Modular anomaly detection for smartphone ad hoc communication

Jordi Cucurull, Simin Nadjm-Tehrani, Massimiliano Raciti

Department of Computer and Information Science, Linköping University
SE-581 83 Linköping, Sweden
[jordi.cucurull,simin.nadjm-tehrani,massimiliano.raciti]@liu.se

Abstract. The rapid replacement of phone handsets with smartphones calls the security aspects of these devices to be strengthened. Measures for prevention, detection, and reaction need to be explored with the peculiarities that resource-constrained devices impose. Smartphones, in addition to cellular broadband network capabilities, include WiFi interfaces that can even be deployed to set up a mobile ad-hoc network (MANET). While intrusion detection in MANETs is typically evaluated with network simulators, we argue that it is important to implement and test the solutions in real devices to evaluate their resource footprint. This paper presents a modular implementation of an anomaly detection and mitigation mechanism on top of a dissemination protocol for intermittently-connected MANETs. The overhead of the security solution is evaluated in a small testbed based on three Android-based handsets and a laptop. The study shows the feasibility of the statistics based anomaly detection regime, having low CPU usage, little added latency, and acceptable memory footprint.

Keywords: intrusion detection, resource footprint, ad hoc networking

1 Introduction

With the expected replacement of the majority of phone handsets with smartphones the need for addressing security issues on Internet-connected devices becomes more urgent. Strengthening the security of handsets, specially on open platforms on which the owner is allowed to make unrestricted downloads and create potential threats to the platform or applications is a major concern of the research community. Measures to enhance security through both prevention and detection need to be explored and the peculiarities of the resource-constrained handsets compared to earlier platforms is an exciting field of research. In this paper we explore the impact of one such security mechanism in terms of the resource claims on a modern smartphone platform.

Smartphones add the possibility of WiFi-based Internet connections to the cellular communication. However, the phones also enable the peer-to-peer mode of communication that has been subject of studies in the mobile ad-hoc networks (MANET) research for over a decade. With the increased connectivity

provided by the infrastructure-based technologies there are no major deployments of ad-hoc networks in every day scenarios. However, experience shows that when disaster strikes the existing infrastructures are severely overloaded, or rendered useless due to damages. Thus, message dissemination in disaster area networks using a phone-to-phone mode of communication is a potential means for establishing situational awareness. Our earlier work, among others, has been focused on studies of energy and bandwidth constrained communication using specially devised protocols for such scenarios [1, 2].

While imposing policies and methods for preventing threats from malware and adversary actions on modern smartphones is a subject attracting a lot of attention [3–8], to our knowledge there is little earlier work that addresses intrusion detection with a focus on the resource footprint. There is, of course, a large body of research on intrusion detection techniques for MANET [9], including our own earlier evaluation of a distributed statistical anomaly detection system [10]. However, these techniques are commonly evaluated in simulation platforms due to the difficulty of performing large scale evaluations on physical testbeds. This can act as a proof of concept for a protocol or a proposed defence mechanism, but will not be able to answer questions on the resource claims.

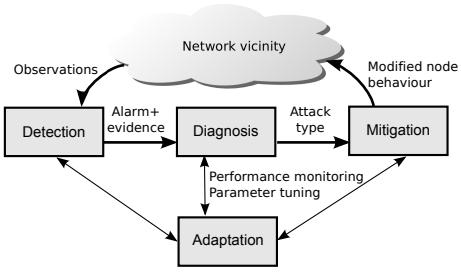

Fig. 1: General Survivability framework

Some approaches based on intrusion detection combine misuse and anomaly detection and one or more techniques for mitigation. An approach we have adopted, called General Survivability Framework (GSF), combines four modules that can be independently developed for various communication protocols (see Figure 1). The detection module is an anomaly detector. It detects deviations from normality by observing the traffic pattern in and out of the observed node. The diagnosis module, reminiscent of a misuse detector, matches the observed anomalous patterns with known attack patterns in order to aid a more direct and focused mitigation. The mitigation module has an action to curve the effects of each known attack, to be performed in the node that detects that attack. It also has a generic mitigation for unknown attacks, e.g. acting in a more careful mode with respect to peer communication. Finally, the adaption module is intended to adjust the GSF by inducing changes in the other algorithms to adapt to changes in the operating conditions, including the handset's monitored

features. In this paper we provide a modular design and partial implementation (detection and mitigation boxes) of the GSF on top of an implementation of an opportunistic dissemination protocol over the Android smartphone platform.

The design and implementation has been carefully devised to care for modularity. This allows convenient replacements of the detection-diagnosis-mitigation modules as well as the ad-hoc communication protocol. This should provide a basis for evaluating other detection engines and separate the effects of the GSF from the underlying protocol that provides the means of communication. The design has been realised on two modern Android smart phone platforms with the goal of evaluating and isolating the added overhead imposed by the GSF.

