratitude goes to Lama Shaer, Riwa Mouawi, Sarah Abdallah, Mohamad Sannan and Khodor Hamandi for additional motivation and constant reminder that we're all in this together.

Last but not least, I would like to thank TELUS Corporation for funding and supporting this research work.

AN ABSTRACT OF THE THESIS OF

Khaled Abdul Nasser Bakhit for Master of Engineering
Major: Electrical and Computer Engineering

Title: DAGGER: Distributed Architecture for Granular Mitigation of Mobile Based Attacks

Over the past years, the world has witnessed sharp advancements in the field of mobile phones. What used to be known as a limited-feature mobile device for basic tasks has evolved into becoming a pocket size computer with enhanced features and computing power known as a smartphone. These devices pack various new capabilities such as access to cellular networks with various standards, access to the Internet, multi-tasking, and support of rich APIs for application development. The popularity of smartphones has been increasing as well, where global shipments reached over one billion units in 2014. This growth has been reflected on the telecom operators' networks usage. Smartphones that represented 18% of the total global handsets in 2012 generated 92 % of the total global handset traffic.

However, this advancement comes with a dark side where smartphones became the main playground for malware developers, with Android operating system being the most vulnerable platform (target of 99% of mobile malware). Several of the collected malicious applications demonstrated advanced capabilities such as data theft, root exploitation, and bot-net related operations. In fact, the interoperability of smartphones has brought Internet security issues to the cellular networks, forcing operators to consider developing detection and mitigation solutions in order to protect their resources and infrastructure.

In this thesis, we present DAGGER, a distributed architecture for collaborating mobile hosts and telecom operators for the granular mitigation of mobile-based attacks. Several security solutions are available in the market for telecom operators to detect anomalies. DAGGER extends those solutions and enables the operators to not only detect the subscriber(s) that generated anomalies, but also to identify the malicious applications behind those abnormalities, allowing the operators to terminate the malwares themselves rather than shutdown the network connection for the mobile subscriber(s). We defined the host-based component and the distributed host-network communication procedure in order to identify and terminate malicious applications causing network anomalies. We implemented a working proof-of-concept host-based component using Xposed framework on Android. Furthermore, we collected daily Android usage data from several mobile phones for several weeks in order to determine normal behavioral pattern of applications and develop Rule-based classifiers for mobile side detection, thus assisting the network. Finally, we performed simulations of various infection scenarios to test our trace-back algorithm. Our results showed malware identification rates on average between 98 % and 100%.

CONTENTS

LIST OF ILLUSTRATIONS

Figure Page

LIST OF TABLES

Table Page

CHAPTER I

INTRODUCTION

Nowadays, the world is witnessing a technological boom in two closely related multi-billion markets: mobile technologies and telecommunication networks.

In the mobile sector, what used to be known as a limited-features device for basic tasks, such as placing phone calls and sending SMS messages, has evolved into something with much more computing power and enhanced capabilities known as a smartphone. These smartphones pack many new capabilities such as access to cellular networks with various standards (GSM/CDMA, UMTS, etc.), access to the Internet with TCP/IP protocol stack support, multi-tasking for running multiple applications simultaneously, data synchronization, and rich open APIs for applications development [1]. In 2014, worldwide smartphone shipment reached one billion units in a single year, where vendors shipped a total of 1,004.2 million handsets, representing an increase by 38.4% over 2012, and more than double over 2011 [2]. Furthermore, smartphone shipment represented over 55 % of all shipped mobile phones, a strong indicator of smartphone popularity growth over feature phones. Moreover, smartphone shipment share is expected to grow up to 65.1% in 2016 [3]. Among these smartphones, Android is the dominant operating system with almost 79% market share, while Apple iOS comes in second with only 15.5% market share [4]. In fact, over 1 million new Android devices get activated on a daily basis worldwide [5]. Android's open nature, with its source code and application-programming interface (API) publically available, is the main reason behind its popularity among hardware vendors and mobile developers [5]. Furthermore, Google's Play Store reached over one million android applications

available for download in 2013 [6], with 1.5 billion active downloads per month [5]. However, the popularity and openness of Android have made it the main playground for malware developers. Around 32.8 million Android phones were infected with malware in 2012 [7]. McAfee labs reported that Android based malware achieved a 35% growth rate since 2012 by the second quarter of 2013 [8]. Furthermore, the Cisco 2014 Annual Security Report announced that 99% of all mobile malware target Android [9]. Among the collected malware samples, several of them demonstrated advanced capabilities. Geinimi Trojan was the first Android-based malware that demonstrated botnet-like capabilities whereby it was responsible for collecting information and receiving commands from a remote server that desired to control the infected phone [10]. This Trojan appeared within repackaged infected legitimate applications in Chinese unofficial app markets. Another well-known Android malware is DroidDream that managed to infect more than 50 applications distributed on the official Google Play Store market [11] [12]. The malware's functionalities were data theft, root exploits to get root access, and botnet-related operations [11]. The goal behind DroidDream was to establish a botnet by infecting 200,000 users [13].

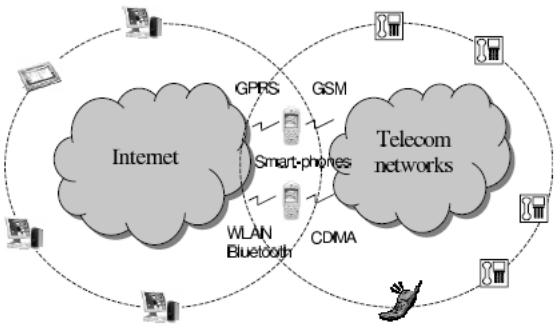

Fig. 1.1. Smartphones end-points of both Internet and telecom networks [1]

In the telecom network sector, smartphones that represented around 18% of the total global handsets in 2012 generated 92% of the total global handset traffic, making them most popular platforms for accessing network services [14]. Furthermore, due to their interoperability capability between various networks illustrated in Figure 1.1, users can stay connected to both mobile networks and the Internet at the same time with a single device [1]. However, due to the rise of malware and botnets, this interoperability has brought Internet security issues to the cellular network [1]. This raises serious concerns since cellular networks are usually planned based on a predictable usage model (voice calls and Short Message Service (SMS)) and do not take into consideration advanced attacks that could be launched through infected smartphones [1].

For this reason, it is important to understand how Android-based malware work and manage to perform malicious activities. Furthermore, it has become vital to consider how mobile-based botnets can negatively impact cellular network services through activities such as distributed denial of service attacks (DDoS). Finally, telecom operators need to consider developing detection and mitigation solutions that would protect the infrastructure from damages and service outbreaks due to infected smartphones.

CHAPTER II

LITERATURE REVIEW

2.1 Analysis of Botnets

Pieterse and Olivier identified common trends and characteristics among the most notorious android malware with traditional botnet functionalities such as delivering spam, stealing personal information, performing distributed denial of service attacks, and communicating with command and control (C&C) servers through various channels such as the Internet Relay Chat (IRC) and Peer-to-Peer (P2P) overlays [13]. Based on the analysis performed, the authors established an Android Botnet Development Model describing botnet development stages: Infection, Propagation, and Execution. In the infection phase, the bot master repackages legitimate applications, injects the malicious code, and modifies the Android Manifest xml file to add any additional permissions. In the propagation phase, the bot master submits the infected application to unofficial third party application markets. Finally, in the execution phase, once the infected application gets installed on the victim's mobile device, it starts listening to commands and/or performing malicious operations. In order to understand the incentives behind the creation of mobile malware, Felt et al. surveyed a collection of mobile malware for various platforms (Android, iOS, and Symbian) [15]. They collected 46 malware samples from January 2009 till June 2011. Their results showed that the top incentives behind malware development were stealing user information such as International Mobile Equipment Identity (IMEI) numbers and contacts list (60.8%), and interacting with premium numbers by placing phone calls and sending SMS

(56.5%). They also mentioned methods that malware use to perform their attacks such as sending sensitive data over the internet, taking advantage of root exploits that are publically available 74% of the time, and phishing. They also noted that 15 samples out of 46 (32.6%) worked as bots and behaved based on commands received from C&C servers. Similar conclusions where achieved in [16] where the authors evaluated 1,260 Android malware samples. They found out that 1,083 (86.0%) samples were repackaged versions of legitimate applications with malicious code injected. In addition, the top incentives behind those malware samples were stealing user personal information (644 or 51.1%), and sending short messages or placing calls to premium-rate numbers (571 samples or 45.3%). It is important to note that 1,172 samples (93.0%) demonstrated botnet functionalities and were able to receive remote commands from C&C servers. Furthermore, the antiviruses were only able to detect 79.6% of the malware in the best case and 20.2% in the worst case, demonstrating ineffectiveness of solutions available on the market for detecting malware. On the other hand, the authors of [15] predicted that with the fast improvements in mobile phones' processing power, battery lifetime, and bandwidth availability witnessed by the current market, a more dangerous malware attack type will become popular: launching DDoS attacks on online servers and cellular networks leading into wide service outages.