The contributions of the paper are as follows (1) We have implemented Random Walk Gossip (RWG) [1], a manycast algorithm for resource-efficient dissemination in disaster area (infrastructure-less) networks on top of an Android platform. (2) We have implemented an instance of the modular GSF, a statistical-based anomaly detection and a mitigation module, thus enabling the study of its performance overheads on a physical device, which extends the earlier NS3-based simulations [10] of the proposed technique. (3) We perform experiments on a small testbed with 3 handsets and a laptop which indicate that the resource overhead of anomaly detection and mitigation, in terms of CPU, memory and latency, is quite low when the load in the network is within a measured operating range.

The paper is organised in the following sections: Section 2 provides the background on the existing GSF and RWG, Section 3 explains the design and implementation for Android, Section 4 details the evaluation done with the real smartphones, Section 5 briefly describes the related work, and Section 6 concludes the paper.

2 Background

2.1 General Survivability Framework

GSF is a generic framework in which arbitrary detection, diagnosis, mitigation, and adaption elements can be introduced. In this paper we base our implementation and evaluation on certain instances (detection and mitigation) of the components named in Section 1. The detection component is based on the statistical anomaly detection algorithm proposed in Cucurull *et al.* [10]. A statistical approach is chosen due to its small footprint, ideal for the resource-constrained nature of smartphones. The GSF periodically captures the network state, an observation, which is represented by a vector of numerical values called features. An observation is composed of network statistics, such as packet rates, estimation of number of neighbour nodes, and so on. The detector algorithm calculates the Euclidean distance between an observation and a normality model local to the smartphone. The normality model of the system is generated in a two-step process during a training period. First, a vector which is the average of many observations is created. Second, a number of observations is taken to calculate

the threshold, which is the mean of the distances plus three times their standard deviation. An observation is categorised as anomalous if the distance is above a threshold. When an anomaly is detected a general mitigation, or specific if the attack is identified, is engaged. Earlier work [10] and an extension of it for multiple mitigations and known attack classifications has provided a proof of concept for feasibility of this approach in a distributed ad-hoc communication setting. This paper will elaborate on realisation of some instances in a real handset deployment.

2.2 Random Walk Gossip

RWG [1], the routing protocol chosen in the implementation, is a message dissemination protocol for intermittently connected ad-hoc networks. The protocol copes with intermittent connectivity, scarcity of bandwidth, and energy, as well as unknown and unpredictable network topologies with partitions. RWG is a manycast protocol, which means that a message is intended to reach a given number k of nodes, with no knowledge of the node IDs in the topology. RWG is based on a store-and-forward mechanism, i.e. each node keeps the messages to forward in a local buffer until it realises they are k-delivered or they expire. The protocol follows a three-way packet exchange (see Fig. 2).

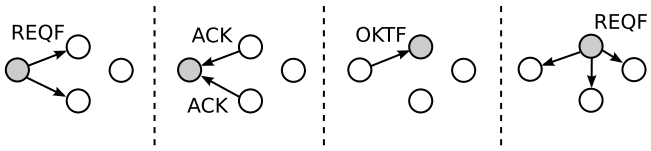

Fig. 2: Random Walk Gossip

First, a Request to Forward (REQF), that includes the message payload, is sent by the current custodian of the message (grey nodes in the picture). The neighbouring nodes hear the REQF reply with an acknowledgement packet (ACK). The custodian randomly chooses one of these nodes and sends an OK to Forward (OKTF) indicating the next custodian. The other nodes retain the message without actively disseminating it. Partitions can be overcome by the movement of nodes. Thus, new uninformed nodes will be informed by some node that keeps the message as *inactive* and restarts to disseminate. In order to keep track of which nodes have seen a given message, each packet header contains a bit vector, *informed*. Each position of the vector maps to one encountered node address, using a hash function. This is also used to indicate whether a current encountered node has seen the message earlier. The vector enables the protocol to know when a message is k-delivered (when k bits are set). Finally, when a node realises that a message is k-delivered it sends a Be Silent (BS) packet to its vicinity.

3 Layered Communication and Anomaly Detection

This section proposes a layered modular design for the implementation of the described GSF and RWG services in Android smartphones with an emphasis on component independence and interchangeability.

3.1 Overall architecture

The communication and anomaly detection services described in this article are offered over arbitrary point-to-point ad-hoc network connections and for any set of applications. An organisation based on a stack of loosely coupled services, such as the well-known IP network stack, is therefore the most appropriate. The service stack is composed of three layers, each one implementing one possible service. Figure 3 details the layers and each implemented service.

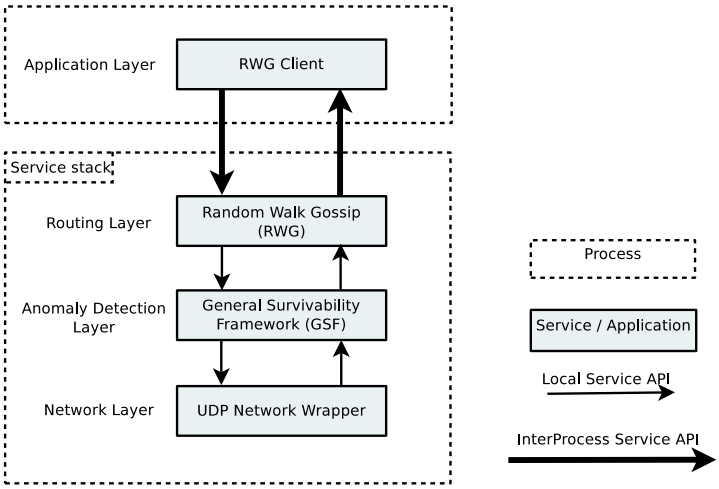

Fig. 3: Layered communication and anomaly detection structure

The network, routing, and anomaly detection layers compose a block that provides the main communication and anomaly detection service. The layers interact among themselves with a well-defined and generic inter-layer service API, which allows to easily exchange, aggregate, or bypass them. For example, the anomaly detection layer is optional and can be bypassed. The application layer comprises the applications that use the service. A specific API to interact with the applications is provided by the top-most layer of the service stack.

In Android we have implemented each of the lower three layers as locally bound services running all of them in a single process. At the same time, the topmost layer of the stack offers the main service via inter-process communication open to the applications, which run as independent processes.

The inter-layer and inter-process service APIs support send, receive, and other control operations. In the inter-layer service API the receive operation is implemented with an event listener, provided by the upper layer, that is invoked when a packet is received. The inter-process API provides the same operations, but they are invoked by inter-process messages. The network packets are exchanged among the layers as raw byte arrays, allowing the abstraction of the APIs from the specific type of packet and communication protocol.

3.2 Network Layer

The network layer provides an abstraction of the network over which the routing protocol will actually run. It includes basic services to initialise the network interface, set up an ad-hoc network, and send and receive packets. As defined by the inter-layer service API there is an operation to send packets and a listener, provided by the upper layer, for notifying their reception. A dedicated thread calls the listener each time a packet is received.

In the present implementation the packets are encapsulated within UDP datagrams, which are present in virtually all current network stacks. Furthermore, this guarantees compatibility with an earlier implementation of RWG for the Symbian platform [2].

3.3 Routing Layer

The routing layer provides the service to route the messages sent by the application layer. This layer also provides the inter-process service API to interact with the applications. This paper describes the implementation of manycast routing using RWG as described in Section 2. However, other protocols can also be implemented since the inter-layer interface is generic enough to support them.

The realisation of the protocol is based on a former implementation for Symbian OS [2]. Due to the extensive use of timers, the architecture follows an event-oriented approach implemented with a task scheduler. The architecture, depicted on Figure 4, includes the following components:

- **Data Storage**: data buffer that stores control information and payload of the messages sent and received to and from the network.
- **Task Storage**: data buffer that stores tasks pending to execute. The tasks may include sending specific types of packets or deletion of expired messages.
- **Task Dispatcher**: thread that executes the tasks present in the task storage at the scheduled time.
- **Wake One Packet**: thread that regularly sends a packet when there is no network activity, to discover neighbours.
- **Application Handler**: method that initiates the transmission of a message sent from the application layer. It creates an entry to the data storage and the required new task to execute.
- **Packet Receiver**: method that processes a packet received from the underlying layer, i.e. network or anomaly detection layers. It creates or updates entries to the data storage and new tasks to execute.

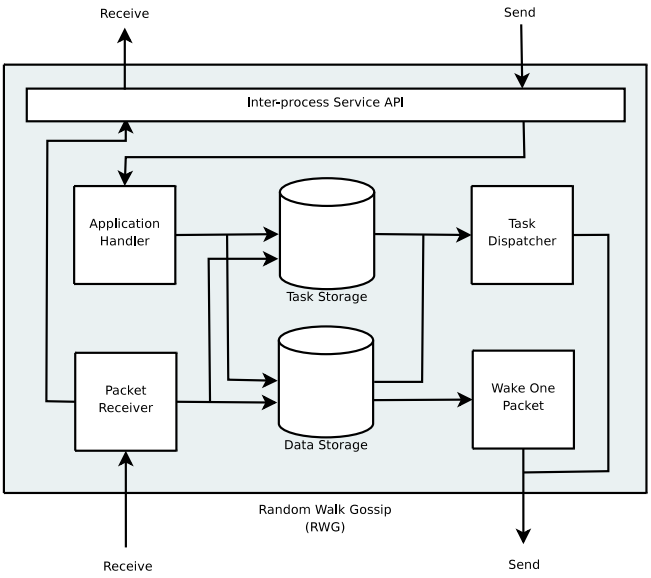

Fig. 4: Random Walk Gossip architecture

The packets sent and received to and from the network are encapsulated in objects of a class called *RWGPacket*. This class provides access to the packet headers and payload. Since the service APIs only support arrays of bytes, the object also provides methods to marshall and unmarshall the packets.