2.2 Botnet Implementation

Several works have been done on coming up with botnet communication schemes that are reliable, efficient, and hard to detect by various intrusion detection systems (IDS). Mulliner et al. in [17] introduced three types of communication mechanism for botnets on Apple's iPhone devices with C&C servers: P2P, SMS, and a

hybrid SMS-HTTP based mechanisms. The P2P based mechanism is designed using P2P tool Kademlia. Each infected phone checks the P2P network every 15 minutes for commands. In SMS based communication mechanism, commands are issued through SMS. Upon infection, the mobile phone informs the infector about its mobile number by an SMS message. The bot master collects the numbers and forms a tree in order to reduce total number of messages sent by a single node to its children nodes. In order to maintain the tree, a ping is sent periodically by the bots to the bot master. In SMS-HTTP based communication mechanism, command messages are encrypted and posted on a website. Website URL is sent via SMS to randomly selected infected devices. The bots then open HTTP connections to the provided URL and download the commands. In order to avoid being discovered, the authors took careful attention of resources consumption on infected mobile devices. The SMS message count was minimized to reduce victim's phone bill, while internet traffic cost was minimized by choosing the suitable wireless attached infrastructure (Wi-Fi or 3G). Similarly, the authors in [18] proposed a hybrid model that uses both P2P and SMS with an advanced topological structure and organization. In the proposed structure, the node elements are divided into 5 levels: bot master, bot server, region bot server, newly infected bot and collection node. Each level has its own role within the topology: the bot master has information about all the bots; the collection node is responsible of collecting information from all the nodes in the topology; the bot servers perform search and forward; the region bots act as bot servers as well as regular bots; and finally, the bots that are responsible for receiving and executing commands. The servers communicate using P2P, while all C&C messages are forwarded via SMS messages to bots in encrypted form. The advantage of using SMS is that it is available on most mobile devices. Furthermore, the

authors propagated their malware using a three step approach. First, the bot master uses an OS exploit to install the malicious app. Then, the bot master chooses a bot server. Finally, existing nodes infect new nodes and propagate the malware. In [19], the authors used a pure SMS based approach in their proof-of-concept Android-based botnet that uses optimized SMS flooding command propagation. In addition, they used an online server to construct topologies and instruct each infected bot to which neighbor node it should send SMS messages. It is important to note that the exchanged messages were encrypted using a shared secret key hidden using stenographic techniques. In addition, to remain undetectable, the authors developed a rootkit that hijacked the system call table and managed sending and receiving SMS messages. Using Edros-Renyi random graphs, the botnet was able to achieve 90% coverage on the 20,000 attacked mobile phones within 20 minutes. On the other hand, Xiang et al. proposed an advanced HTTP-based communication mechanism between bots and bot master using URL Flux protocol [20]. In this protocol, the bots use a pool of web 2.0 addresses of microblogs and a Username Generation Algorithm (UGA). To retrieve commands, a bot connects to one of the microblogging servers and searches for a valid user generated by UGA. Once a valid user is found, the botnet caches the name for later re-use and retrieves commands embedded within RSS-based feeds in compressed form to minimize network data consumption. The botnets use bot master's public key to verify the downloaded command messages. By using multiple microblogs for command publishing, the protocol solves the single point of failure problem and makes it difficult to retrace C&C commands to the original bot master. Other work included using the Session Initiation Protocol (SIP), which is used in IMS for service delivery, to conceal botnet traffic preventing detection in 4G networks [21].

2.3 DDoS Attacks on Cellular Networks

Telecom network operators used to plan their network capacity based on the assumption that traffic is highly predictable and only dumb terminals will be accessing the network for placing calls and sending SMS [1]. However, the appearance of smartphones with the ability to interoperate between telecom networks and the Internet, cellular networks have become prone to dangerous Internet-based security threats. The authors in [1] demonstrated such threats using a mathematical example. In GSM networks, base stations with n carrier frequencies can be completely exhausted by 8n coordinated botnets in the same cell initiating calls to occupy all the available time slots, thus affecting the availability of the cellular network by raising the call blocking probability above the required 0.01% limit. The call blocking probability is calculated using Erlang B formula in GSM networks where C is the number of radio channels and α is the planned call volume to support:

$$Call\ Blocking\ Probability = B(C, \alpha) = \frac{\frac{\alpha^C}{C!}}{\sum_{i=0}^{C} \frac{\alpha^i}{i!}}$$

By setting α=15.63 (typically 15-16 simultaneous users) and B < 0.01%, we can see that 32 voice channels are required. Table 2.1 demonstrates the effects of the number of infected smart phones within close proximity, placing phone calls simultaneously, on the base station's call blocking probability.

Table 2.1. Effect of infection spread on call blocking probability

Infected Smartphones Count	Call Blocking Probability
16	16.4 %
24	53.6 %
32	Out of Service

In this context, a wide-spreading smartphone botnet network can cripple the telecom infrastructure and jeopardize critical call centers. Furthermore, the authors in [22] performed a DDoS attack using overwhelming signaling requests on the Home Location Register (HLR) that is the heart of 2G/3G networks. The authors proved that depending on traffic condition and the capacity of the HLR system with the assumption that one million users are serviced by each HLR, 11,750 (1.2%) to 141,000 (14.1%) botnet infected phones can overload the HLR and render it unusable for clients. Khosroshahy et al. demonstrated a botnet DDoS attack on the 4G cellular network by instructing botnet nodes to start uploading/downloading dummy data in order to overwhelm the air interface, Long Term Evolution (LTE), thereby denying service for voice users [23]. Based on their simulation results, they determined that only 3% of infected subscribers are capable of lowering the voice quality from 4.3 to 2.8 in the Mean Opinion Score (MOS) scale, while only 6% infection rate can cause a complete service outage. Finally, Mulliner et al. demonstrated in [24] a DDoS attack on both mobile clients and the cellular network without using a malware on the mobile handsets. They showed that receiving malformed SMS messages can crash certain mobile phones. This attack remotely forces the mobile phone to reboot, disconnect, and re-authenticate with the network causing a higher load on the network core infrastructure. In fact, if the phone crashes before acknowledging the received malformed SMS, the network will be under the impression that the message did not get delivered and will keep on retransmitting it.

2.4 DDoS Attacks on Internet Servers

Mobile phones can also be used to launch DDoS attacks on online Internet servers by overwhelming the servers with requests. In [25], the authors studied the IP addresses that are exposed by cell phones on 3G networks to the Internet servers. They showed that cellular networks assign new IP addresses to mobile phones as often as every few minutes, or upon reconnection to the data network. In fact, the assigned IP addresses do not contain enough information about the locality of the devices, thus making IP-based user identification and geolocation almost impossible. Felt et al. claims that this feature makes mobile-based DDoS attacks on websites much more effective than stationary-based attacks where public IP addresses are mostly static or infrequently changing and cannot be forcibly reassigned [15]. Servers will not be able to stop mobile-based DDoS attacks by blocking and blacklisting IP addresses of anomalously behaving visitors since the source addresses are constantly changing.

2.5 Vulnerabilities of Android

Android has become the most popular mobile operating system in the smartphone market due to its open source nature. Unfortunately, the popularity and flexibility of Android have made it the main playground for malware writers. A lot of research has been done to find security vulnerabilities in Android that could be exploited by malware developers. In [26], the authors managed to place phones calls without having the required privilege. In particular, by exploiting Tcl commands, a non-privileged application managed to send 50 SMS messages to any number specified by the attacker. In addition, the authors of [27] managed to develop a malware in native Linux binary that was used to bypass Android permissions and allow the intruder to

perform any operations. Even though some of the explored problems have been solved, these works demonstrated that a non-privileged vulnerable application could cause a lot of harm on the smartphone with security vulnerabilities. However, understanding Android permissions given to requesting applications is also vital, and ignoring the importance of that can put users at risk. A series of interviews and surveys conducted with a group of Android users in [28] demonstrated that only few users understand the differences between the various permissions requested by applications. Pieterse and Olivier highlighted that certain combinations of permissions can indicate malicious behavior by the requesting applications [13]. For example, RECEIVE_SMS, READ_SMS, and INTERNET may indicate that an application is receiving commands from a C&C server. INTERNET and WRITE_EXTERNAL_STORAGE can show the possibility of the application downloading additional content such as malicious payload. READ_CONTACTS, READ_SMS and READ_PHONE_STATE can be a sign of application stealing personal information. Lastly, RECEIVE_SMS, READ_SMS, WRITE_SMS, and SEND_SMS can point to the possibility of applications sending out SMS messages. In fact, Zhou and Jiang computed in their dataset that the average number of permissions requested by malicious applications was 11 as compared to 4 for benign applications [16].

2.6 Malware Detection

A stream of work has been developed in order to build solutions that can detect and prevent malware based attacks. In this section, we separate between host-based solutions and network-based solutions. Several Android-based intrusion detection systems were suggested for detecting malicious applications. In [29], the authors

presented a behavioral-based detection framework named Andromaly. This host-based IDS monitors the system by collecting various system metrics. Afterwards, these metrics are fed into a detection unit that uses machine learning and classification algorithms to classify whether a certain given application is benign or malicious. This framework is considered as a behavioral-based IDS that analyzes applications based on their behavior. A deeper-analysis solution is proposed in [30] where tracking is performed at a very low-level. In particular, the proposed solution TaintDroid tracks how applications use the user sensitive data by using "system-wide dynamic taint tracking and analysis". In [31], the authors extended TaintDroid and proposed QuantDroid that monitors inter-process communication (IPC) between applications in order to detect privilege escalation attacks. The authors modified the android.os.Binder module and monitored the information transmission between applications. The application measures the size of the information passed and applies a threshold to detect a potential information leakage. Both, TaintDroid and QuantDroid, require deep modifications in the Android operating system making them impractical for deployment. Other researchers suggested more lightweight solutions for Android malware detection. In [32], the authors suggested a network traffic analysis scheme for detecting malware violating user privacy. By using virtualized Android devices and sample malware applications, they collected network HTTP traces and analyzed them for leaks of personal information using content matching (header flags, syntactic matches in HTTP messages). They also viewed the list of contacted C&C servers and compared them against Domain Name/IP blacklists. However, their methods are limited since blacklisting requires prior discovery of malicious domains and it is ineffective in peer-to-peer communication. Moreover, plain-text content matching techniques can be

rendered ineffective with end-to-end encryption. Another approach, called Crowdroid, involved detecting repackaged malicious applications by counting system calls performed by a certain installed application and comparing the results with the data collected from profiling the original benign application with the same name and version [33]. In fact, this platform relies on crowdsourcing for collecting the required behavioral traces of applications. Other proposals considered offloading malware detection from host-based to cloud-based detection. The rationale behind this option is that smartphones are considered as devices with limited resources, which prevents the integration with resource hungry heavyweight security monitoring solutions. Thus, it is suggested to run exact replicas of mobile devices on emulators in the cloud. Online servers are not subject to the same constraints and hence can run multiple detection techniques simultaneously. For real-time monitoring, a given mobile device continuously synchronizes with its associated emulator by passing device input and network connections to the cloud [34] [35].