3.4 Anomaly Detection Layer

The anomaly detection layer provides a service to detect anomalies at network routing level. It is placed between the routing layer and the network layer to intercept the packets sent and received. Thus, it is able to detect anomalies and apply a response without any modification to the other layers' code. The architecture of the service, depicted on Figure 5, includes the following components:

- **IDS Engine**: main thread that governs (and periodically runs) the steps of the intrusion detection loop, i.e. data collection, anomaly detection, diagnosis, and mitigation.
- **Data Source**: component that processes each packet received or sent to generate statistics for further analysis. It returns the statistics generated, as a vector of doubles, when they are requested.
- **Anomaly Detector**: implements the detection box of GSF. This component periodically analyses the statistics created by the data source to detect anomalies. It returns alerts if an anomaly is detected.
- **Diagnoser**: implements the diagnosis box of GSF. This component diagnoses a specific attack when a detected anomaly matches a known attack. It returns a code that indicates the diagnosed attack.

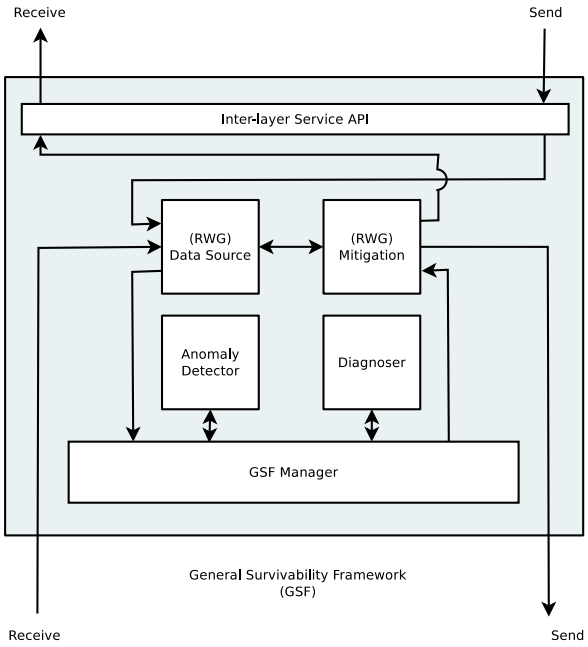

Fig. 5: General Security Framework architecture

– **Mitigation**: implements the mitigation box of GSF. This component applies mitigation measures, such as modifying or rejecting the flow of packets sent and received, when an anomaly is detected.

The service requires the *RWGPacket* class of the routing protocol to read and manipulate the packets received. The Data Source and Mitigation components are the only ones dependant on this class and the routing protocol present in the routing layer. Hence the replacement of the routing protocol only requires the modification of these two components.

3.5 Application Layer

The application layer comprise all the applications that use the main communication and anomaly detection service. It communicates with the service stack through the inter-process service API. Many applications connected at the same time to the service and running on different processes are supported.

3.6 Implementation on Android

This section discusses the implementation details of the communication and anomaly detection service. It describes the decisions taken during the implementation, and the configuration of the system. Java has been the language

used for most of the project, with the exception of a few tools used for configuring the network. Since the service is intended to run on resource-constrained handheld devices, performance has been a main criteria for the implementation decisions.

UDP Network This module is implemented as a local Android service. It runs on the main thread of the service stack and has an additional thread that monitors the reception of packets, calling the listener provided by the upper layer when a message is received. This component also initialises the ad-hoc network. A rooted phone is required for this purpose, since the network ad-hoc mode is not supported by the Android API and a set of native tools must be run with administrator rights.

Random Walk Gossip This module, implemented as a local service, runs on the main thread of the service stack and includes two additional threads for the Task Dispatcher and Wake One Packet components. These two threads only wake up when new tasks or actions are scheduled or ready to be executed by using signals and a timer respectively. This saves up resources spent on polling lists or checking the state of the system. Other aspects optimised are frequent protocol operations, such as the sending, reception, and management of messages, which involve:

1. *Creation, search, and elimination of packet entries in the Data Storage*: the Data Storage has a doubly-indexed organisation based on a HashMap and a TreeSet. The HashMap provides constant access to the entries by the ID of the message, operation repeated each time a packet arrives. The TreeSet provides fast sequential access, $O(n)$, to the entries sorted by the time-to-live parameter of the message to speed up the deletion of expired messages.
2. *Creation, search, and elimination of tasks in the Task Storage*: the Task Storage has a two-indexed organisation also based on a HashMap and a TreeSet. In this case the hashmap organises sets of tasks by message ID, which provides fast access, $O(\log n)$, to all the tasks related to a message, e.g. to remove them when the message expires. And the TreeSet provides fast creation and access, $O(\log n)$, to the tasks ordered by their release time.
3. *Marshalling and unmarshalling of packets to/from byte array*: the marshall and unmarshall methods of the RWGPacket class have been implemented minimising the creation and destruction of objects. It is also worth mentioning that special classes have been created to support unsigned integers and binary operations to deal with the headers of the network packets.

General Survivability Framework This module, implemented as a local service, runs on the main thread of the service stack and includes one additional thread for the IDS Engine component. This thread is waken up following the evaluation period of the IDS. The value chosen for this period is a trade-off between the IDS performance and the CPU utilisation.

Each of the four components (as described in Figure 5) that are part of the IDS loop have a well defined interface (IDataSource, IAnomalyDetector, IDiagnoser, and IMitigation) and are easily replaceable. This facilitates the implementation and evaluation of different detection and diagnosis algorithms and mitigation techniques. Other aspects that have been considered are:

1. *Anomaly detection frequency*: in earlier versions of GSF, the anomaly detector was triggered each time a packet was received, and later the alerts were aggregated to issue a verdict. In this implementation, the anomaly detection is triggered periodically, and when a packet is received, only the statistics to feed the detector are updated, reducing the number of operations performed.
2. *Marshalling and unmarshalling of packets to/from byte array*: the Data Source and Mitigation components of the IDS unmarshall the packets using the RWGPacket class. The IDataSource and IMitigation interfaces are independent of the routing protocol and oblivious to this class. Hence, a generic IPacket base class was created to ease the exchange of generic packet objects among the components. Thus, packets are unmarshalled by the Data Source and forwarded in this state to the Mitigation component, avoiding the need to do the operation twice.

4 Performance evaluation

This section describes the evaluation of the performance of the implemented services by testing over a simple setup with three Android smartphones. The main phones are a LG P990 Optimus 2x and a Samsung I9100 Galaxy S2, both with dual core ARM Cortex-A9 processors at 1Ghz and 1.2Ghz, 512MB and 1GB of RAM, and Android v2.2 and v2.3 respectively. The Samsung and LG phones, with implementations of RWG and GSF, were used to set up an ad-hoc network. An application to generate load, which creates messages periodically, has been installed in the LG phone. An HTC G1 smartphone and a regular laptop were used to create an attack and monitor the network respectively.

Four tests were performed with different network loads, half of them with GSF enabled and half of them with the Drain attack [10]. This attack sends ACK packets with random non-existing identities that trigger the retransmission of the messages that the neighbours store in their buffers. A explosion of packets sent to the network and processed by the receivers is created. The mitigation described in Cucurull *et al.* [10] has been implemented. Hence the ACK, OKTF, and BS packets coming from unknown nodes during an alarm are rejected (known nodes are the ones that sent at least one REQF packet during non alarm periods). This mitigation also includes a limited forwarding of the information contained in the *informed* vectors exchanged, although in this particular case without effect. The mitigation is strictly applied during periods in which an alarm is raised.

All the tests lasted 10 minutes, plus one minute for network initialisation. The last 5 minutes of each test contained the Drain attack. The normality models were trained during 5 minutes with each network load. The rate of the attack

was set to 2 ACK packets/second. The parameters used for the routing protocol are the ones described in Asplund *et al.* [1], except the time the protocol waits for ACK messages after sending a REQF. This parameter was set to 0.3 seconds following further studies on the Symbian platform [11]. To keep compatibility with previous implementations, the chosen size for the *informed* vectors was limited to 16 bits each. The time to live of the packets was set to 60 seconds, which scales it to be consistent with earlier simulations. The main parameter of GSF is the evaluation time, which is also used to derive the packet statistics. The value used was 5 seconds, that is enough to get relevant statistics in the network created and provide reasonable attack detection delays given the explained setup.

4.1 Evaluation scenario

The scenario comprised one phone (LG) sending messages and another one (Samsung) receiving and retransmitting them. The main purpose of these tests is to analyse the overhead of the GSF implementation and demonstrate its function.

The performance tests use four metrics: the CPU usage, the memory usage, the propagation latency for a message, and the Packet Transmission Rate (PTR) in the network. The CPU and memory usage were obtained with a small C application installed in the Samsung phone that monitored the process of the service. The CPU usage is read every 250ms and the values shown in the next sections are the average of all the observations. The CPU usage covers both CPU cores, i.e. when only one core is used at 100% the metric shows a total CPU usage of 50%. The propagation latency has been calculated in the LG phone. It is the average time elapsed starting from the time a message is sent until it is received back again because of its dissemination by the other phone. The PTR, which shows the total number of packets exchanged per second in the network, has been calculated with a laptop sniffing the traffic with the Wireshark application.

The tests have been performed in four different settings which are the product of enabling and disabling the GSF and applying or not the Drain attack.