However, the provided host-based solutions might not always be effective at detecting malware or DDoS attacks launched by botnets, especially the ones that only target network infrastructures and servers, without leaking user information which is the main concern of many suggested host-based IDS. For this purpose, network-based solutions were investigated. The authors in [1] recommended several detection methods for DDoS attacks. They suggested observing variations in: blocking rate at base stations and switches, data packet drop rate at GPRS or CDMA Internet Access Points, call centers load, and end-user behavior such as endless call initiations followed by abortions, connected calls without voice traffic, non-interactive bi-directional traffic, prolonged data packet transmission, and spamming. In addition, there have been several

network intrusion detection systems (NIDS) already available on the market for years. One of those NIDS systems is the open source lightweight network security solution SNORT which provides a layer of defense by monitoring network traffic for predefined suspicious activity or patterns, defined using SNORT rules [36]. A similar solution called Bro is presented in [37]. However, those solutions only detect abnormal network traffic and cannot trace back to the original application generating such traffic. For this reason, Gelenbe et al. proposed a network-based architecture NEMESYS for gathering data and detecting mobile-based malware and botnets [38] [39]. The framework consists of mobile and virtualized honeypots that "are responsible for interacting with web servers to locate websites with malicious content targeting mobile users, and for collecting related mobile threat data". Each honeypot contains a web crawler, an Android client, and a detector component. The system stores the data collected from the honeypots along with network traffic and billing and control-plane data in a data collection infrastructure. Mobile Network Operators then analyze the data using Data Mining techniques to identify and predict abnormal behaviors and attacks. Finally, results are presented using visual analytics technologies to security analysts.

DAGGER, proposed in this thesis, is designed to enhance network-based intrusion detection systems by enabling them to not only detect anomalies but also granularly identify and terminate the malicious applications behind these anomalies. To the best of our knowledge, no existing NIDS can perform such detailed trace-back operation.

CHAPTER III

DAGGER

Our proposed architecture is designed to allow telecom operators to granularly identify malicious mobile applications that are generating anomalies on the network such as signaling attacks, DDoS, or spamming. The intrusion detection system used can be any from the wide selection of proposals already available. Our aim is to extend the mitigation ability of NIDS by allowing them to identify and terminate misbehaving applications on the mobile hosts. Our architecture also enables communication between host-based and network-based intrusion detection systems in an effort to improve collaboration in malware detection and mitigation. DAGGER consists of a host-based security component distributed across mobile clients that can communicate with various network entities that are running anomaly detection software such as base stations, femtocells, or the network core. We also define a mechanism for the communication that would enable the network to trace back detected anomalies to specific offending mobile applications.

3.1 Assumptions

Following are the assumptions that the proposed architecture is based upon:

- A host-based component is deployed on all smartphones that access the telecom operator's services.
- The network runs a NIDS that can detect anomalies and identify the mobile subscriber(s) from which the anomalies are originating.

- The network NIDS collects and stores logs of events along with timestamps.
- The NIDS is capable of communicating with the host-based component.
- The NIDS is deployed at the entry points of the network (base stations, femtocells, etc.) or the network core.

3.2 Host Based Component

The host-based component is deployed on smartphones that use the telecom network services. Its functionalities consist of the following:

1- Logging network-related traffic generated by applications to enable the telecom operator to identify applications based on their network-related behavior.
2- Terminating the execution of applications flagged as malicious.
3- Mediating the communication between the host-based IDS and the network.

In order to accomplish these goals, the host-based component consists of several components illustrated in Figure 3.1.

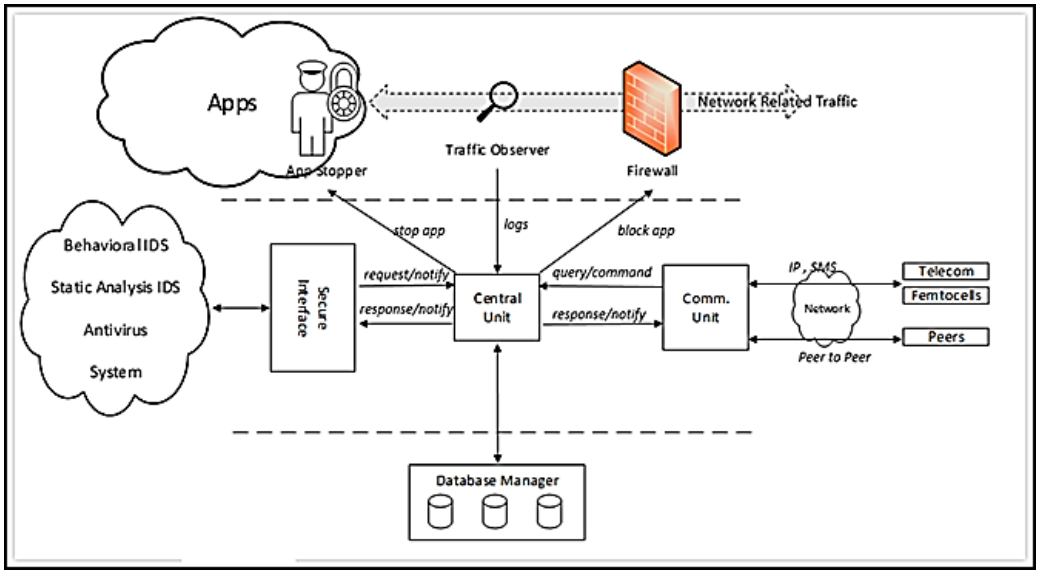

Fig. 3.1. Host-based component architecture

3.2.1 Firewall

The mobile-based firewall extends the usual capabilities of traditional firewalls due to the vast network communication capabilities available to smartphones. It is responsible for controlling incoming/outgoing traffic not only related to the Internet, but also SMS operations, signaling and other telecom-related operations. Moreover, upon the detection of a malicious application, the firewall is responsible for blocking all incoming/outgoing traffic related to the flagged application.

3.2.2 AppStopper

This component is responsible for blocking the execution of malicious applications in order to prevent them from causing any additional harm. It is triggered when a detection occurs and remains active until the malicious application is uninstalled or identification decision reversed.

3.2.3 Traffic Observer

The traffic observer is responsible for logging the events seen by the firewall. These logs are then stored in the database and later used to answer queries generated by the network. Table 3.1 illustrates the logged parameters.

Table 3.1 Event log generated by traffic observer

Property	**Value**
Event Type	recorded event type
Extra	extra information if available
Timestamp	time at which the event was observed
Application	application that created this event

For example, in case of an outgoing HTTPS request:

- event type: Opening HTTPS Request
- extra: 69.171.234.25
- timestamp: 1374169973088
- application: com.facebook.katana

3.2.4 Database Manager

This component is responsible for storing the log events seen by the Traffic Observer and answering queries sent by the network. Since smartphones have limited storage, the amount of logs stored is truncated in order to minimize the overall database size. Rather than storing every log event, the Database Manager only increments the count (number of times this event was created since first occurrence) and updates the

timestamp of a previous similar log event. Events with matching properties are considered as one.

3.2.5 Communication Unit

This component is responsible for communicating with the network and/or peers using SMS or Internet-based communication.

3.2.6 Secure Interface

This interface provides the means of communication with a trusted host-based IDS, which can notify the network of a local detection and provide its own logs as extra information about the malicious application that was flagged. The secure layer comes from an authentication procedure prior to the initialization of the communication.

3.2.7 Implementation

We implemented DAGGER on Android using the Xposed Framework [40]. For the Firewall and Traffic Observer components, an Xposed module is hooked to all connection-initiating methods. For the AppStopper component, an Xposed module is hooked to the life cycles related methods of Activity and Service classes. When an application needs to be stopped, the system's process killing command is executed and all future attempts to launch an Activity and/or Service are blocked. The user will be prompted to uninstall the flagged application. It is important to note that our prototype application requires a rooted Android smartphone.

3.3 Host Side Detection

DAGGER enables the host-based IDS to interact with the network and notify it about malware detection. Upon detection of a malware, the local IDS sends a notification to the host-based component along with its logs through the secure interface. The host-based component in turn forwards the information to the network. Based on the confidence level of such detection notification, the network can decide to either wait for similar reports from other hosts or take action immediately. The logs sent from host-based IDS can be used to determine the confidence level.

3.4 Network Side Detection

When the network side detects an anomaly, it is capable of identifying the mobile subscribers responsible for such abnormality. However, rather than terminating the network service for these subscribers, the network proceeds into communicating with host-based components in order to granularly identify the mobile applications responsible for creating anomalies and to terminate them. The communication sequence occurs according to Figure 3.2 below.

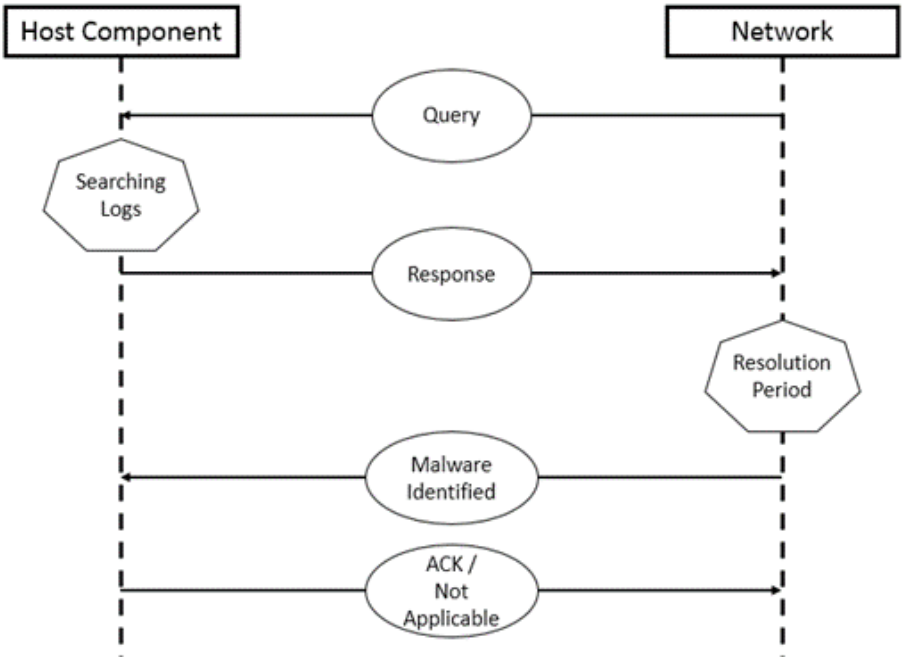

Fig. 3.2 Communication sequence between the network and host component

3.4.1 Query

The network sends a query to the mobile subscriber that was flagged by the NIDS. The query contains traffic information of a suspicious event (event type, event extra) along with the timestamp during which this event was observed.