4.2 CPU and memory usage

The system has been evaluated for different network loads up to 16 msg/sec. The average CPU usage obtained, depicted in Figure 6, show that:

- The GSF functionality only imposes a slight increment to the CPU usage. GSF, by the virtue of its design, has been optimised to consume as little resources as possible despite the fact that the evaluation interval is kept short for a faster detection latency.
- When GSF is enabled and an attack is introduced, the CPU usage is kept to levels similar to the case without attack, except for the load of 16 msg/sec. Instead, when an attack is introduced and GSF is not enabled, the CPU usage easily exceeds the 40%.

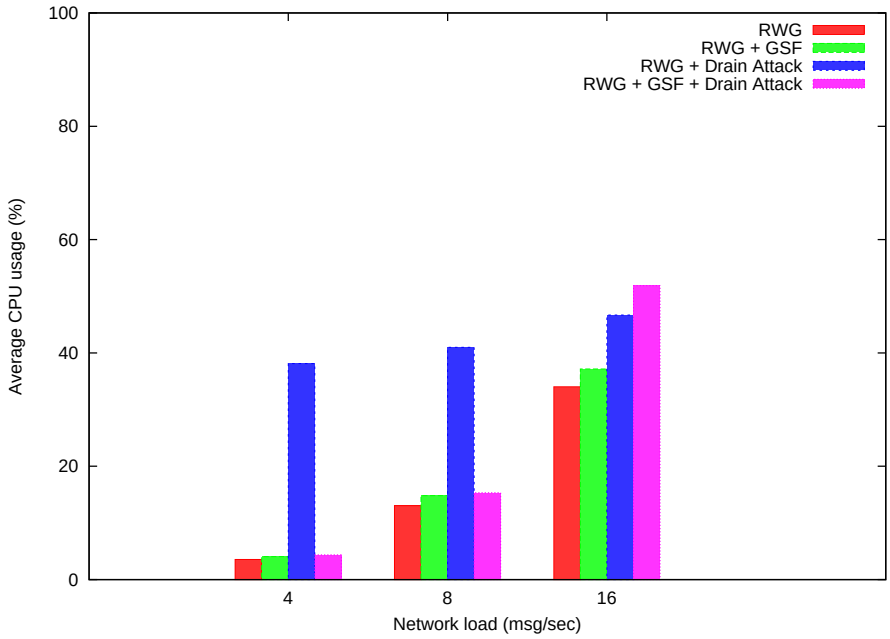

Fig. 6: Average CPU Usage

The maximum CPU usage in the three cases (with no attack, only the RWG + GSF on) was found to be 20%, 25%, and 84% respectively.

The CPU usage increase during the attack is a consequence of the high number of packets produced. The packets saturate the service causing message retransmissions that produce more saturation. When the load is set at 16 msg/sec, the attack produces a cascading chain of retransmissions that saturate the system before giving any chance to the GSF to completely mitigate the effects of the attack. Due to this increase of packets (around twice the normal rate) and the overhead of GSF processing them, the CPU usage is a bit higher than when GSF is not enabled.

Another aspect observed is that, although the service stack is composed of many threads, the CPU use is not well-balanced. Only one thread takes care of the received packets, and too many resource consuming operations are assigned to it. When the number of packets exceed a certain limit, one core is used at 100% while the other is idle. It is worth mentioning that most of the CPU usage is due to the cost of processing each packet received in the routing layer.

Regarding the memory, in all the cases the service stack has used between 21 and 27 MB. No significant changes have been observed either when enabling the GSF or in presence of the Drain attack.

4.3 Latency

The propagation latency has been evaluated for the same network loads. The results, depicted in Figure 7 with logarithmic scale, show that:

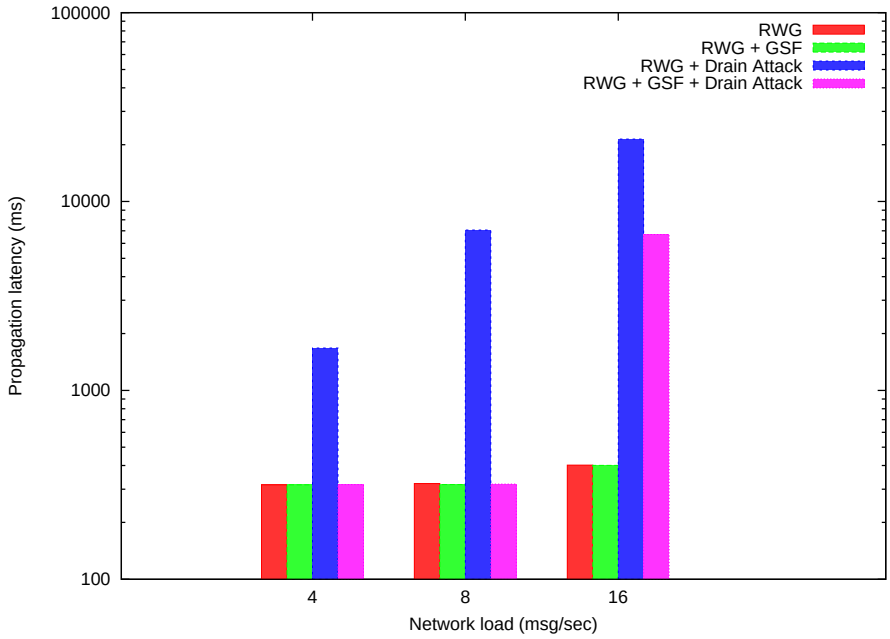

Fig. 7: Message propagation latency

- GSF does not impose an increment to the message latency when it is enabled.
- Similar to observations on the CPU usage, when GSF is enabled and an attack is introduced, the latency is kept close to the one without attack except in the high load case (16 msg/sec). Instead, if GSF is disabled the latency increases exponentially, because the attack saturates the network with unnecessary packets.

As mentioned before, with the 16 msg/sec load the whole network is destabilised before GSF can mitigate the attack.

4.4 Packet transmission rate

The PTR has also been evaluated for three loads with similar effects. When measuring PTR we are considering the total load on the network, both data (embedded in the REQF packet of the protocol) and signalling (ACK, OKTF,

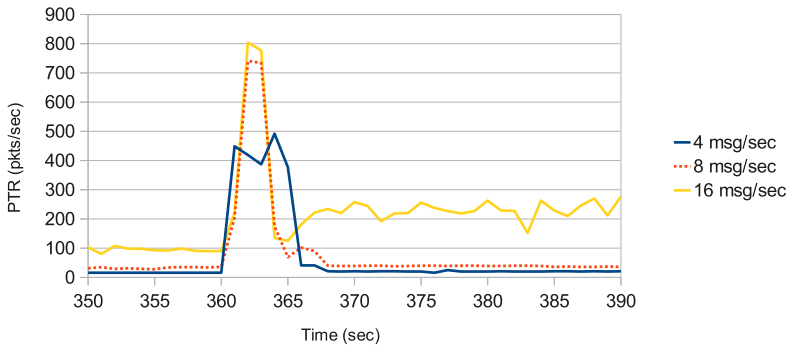

Fig. 8: Packet transmission rate

BS). Figure 8 shows a segment of a curve that represents the packet rate observed under different loads of data with the GSF enabled and the Drain attack.

The behaviour is analog to the one observed in previous simulations [10]. The normal rate without attack, until 360 seconds when the attack starts, is around 16, 30, and 100 packets/sec for the loads of 4, 8, and 16 messages/sec respectively. When the attack starts it increases to 400, 750, and 800 packets/sec respectively. But, after 5 seconds when the attack is detected, and matching with the evaluation interval, the rate decreases to around 20, 40, and 200 packets/sec because of the mitigation applied. These rates, after around 60 seconds, go back close to the initial ones, except for the 16 messages/sec load. This difference of rate before the attack and during mitigation is inherent to RWG and due to some retransmissions of messages lost during the traffic peak produced by the attack in the first 5 seconds.

5 Related work

With the large number of smartphones sold [12], the security aspects have become matter of concern. Recently, many studies [3, 13, 14] are devoted to the analysis of their security. Shabtai *et al.* [14] studies the security mechanisms integrated in the Android devices. It also includes a taxonomy of the identified threats and possible types of solutions. These include network malicious activity and draining mobile device's resources as threats and anomaly detection as potential solutions among others. In Android most of the intrusion detection work is devoted to malware detection. Nevertheless, the mechanisms applied share many characteristics with the ones applied for mobile ad-hoc networks.

Cheng *et al.* [4] present SmartSiren, a collaborative mechanism for virus detection and mitigation on smartphones. The system detects viruses spread via SMS/MMS and Bluetooth by monitoring the messaging activity.

Bose *et al.* [5] propose a behaviour-based malware detector for smartphones. It monitors the behaviour of the applications, taking into account the temporal patterns of the system calls and events, creating behaviour signatures. The signatures are compared and classified against a database that contains samples of fair applications and malware.

Schmidt *et al.* [6] presents a work on malware detection for smartphones. The work is focused on the selection and management of features to monitor the phone. The system is composed of a client (for Symbian S60 and Windows Mobile 6), installed in the phone. It periodically collects and sends a number of features to a server that analyse them with existing knowledge-based learning algorithms.

Kim *et al.* [7] describe a misuse detection approach based on monitoring the power consumption of smartphones. A database with power signatures of fair applications and malware is populated. They propose a two component approach that includes a client in the phone that collects the signatures and a data analyser, which can be in the phone or in a server, that performs the detection. The system is implemented over Windows Mobile 5.0.

Shabtai *et al.* [8] present Andromaly, a behavioural host-based malware detection system for Android devices. The system is composed of many components grouped as feature extractors, processors (for detection and analysis), main service, and graphical user interface. They perform extensive tests with six different machine learning classification algorithms and three feature selection algorithms to study the obtained detection performance. This paper shares the philosophy of our work, but it is applied to malware instead of network attacks.