3.4.2 Searching

The host-based component receives and forwards the query to the Database Manager which in turn performs a search among the stored logs. The search criteria attempts to find logs that match the event type, event extra, and timestamp (within a certain range of error to accommodate delays) in the query, and returns the application or list of applications found. It is important to note that the appended timestamp

information to the query increases the probability of matching only one single application rather than a list.

3.4.3 Response

The host-based component sends a response to the network containing the list of applications matching the query during the search phase.

3.4.4 Resolution Period

The telecom operator might need to send queries to multiple mobile clients to collect several responses. The process that controls the number of distributed queries is explained in Algorithms 1 and 2 in section 3.5. Afterwards, the operator proceeds into analyzing the received responses using Algorithm 3 (section 3.5) in order to identify the applications that are suspected to cause network anomalies. Those two steps occur during the resolution period.

3.4.5 Malware Identified

The network notifies the mobile subscribers demonstrating abnormal behavior about the malicious application(s) found. The host components proceed into taking targeted action against those applications.

3.4.6 ACK/Not Applicable Response

If the host component determines that the flagged malicious application(s) are installed on the mobile device, it sends an acknowledgment to the operator. Otherwise,

it sends a “not applicable” message forcing the network into continuing its investigation with that particular mobile subscriber.

3.5 Resolution Period Algorithms

In this section, we present algorithms related to the network-side detection and identification of malicious applications.

First, we provide two querying algorithms: parallel mode (Algorithm 1), and sequential mode (Algorithm 2). The decision of selecting these algorithms depends on the size of mobile subscriber pool generating anomalies, network traffic congestion, and number of simultaneous connections the network is capable of establishing in parallel.

Algorithm 1: Parallel Querying Mode

```
1. User_List= the users that created the anomaly at time T
2. Query= anomalous event details + T
3. do simultaneously for each user in User_List
4. {
5.       send the Query
6.       wait for Response and append to Responses_List
7. }
8. Malware_List= identify( Responses_List )
9. send Malware_List to every user in User_List
10. each user must respond with an Acknowledgement
```

Algorithm 2: Sequential Querying Mode

```
1. User_List= the users that created the anomaly at time T
2. Unresolved_List= users requiring further investigation
3. Query= anomalous event details + T
4. while User_List not empty {
5.       User u= remove first user from User_List
```

```
6.      if Malware_List empty
            AND all users in User_List were queried
7.              empty User_List into Unresolved_List
8.              break from while loop
9.      else if  Malware_List empty {
10.             send Query to u and wait for Response r
11.             Append r to Responses_List
12.             Malware_List= identify( Responses_List )
13.             if Malware_List still empty {
14.                     append u to end of User_List
15.                     continue with the while loop
16.             }
17.     }
18.     send Malware_List to u
19.     if u responds with an Acknowledgment
20.             continue with the while loop
21.     else if u responds with Not Applicable {
22.             if u was not queried
23.                     do steps 10, 11, 12 and 14 with u
24.             If not all users in User_List were queried
25.                     append u to end of User_List
26.             else
27.                     append u to Unresolved_List
28. }
```

Next, we present Algorithm 3, which is related to the identification step where the network deduces a list of malicious applications based on the responses received. A threshold value is used to determine whether a repeated application identification in the responses is malicious or not. This algorithm's complexity makes it capable of determining malwares like Trojans embedded in multiple applications such as DroidDream.

Algorithm 3: Malware Identification

```
1.  Responses_List= List of responses from users
2.  Malware_List= List of detected malware
3.  Hash_Table: key: App   value: Record (count, wasAlone)
4.  //count: number of times this App was seen
5.  //wasAlone: indicate if App was alone in 1 of the responses
6.  for each App_List L in Responses_List
7.  {
8.      for each app A in L
9.      {
10.         if A is not present in Hash_Table
11.         {
12.             Record r= new Record
13.             r.count= 1
14.             r.wasAlone= is size of L equals to 1
15.             add ( A, R ) to Hash_Table
16.         }
17.         else
18.         {
19.             Record r= get Record for A in Hash_Table
20.             r.count ++
21.             if r.wasAlone == false
22.                     r.wasAlone= is size of L equals to 1
23.          }
24.     }
25. }
26. for each  app A in Hash_Table
27. {
28.     Record r= get Record for A in Hash_Table
29.     if r.wasAlone OR r.count > threshold
30.         append A to Malware_List
31. }
Malware_List contains list of applications flagged as malicious
```

Consider the following list of responses received from subscribers in Table 3.2 to a certain query generated by the network due to some observed anomaly generated by a list of applications infected by a Trojan.

Table 3.2 Example response list

Subscriber	Response App List
A	1
B	1, 11
C	2
D	2, 3, 4
E	3, 5
F	4
G	4, 10
H	6, 7, 8

The malware identification algorithm will deduce the hash table shown in Table 3.3.

Table 3.3 Example hash table constructed

App ID	Count	wasAlone
1	2	Yes
2	2	Yes
3	2	No
4	2	No
5	1	No
6	1	No
7	1	No
8	1	No
10	1	No
11	1	No

According to the algorithm, and taking the threshold to be 2, applications 1, 2, 3, 4 will be flagged as malicious. It is important to note that the information received from subscriber H is not sufficient to make a decision; hence further investigation with H is required.

CHAPTER IV

ENHANCED DAGGER

In this section, we present some advanced enhancements into DAGGER that would further improve its capabilities in combating malware as well as perform host-based detection to help reduce attack impacts on network performance such as DDoS assaults.

4.1 Enhanced AppStopper

One problem with the original DAGGER proposal is “False Positive” detection where a harmless application gets flagged as malicious and terminated immediately due to, for example, some intensive task it was performing based on the end-user’s request. In order to reduce termination and deletion of benign applications performing temporary network intensive tasks (events detectable by NIDS), we modify the steps taken by AppStopper after the “Malware Identified” and “ACK” communication sequences.

Originally, the network would flag application A as malicious and send a “Malware Identified” message to related subscribers. The AppStopper component in the host-based DAGGER application installed on subscribers’ smartphones will proceed into terminating and deleting the flagged application immediately followed by an “ACK” message sent back to the network.

In order to reduce the rate of false positive, we introduce two new modifications. The first modification is the introduction of a probation state into a flagged application by the “Malware Identified” message.

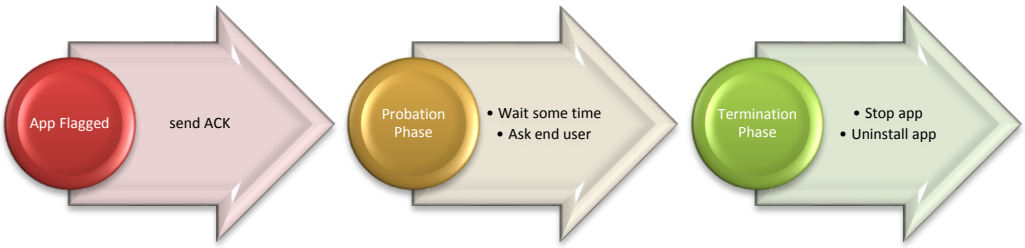

Fig. 4.1 Flagged application states

When an application gets flagged, the host-based component must send an "ACK" message to the operator to let the operator know that the source of the problem has been correctly identified. Afterwards, the host-based component places the flagged application into probation phase where it waits for a certain amount of time defined later in this thesis during which the flagged application is monitored. Once timeout occurs, if the flagged application stops behaving anomalously, the termination sequence is stopped. Otherwise, the host-based component can resort into either asking the end-user whether the application's activity is normal and within context, or check records of pervious probation phases. If the user flags the application activity as abnormal or records show multiple probation phases, the application gets stopped and terminated. If the application keeps on acting anomalously after surviving the probation phase, the network will reinitiate the trace-back process with the similar query, placing the application back into a repeated probation phase that increases the chances of termination. Additional details on the probation phase are provided in Chapter 5 where application activity monitoring experiments are presented and analyzed in order to define a low false positive probation period.

The second modification is adding an URGENT flag to the "Malware Identified" message, thus forcing the AppStopper component to skip the probation cycle and proceed immediately into stopping and terminating the flagged application. In case of critical attacks that are highly impacting the network infrastructure, the network might not be able to handle the probation wait periods and might require to terminate applications immediately. Here, we can deduce that benign apps that are abusing network resources are also considered malicious.

4.2 Rule-Based Detection

DAGGER requires the host-based component to collect and log activity data of mobile applications installed on the smartphone. However, it only uses this information to answer trace-back queries sent by the network, which are usually sent in case of a detected anomaly or attack. The detection job is entirely left for the network. In this modification, we take further advantage of the collected log data and add detection capabilities to the host-based component in order to aid the network in early attacks mitigation.

We propose adding a "Rule-Based Classifier" component into the host-based application that logs data and detects abnormalities as defined by the telecom operator's rules, such as exceeding the number of HTTP requests allowed per minute.

For this purpose, we need to modify the log structure stored by the database. Rather than storing just the last observed time stamp of an event, we add an observation window period where we count the number of such event occurring within a certain timeframe defined by *Current Time – Reference Time*. Table 4.1 illustrates the structure of the new log, while Algorithm 4 describes the event log storage procedure.

Table 4.1 Enhanced log stored by database manager

Property	Value
Application	application that created this event
Event Type	recorded event type
Extra	extra information if available
Last_Timestamp	last timestamp at which the event was observed
Total_Count	Number of times this event was observed since installation
Reference Time	Reference Time. Reset to Last_Timestamp every window period
Count	Number of times this event was observed since reference time

Algorithm 4: Enhanced Log Storage Procedure

```
1.  Input: Application A generated event E(Type, Extra) at time T= Current_Time
2.  Application= A
3.  Event_Type= E.Type
4.  Extra= E.Extra
5.   Last_Timestamp= T
6.  if no previous record found for (A, E)
7.  {
8.          Total_Count= Count= 1
9.          Reference_Time= T
10. }
11. else if T – Reference_Time > Timeout
12. {
13.         Total_Count=  old.Total_Count + 1
14.         Count= 1
15.         Reference_Time= T
16. }
17. else
18. {
19.         Total_Count=  old.Total_Count + 1
20.         Count= old.Count + 1
```

21. }
22. Store related variables in the database for (A,E)

We can use rules to detect two kinds of abnormalities: distribution-based and frequency-based. Distribution-based abnormalities occur when an application is generating an abnormal number of distinct events, such as connecting to too many distinct web servers, while frequency based abnormalities occur when an application is regenerating the same event an abnormal number of times within a certain period, such as opening too frequent connections to the same server within a certain time limit. See Figure 4.2 which shows a sample template of a rule set. If all the conditions of any given rule are matched, the responsible application gets classified as malicious. Otherwise, it will be classified as normal and no alerts will be raised.

```
Rule 1:
    Event_Type= Opening HTTP Request
    Count( distinct Event_Extra ) > limit
    Classify as Malicious
Rule 2:
    Event_Type= Opening HTTP Request
    Frequency( Event_Extra ) > average requests/Window period
    Classify as Malicious
Classify as Normal
```

Fig. 4.2 Rule set template

Rule sets can be defined and modified by the operators, and can be designed for a specific application, or be generic, applicable to all applications present at the end device. In Chapter 5, we present a set of experimental procedures where we collected

network activity data from several smartphones for couple of weeks and analyzed them in order to give detailed definition of rules and evaluate their performance.

These rules can then be applied against the log database to detect malicious applications at an early stage, as well as protect the host-based application against database overloading attacks where malicious applications generate too many distinct events. However, we still leave the more complex resource demanding analysis and detection to the NIDS deployed at the network side.

CHAPTER V

EXPERIMENTAL RESULTS AND ANALYSIS

In this chapter, we present various experiments performed in order to study the feasibility, detection efficiency and mitigation capability of our proposed architecture.

5.1 Feasibility

One of the essential components of our proposed architecture is the host-based component that must be deployed on the subscribers' smartphones. However, one of the main constraints of mobile phones is the limited available storage space. For this reason, it is undesirable for the logging functionality of the host-based component's database manager to leave a heavy footprint on the device's limited storage space.

In order to get a better insight of the projected database size, we deployed a monitoring Android application on 6 smartphones of regular users that ran for 39 days, to determine the average log size per application. We looked at non-system and non-browser applications that required the permission *android.permission.INTERNET* that grants access to the Internet. According to [16], the Internet permission is the most requested permission among benign and malicious applications. Our dataset consists of 134 applications and the results are shown in Figure 5.1.

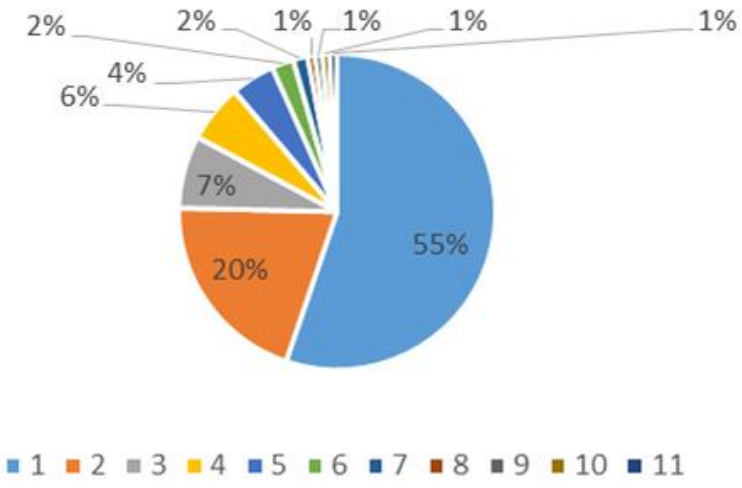

Fig. 5.1 Number of remote servers connected to per application

The average number of remote servers communicated with per application is 2.33 per device, indicating that our database log size on the mobile device will not be an issue. Other network related permissions such as sending SMS or changing the network state are not as popular and applications requesting them will not have a heavy footprint on the database. It is important to note that we ignored system applications in our results because we assume these applications are fully trusted. We also ignored browser and feeds applications because their network traffic depends on the end-users' behavior. Our data also showed that 55.22% of the total applications communicated with servers through port 80 (HTTP), while port 443 (HTTPS) coming in second with 47.01%. Of course, these numbers will certainly change with time as applications evolve and their backend servers scale and add additional services; however phone storage space will also grow and our proposal will remain feasible on the client side.

5.2 Detection Efficiency

The detection efficiency of DAGGER depends on the querying communication method chosen by the network, either Parallel or Sequential mode. In order to compare the advantage of one querying approach versus another, we developed a simulator to monitor the resolution phase. The simulator takes as parameters:

1. Number of subscribers.
2. Number of malicious Trojan infected applications generating identical events.
3. Probability of more than one malicious application within a single subscriber device.
4. Response error probability.
5. Maximum number of benign applications allowed to be flagged in case of an error.

When the number of malicious applications available is greater than one, they are distributed among the simulated subscribers using Zipf law with skew of 0.85 as shown in Figure 5.2. This is done to reflect the fact that in reality applications vary in popularity. Our experimental setup is presented in Table 5.1.

We ran the simulations 100 times for each querying mode. Table 5.2 provides the averages of obtained results.

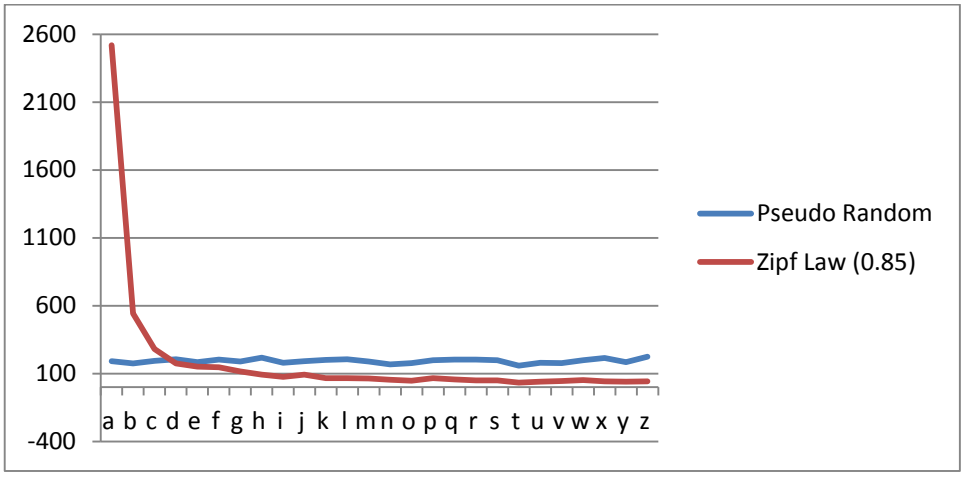

Fig. 5.2 5000 Alphabetic letters distribution using pseudo random vs Zipf law

Table 5.1 Experimental setup for querying mode comparison

Parameter	Value
Subscriber count	1000
Malicious applications	49
Probability of more than 1 malware in a single device	0.15
Response error probability	0.05
Maximum benign applications allowed to flag in error	2

Table 5.2 Experimental results of querying mode comparison

Parameter	Parallel	Sequential
Number of queries	1000	52
Number of responses	1000	52
Number of malware identification messages	1000	1140
Number of ACK/Not Applicable replies	1000	1140
Total number of exchanged messages	4000	2384
Detection rate	100 %	98 %
Misclassification	0 %	0 %

We also designed a best-case scenario simulation where there's only one single malicious application in the wild with 0 % error probability. Sequential mode had an advantage of decreasing the total number of exchanged messages from 4000 to 2002,

thus reducing the overall generated traffic overhead. However, both modes demonstrated 100% detection rate and 0% misclassification rate.

On the other hand, we designed a worst-case scenario simulation with each subscriber having one unique malicious application. The Sequential mode underperformed the Parallel mode by generating 1998 more messages. Nonetheless, both modes also revealed 100% detection rate and 0 % misclassification rate.

We can conclude that Parallel mode is fast but suffers from excessive traffic generation and receiving vast amount of redundant information. On the other hand, the Sequential mode is slower but can reduce traffic overhead and is suitable for mitigating wide DDoS attacks against the network as demonstrated in [22] and [23].

5.3 Mitigation Capabilities

After verifying the effectiveness of the proposed querying algorithms, we proceed into real world testing in order to verify the mitigation capability of our proposed architecture.

For this purpose, we implemented a simple content server that listens for incoming requests and serves the requested content. The content server is protected by a simple DAGGER enabled NIDS that registers incoming connections and detects anomalies. The anomaly detection algorithm runs every minute to check for the number of requests generated by each client within the last minute. If the number exceeds the set threshold, then the NIDS deduces that the corresponding client is behaving maliciously. After determining the list of misbehaving clients, the NIDS proceeds into initiating the resolution phase of DAGGER in order to detect and to terminate the responsible mobile applications (see Algorithm 5).

Algorithm 5: Implemented NIDS Anomaly Detection Algorithm

```
1. User_List= the users that connected to the content server within the last minute
2. Suspicious_List= users that are acting anomalously. Initially empty.
3. T: current timestamp.
4. for each User u in User_List
5.         if Number of requests made by u exceed set threshold
6.                 append u to Suspicious_List
7. if Suspicious_List is not empty
8. {
9.         Query q: Find apps that opened socket connections to the content server
10.                with given timestamp = T
11.        apply DAGGER resolution and mitigation phase with Query q
12.        clear Suspicious_List
13. }
```

We have setup 8 Android emulators with the DAGGER component installed on each one. We deployed on 4 of the emulators a simple application that generated on average 7.5 requests per minute to the content server.