Our approach, compared to Cheng *et al.* [4] and Schmidt *et al.* [6], is not dependent on external servers and, as opposite of Bose *et al.* [5], it is fully implemented over the smartphone. These two characteristics are very important in an ad-hoc environment in which communications are not reliable. The approach also includes mitigation, usually not present in other works [6, 7, 5, 8]. As shown in Section 4 the cost of our solution is not expensive in terms of CPU or memory usage overhead. A complete resource usage evaluation, as opposed to our work, is not usually provided [4–7].

6 Conclusions

This article described the design of an intrusion detection service for Android smartphones applied to network routing level. The design emphasises the modularity of the service to facilitate its implementation with different routing protocols and with different detection algorithms and attack mitigation techniques.

An implementation of the service on top of a dissemination protocol for intermittently-connected networks was developed. Its performance was tested and analysed using several metrics, such as the CPU and memory usage, the rate of packets exchanged in the network, and the latencies to propagate the messages. These tests demonstrated that the intrusion detection service produces

a low overhead on the phone and the network, and confirmed the reduction of the impact of attacks.

Further work can be done along two different lines. While the implementation has provided evidence for the feasibility of the deployment of a survivability framework, a more comprehensive implementation would need to address diagnoses of different known attacks and mitigations thereto. At the same time larger testbeds, with tens of smartphones, can be created to evaluate the capacity of the GSF to detect and mitigate attacks on a more realistic testbed. Further work could also include more detailed studies of the energy consumption and energy profiling of the whole service.

Acknowledgements

This work was supported by a grant from the Swedish Civil Contingencies Agency (MSB) and the national Graduate school in computer science (CUGS). We want also thank Ekhiotz Jon Vergara and Mikael Asplund for inspiration due to earlier works.

References

1. Asplund, M., Nadjm-Tehrani, S.: A partition-tolerant manycast algorithm for disaster area networks. IEEE Symposium on Reliable Distributed Systems (2009) 156–165
2. Vergara, E.J., Nadjm-Tehrani, S., Asplund, M., Zurutuza, U.: Resource footprint of a manycast protocol implementation on multiple mobile platforms. In: Next Generation Mobile Applications, Services and Technologies, 2011 . NGMAST '11. The Fifth International Conference on, IEEE (Sept. 2011) To appear.
3. Landman, M.: Managing smart phone security risks. In: 2010 Information Security Curriculum Development Conference. InfoSecCD '10, New York, NY, USA, ACM (2010) 145–155
4. Cheng, J., Wong, S.H., Yang, H., Lu, S.: SmartSiren: virus detection and alert for smartphones. In: Proceedings of the 5th international conference on Mobile systems, applications and services. MobiSys '07, New York, NY, USA, ACM (2007) 258–271
5. Bose, A., Hu, X., Shin, K.G., Park, T.: Behavioral detection of malware on mobile handsets. In: Proceeding of the 6th international conference on Mobile systems, applications, and services. MobiSys '08, New York, NY, USA, ACM (2008) 225–238
6. Schmidt, A.D., Peters, F., Lamour, F., Albayrak, S.: Monitoring smartphones for anomaly detection. In: Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications. MOBILWARE '08, ICST, Brussels, Belgium, Belgium, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) (2007) 40:1–40:6
7. Kim, H., Smith, J., Shin, K.G.: Detecting energy-greedy anomalies and mobile malware variants. In: Proceeding of the 6th international conference on Mobile systems, applications, and services. MobiSys '08, New York, NY, USA, ACM (2008) 239–252

8. Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., Weiss, Y.: Andromaly: a behavioral malware detection framework for Android devices. Journal of Intelligent Information Systems (2011) 1–30
9. Xenakis, C., Panos, C., Stavrakakis, I.: A comparative evaluation of intrusion detection architectures for mobile ad hoc networks. Computers & Security **30**(1) (2011) 63 – 80
10. Cucurull, J., Asplund, M., Nadjm-Tehrani, S.: Anomaly detection and mitigation for disaster area networks. In Jha, S., Sommer, R., Kreibich, C., eds.: Recent Advances in Intrusion Detection. Volume 6307 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2010) 339–359
11. Vergara, E.J.: Implementation of a manycast protocol for intermittently connected mobile ad hoc networks in disaster areas (2010) Master Thesis. Linköping University, RTSLAB - Real-Time Systems Laboratory.
12. IDC: Press Release (June 2011) http://www.idc.com/getdoc.jsp?containerId=prUS22871611.
13. Enck, W., Ongtang, M., McDaniel, P.: Understanding Android security. Security Privacy, IEEE **7**(1) (jan.-feb. 2009) 50 –57
14. Shabtai, A., Fledel, Y., Kanonov, U., Elovici, Y., Dolev, S., Glezer, C.: Google android: A comprehensive security assessment. Security Privacy, IEEE **8**(2) (march-april 2010) 35 –44