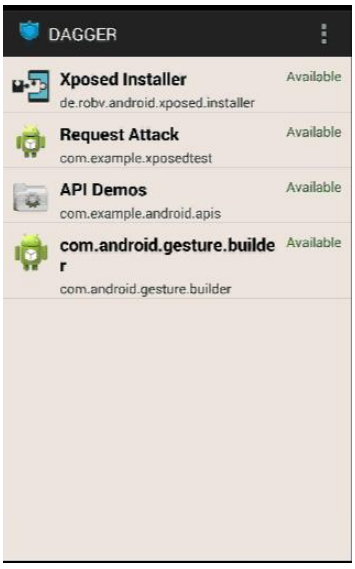

Fig. 5.3 DAGGER application running

Furthermore, we installed a self-implemented malicious application on the remaining 4 emulators that would launch a DDoS attack on the content server by simply creating too many HTTP requests.

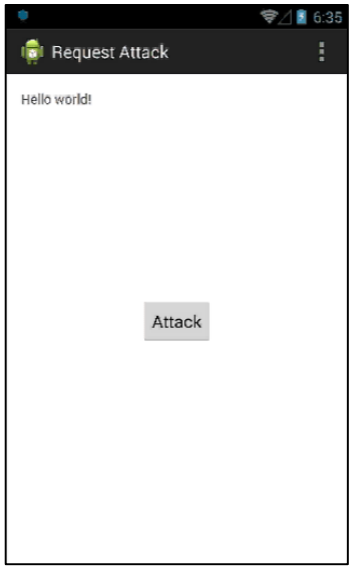

Fig. 5.4 Self-implemented malicious application

The overall architecture of our experiment setup is illustrated in Figure 5.5.

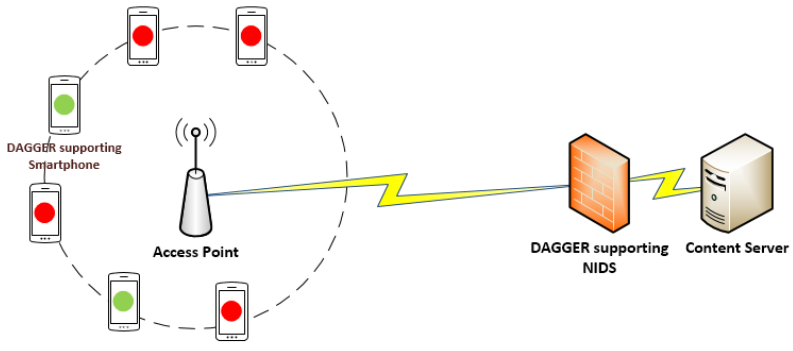

Fig. 5.5 Exeperimental setup architecture

We observed the variation of the number of requests per minute and the total number of connections received by the content server with time. The plot of the variations is presented in Figures 5.6 and 5.7, respectively.

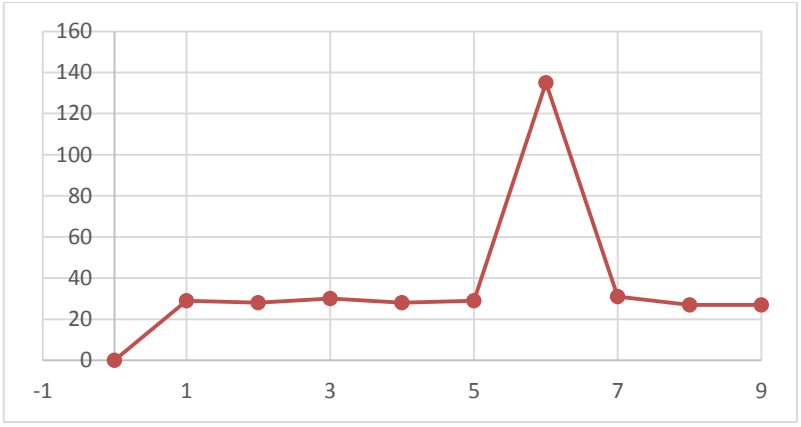

Fig. 5.6 Number of requests received per minute

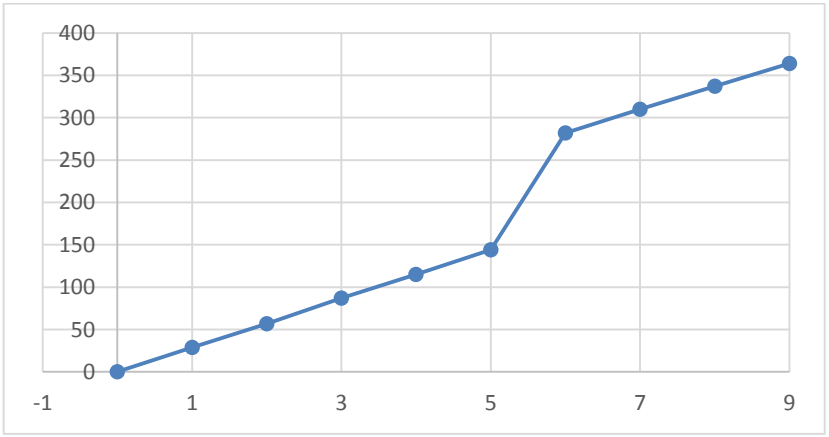

Fig. 5.7 Variation of total number of requests with time

We observe that the variations were at a stable rate between T_0 and T_5. At T_5, the malicious applications running on 4 Android emulators launched their DDoS attack on the content server, resulting in abnormal incoming traffic rate at the content server.

The attack persists till T_6, after which it gets detected by the NIDS and terminated using DAGGER. Figure 5.8 illustrates the output that was generated by the security components of the server. It is important to note that the resolution phase used the Sequential mode algorithm and we achieved a 100% detection and mitigation rate.

```
NIDS detected 4 anomalies!

Query: FIND Apps WHERE event_type: EVENT_OUTGOING_SOCKET AND event_extra:
192.168.56.1;1235 AND time: 1406456085242

malware list:[ com.example.xposedtest ]

detection & mitigation duration: 4582 ms
# of queries: 1
# of replies: 1
# of commands: 4
# of command replies: 4
total msgs: 10

Success for: 4 out of 4
```

Fig. 5.8 NIDS output

The host-based DAGGER components on the emulators were able to identify the malicious applications based on the query sent by the NIDS. Afterwards, they were able to stop and uninstall the applications using the AppStopper component once instructed by the NIDS.

Fig. 5.9 DAGGER in action

After T_6, we notice the traffic load returning to normal at the content server.

5.4 Rule-based Classifiers

In this section, we present and define parameters for Rule-based classifiers that can be deployed on the enhanced DAGGER host-component and perform detection of frequent connections attack ahead of the NIDS. In order to define proper parameters, we analyzed network activity data collected from 16 different users using an Xposed based Android application for an average of nearly 30 days. There was a total of 255 applications detected with only 50 being system apps. Furthermore, out of those 255 applications, it was observed in the logs that 1 system and 145 non-system applications connected to the Internet. We will only analyze the 145 applications in our study.

Table 5.3 Collected data states

User	Total Apps	Non-Sys Apps With Internet Logs	Estimated Data Collection Duration (Based on logs timestamps)
1	151	51	8 days, 8.5 hrs.
2	175	64	6 days, 8.5 hrs.
3	246	66	12 hrs.
4	166	68	120 days, 9 hrs.
5	168	60	30 days, 15.75 hrs.
6	186	58	87 days, 8 hrs.
7	151	43	13 days, 23.5 hrs.
8	140	43	1 day
9	155	48	42 days, 2.25 hrs.
10	146	45	26 days, 11.25 hrs.
11	142	46	14 days, 3 hrs.
12	149	47	34 days, 17 hrs.
13	149	47	9 days, 13.5 hrs.
14	145	43	41 days, 3.5 hrs.
15	167	46	17.5 hrs.
16	149	55	45 days, 15.5 hrs.
Avg	**161.56**	**51.875**	**30 days, 4.5 hrs.**

We are interested in the apps' internet activity collected logs that were in the following form: [timestamp, app UID, source IP, destination IP]. We aggregated those logs and counted number of connections every app UID made to a single destination IP within a one minute timeframe. So, our final logs were of the following form: [log ID, timestamp, app UID, destination IP, count] where each (app UID, destination IP) pairs were separated by at least one minute time window. Count is the number of connections made by the application to the destination IP within the given minute.

Table 5.4 Internet activity logs per user

User	Logs	User	Logs	User	Logs	User	Logs
1	54,260	5	20,976	9	64,311	13	5083
2	46,499	6	48,961	10	4972	14	130,509
3	7,033	7	79,876	11	46,897	15	354
4	203,418	8	20,425	12	13,498	16	20,976
Average				**51,003**			

After pre-processing and aggregating the data, we proceeded into separating the users into training and testing groups. The training group was used for defining and determining the rules' parameters while the testing group was used for verifying. In our case, we are assuming that the logs are generated by benign applications and our goal is to lower false positive detection by the rule-based classifiers.

Before presenting any results, we need to discuss the generation of the rules. As mentioned earlier, the DAGGER host-based component supports two set of rules: a general rule applicable for all applications, and a targeted rule applicable only on a selected application. In the following set of experiments, we will use rules to monitor number of connections generated per app per IP within a one minute timeframe. In addition, we will test using a single rule threshold against a set of several thresholds depending on the hour of the day, since application usage varies within 24 hours. Figure 5.10 displays the average number of Internet connections made per minute by the most popular Android applications (Facebook, Facebook Chat, Instagram, Viber and Whatsapp) throughout the day periods (discrete points). We define each period within the following hours of the day: *Morning* between 6 and 11, *Midday* between 12 and 17, *Evening* between 18 and 23, and *NIGHT* between 0 and 6. We can observe that there is no significant variation in the usage pattern, indicating that it might not necessarily to have multiple time dependent thresholds within the rules.

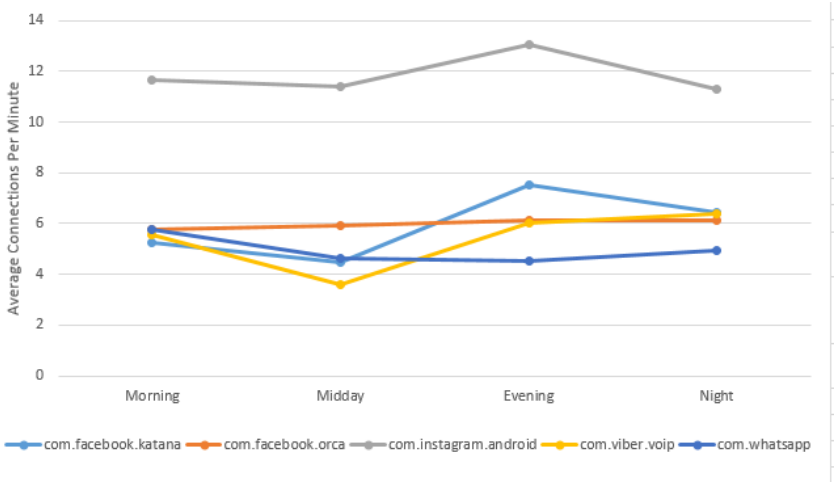

Fig. 5.10 Average connections per minute of popular apps by day periods

Accordingly, since our log data takes the form off [log ID, timestamp, app UID, destination IP, count], our definition of the threshold is as follows:

$$threshold = average\ (\ count\ ) + standard\ deviation\ (\ count\ )$$

In our experimental results presented later in this section, appending the standard deviation to the threshold significantly decreases the false positive rate. Each Rule can come with up to 5 thresholds: 4 for each time period of the day, and one general threshold independent of the time (*All-Day*). The unit of a threshold is defined as requests per minute. Selection of threshold depends on the classification configurations. In case the classifier is configured to use specific targeted rules and its datastore does not contain a rule for a certain application, it will use the general rule by default.

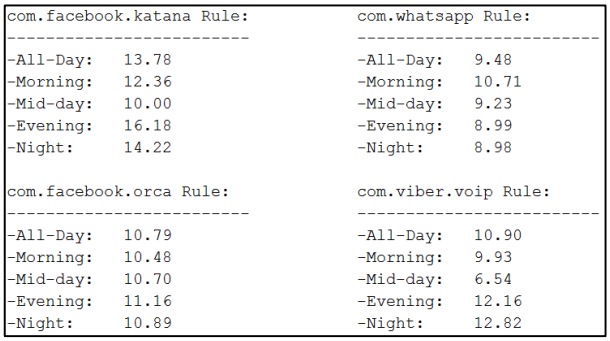

Fig. 5.11 Sample targeted rules trained on the entire dataset

In what follows, we performed Rule-based classification on the dataset at hand. We used 11 users for training the rules while the remaining 5 users for testing. We performed 10 fold verification and computed the averages for each set of tests. Furthermore, we introduced the notion of probation period which is the amount of time the host-based application waits before taking into consideration the flag raised by a rule classifier towards any given application, since a classifier might flag a benign application being active as malicious. An application enters probation period once it gets flagged by a rule-based classifier by illustrating number of connections per minute to be greater than the threshold. If the application keeps on triggering the classifier after the wait period, it gets classified as malicious and further action will be taken such as asking the end-user or directly stopping the application. Otherwise, the flag gets cleared and no action will be taken against the app.

We will be studying effects of various classification configurations and false positive rate as a function of probation waiting period.

In the first study, we will compare using averages as thresholds versus averages with standard deviations. The results of this comparison are plotted in figures

5.12 and 5.13 respectively. We can observe that appending the standard deviation gives lower false positive rates as a function of probation period.

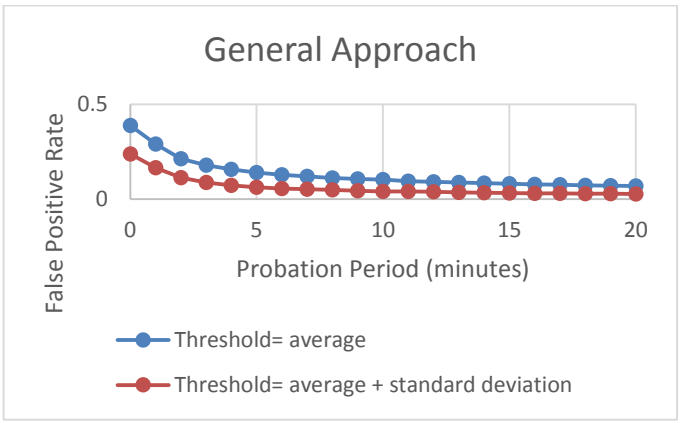

Fig. 5.12 Threshold configuration vs probation time in general approach

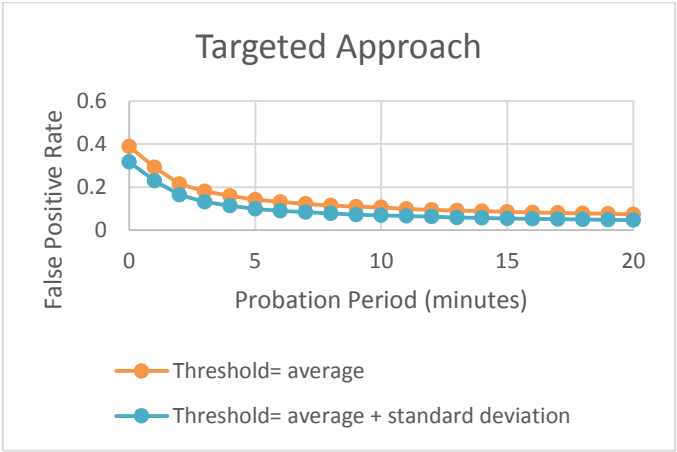

Fig. 5.13 Threshold configuration vs probation time in targeted approach

In the second study, we will compare time dependent to time independent thresholds with standard deviation appended. The results in figures 5.14 and 5.15 indicate that both approaches give approximately similar results.

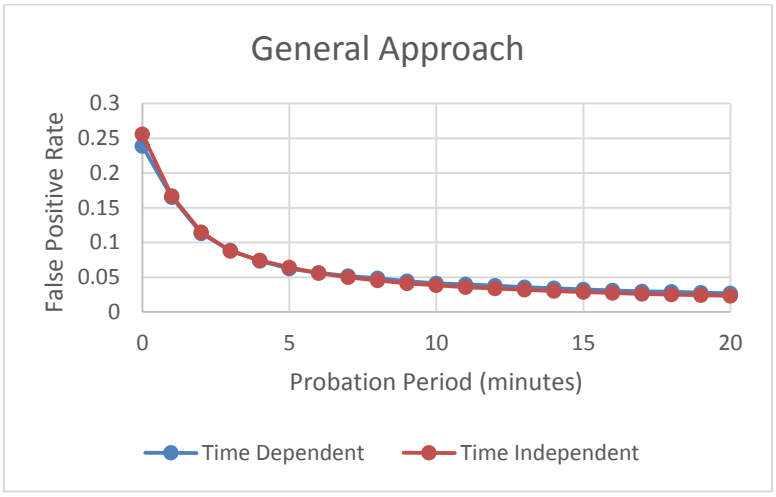

Fig. 5.14 Time dependent vs time independent thresholds in general approach

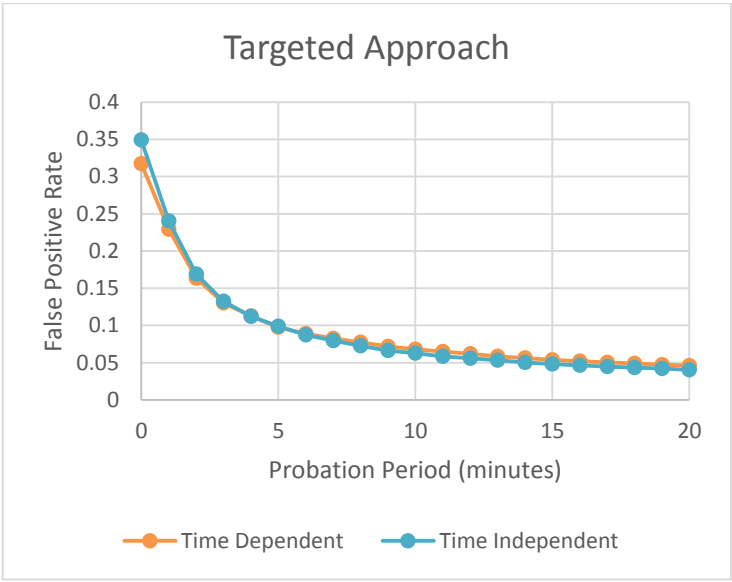

Fig. 5.15 Time dependent vs time independent thresholds in targeted approach

In the final study, we will set the threshold configuration to time independence with standard deviation appended and compare the general rules approach to targeted rules approach and attempt to propose a suitable wait time for the probation period.

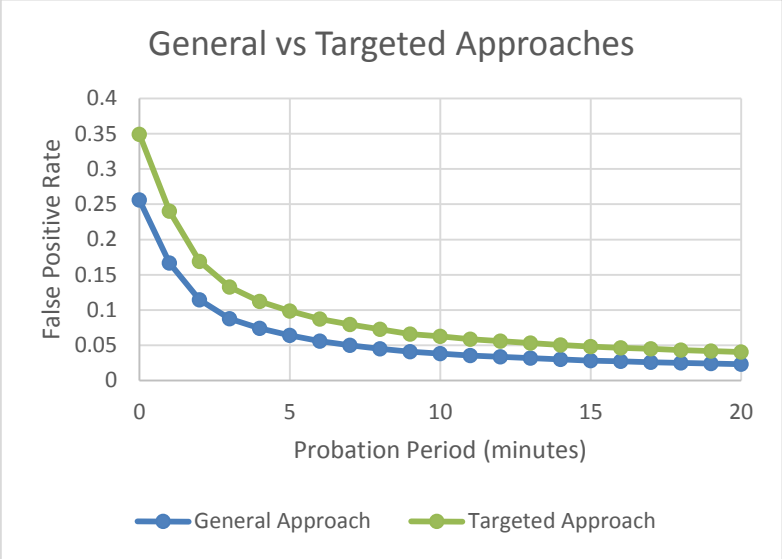

Fig. 5.16 General versus targeted approaches

According to figure 5.16, the generative approach yields better results where it starts with 25.6 % false positive rate compared to 34.9 % of the targeted approach, and reaches 2.32 % false positive rate at wait time= 20 minutes comparing to 4.05 % of the targeted approached. In addition, at time= 5 minutes, the false positive rate achieved by the generative approach is 6.4 %, a tolerable rate within such a short probation period.

```
General Rule:
-------------------------
-All-Day:    16.82
-Morning:    15.65
-Mid-day:    13.68
-Evening:    17.90
-Night:      18.35
```

Fig. 5.17 General rule resulting in lowest false positive rates trained on the entire dataset

We believe the probation period should be at least 5 minutes long, which is long enough to lower false positive flagging, and short enough to limit the negative

impacts that might occur from a possible malicious application. Further steps can be taken to lower the error rate by asking the end user about the flagged application's activity.

CHAPTER VI

CONCLUSION

In this thesis, we proposed DAGGER, a distributed architecture collaborating mobile hosts and telecom operators in granular mitigation of mobile-based attacks. DAGGER extends the available security solutions in the market for telecom operators by allowing the network to not only detect the subscriber(s) that generated anomalies, but also to granularly identify and stop the malicious applications behind such abnormalities. Furthermore, we proposed adding some enhanced capabilities into DAGGER host-based component such as probation periods to decrease false positives and rule based classification to enable earlier detection of malicious applications in order to prevent them from seriously affecting the network. DAGGER would help the telecom operator avoid shutting down services for infected mobile subscribers, protect their infrastructure from malicious applications, and become an effective player in the fight against mobile-based attacks.

We believe that DAGGER respects the privacy of mobile subscribers since it does not require the telecom operator to know the full list of installed applications on the subscribers' mobile devices. Moreover, logged information is stored on mobile devices, and only those related to malicious activities defined by precise queries are shared with the network. Certainly, sharing with the operator information the existence of flagged applications might be considered as a privacy violation however this is information that is also known to the app stores, so it is a small price to pay for the provided protection against malicious applications.

Future work would include evaluating the performance of DAGGER and its overall overhead impact on mobile subscribers as well as on the network since it involves several messages exchange, running intrusion detection systems, and collecting log data. Moreover, a detailed malware study should be conducted in order to determine their signatures while interacting with the network and to further tweak the parameters and thresholds involved in DAGGER, as well as construct effective rule sets to enhance the detection capabilities and decrease false positives as well as false negatives.

BIBLIOGRAPHY

[1] C. Guo, H. J. Wang, and W. Zhu. “Smart-phone attacks and defenses.” In ACM Workshop on Hot Topics in Networks, 2004

[2] Internation Data Corporation. (2014, Jan), Worldwide Smartphone Shipments Top One Billion Units for the First Time, According to IDC. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS24645514

[3] Canalys. (2013, Feb), Mobile device market to reach 2.6 billion units by 2016. [Online]. Available http://www.canalys.com/newsroom/mobile-device-market-reach-26-billion-units-2016

[4] Jon Fingas. (2014, Jan), Android Climbed to 79 Percent of Smartphone Market Share in 2013, but Its Growth Has Slowed. [Online]. Available: http://www.engadget.com/2014/01/29/strategy-analytics-2013-smartphone-share/

[5] Android. (2014, April). Android, the World’s Most Popular Mobile Platform. [Online]. Available: http://developer.android.com/about/index.html

[6] Dan Rowinski. (2013, July), Google Play Hits One Million Android Apps. [Online]. Available: http://readwrite.com/2013/07/24/google-play-hits-one-million-android-apps#awesm=~oAJj9rfGtoEqtW

[7] Fox Allen. (2014, Jan.), 99% of All Mobile Malware Targets Android Devices. [Online]. Available: http://www.techlicious.com/blog/99-of-all-mobile-malware-targets-android-devices/

[8] McAfee. (2013, Aug.), McAfee Labs Q2 Report Finds Mobile Threats Rebound. [Online]. Available: http://finance.yahoo.com/news/mcafee-labs-q2-report-finds-040100453.html

[9] Cisco. (2014, Jan.), Cisco Annual Security Report Documents Unprecedented Growth of Advanced Attacks and Malicious Traffic. [Online]. Available: http://newsroom.cisco.com/release/1310011/Cisco-Annual-Security-Report-Documents-Unprecedented-Growth-of-Advanced-Attacks-and-Malicious-Traffic

[10] Tim Wyatt. (2010, Dec.), Security Alert: Geinimi, Sophisticated New Android Trojan Found in Wild. [Online]. Available: https://blog.lookout.com/blog/2010/12/29/geinimi_trojan/

[11] Kevin Mahaffey. (2011, Mar.), Security Alert: DroidDream Malware Found in Official Android Market. [Online]. Available: https://blog.lookout.com/blog/2011/03/01/security-alert-malware-found-in-official-android-market-droiddream/

[12] Tony Bradley. (2011, Mar.), DroidDream Becomes Android Market Nightmare. [Online]. Available: http://www.pcworld.com/article/221247/droiddream_becomes_android_market_nightmare.html

[13] H. Pieterse, and M. Olivier. "Android botnets on the rise: Trends and characteristics." In Information Security for South Africa (ISSA), 2012, pp. 1-5. IEEE, 2012.

[14] Cisco. (2013, Feb.), Cisco visual networking index: Global mobile data traffic forecast update, 2012–2017. [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white paper c11-520862.pdf

[15] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner. "A survey of mobile malware in the wild." In Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, pp. 3-14. ACM, 2011.

[16] Y. Zhou, and X. Jiang. “Dissecting android malware: Characterization and evolution.” in the 2012 IEEE Symposium on Security and Privacy, 2012.

[17] C. Mulliner, and J. P. Seifert. “Rise of the iBots: Owning a telco network.” In the 5th IEEE International Conference on Malicious and Unwanted Software (MALWARE), 2010.

[18] G. Geng, G. Xu, M. Zhang, Y. Yang, and G. Yang. “An improved sms based heterogeneous mobile botnet model.” In the Proceeding of the IEEE International Conference on Information and Automation, 2011.

[19] J. Hua, and K. Sakurai. “A sms-based mobile botnet using flooding algorithm.” In the 5th Workshop in Information Security and Privacy (WISTP), 2011.

[20] C. Xiang, F. Binxing, Y. Lihua, L. Xiaoyi, and Z. Tianning. "Andbot: towards advanced mobile botnets." In Proceedings of the 4th USENIX conference on Large-scale exploits and emergent threats, pp. 11-11. USENIX Association, 2011.

[21] A. Berger and M. Hefeeda. “Exploiting sip for botnet communication.” In Proc. 5th IEEE Workshop Secure Network Protocols, 2009, pp. 31–36. [5]

[22] P. Traynor, M. Lin, M. Ongtang, V. Rao, T. Jaeger, P. McDaniel, and T. La Porta. “On cellular botnets: Measuring the impact of malicious devices on a cellular network core”. In CCS, 2009

[23] M. Khosroshahy, D. Qiu, M. Ali, and K. Mustafa. "Botnets in 4G cellular networks: Platforms to launch DDoS attacks against the air interface." In Mobile and Wireless Networking (MoWNeT), 2013 International Conference on Selected Topics in, pp. 30-35. IEEE, 2013.

[24] C.Mulliner, N. Golde, and J. Seifert. “Sms of death: From analyzing to attacking mobile phones on a large scale.” In USENIX Security, 2011.

[25] M. Balakrishnan, I. Mohomed, and V. Ramasubramanian. “Where's that phone? Geolocating IP addresses on 3G networks.” In IMC,2009

[26] L. Davi, A. Dmitrienko, A. Sadeghi, and M. Winandy. “Privilege escalation attacks on Android.” In Information Security, 2011.

[27] A. Schmidt, H. Schmidt, L. Batyuk, J. Clausen, S. Camtepe, S. Albayrak, and C. Yildizli. "Smartphone malware evolution revisited: Android next target?" In Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on, pp. 1-7. IEEE, 2009.

[28] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, (2012, Feb.) “Android Permissions: User Attention, Comprehension, and Behavior,” Not published. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-26.pdf

[29] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss. "“Andromaly”: a behavioral malware detection framework for android devices." Journal of Intelligent Information Systems 38, no. 1 (2012): 161-190.

[30] W. Enck, P. Gilbert, B. Chun, L. Cox, J. Jung, P. McDaniel, A. Sheth. “TaintDroid: An information-flow tracking system for realtime privacy.” In Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), Vancouver, 2010.

[31] T. Markmann, D. Gessner, and D. Westhoff. “QuantDroid: Quantitative approach towards mitigating privilege escalation on Android.” In *IEEE ICC'13*, Budapest, Hungary, 2013.

[32] D. Iland, A. Pucher, and T. Schäuble. "Detecting android malware on network level." University of California, Santa Barbara 12 (2011).

[33] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani. "Crowdroid: behavior-based malware detection system for android." In Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, pp. 15-26. ACM, 2011.

[34] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos. "Paranoid Android: versatile protection for smartphones." In Proceedings of the 26th Annual Computer Security Applications Conference, pp. 347-356. ACM, 2010.

[35] S. Zonouz, A. Houmansadr, R. Berthier, N. Borisov, and W. Sanders. "Secloud: A cloud-based comprehensive and lightweight security solution for smartphones." Computers & Security 37 (2013): 215-227.

[36] M. Roesch. "Snort: Lightweight intrusion detection for networks." In LISA, vol. 99, pp. 229-238. 1999.

[37] V. Paxson. "Bro: a system for detecting network intruders in real-time." Computer networks 31, no. 23 (1999): 2435-2463.

[38] E. Gelenbe, G. Görbil, D. Tzovaras, S. Liebergeld, D. Garcia, M. Baltatu, and G. Lyberopoulos. "NEMESYS: Enhanced network security for seamless service provisioning in the smart mobile ecosystem." In Information Sciences and Systems 2013, pp. 369-378. Springer International Publishing, 2013.

[39] O. Abdelrahman, E. Gelenbe, G. Gorbil, and B. Oklander. “Mobile network anomaly detection and mitigation: The NEMESYS approach.” In Information Sciences and Systems 2013, pp. 429-438. Springer International Publishing, 2013.

[40] Xposed Framework. (2014). [Online]. Available: http://repo.xposed.info

[41] K. Bakhit, I. El Hajj, A. Chehab, and A. Kayssi, “DAGGER: Distributed architecture for granular mitigation of mobile based attacks,” In the 11th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA’2014), 2014.