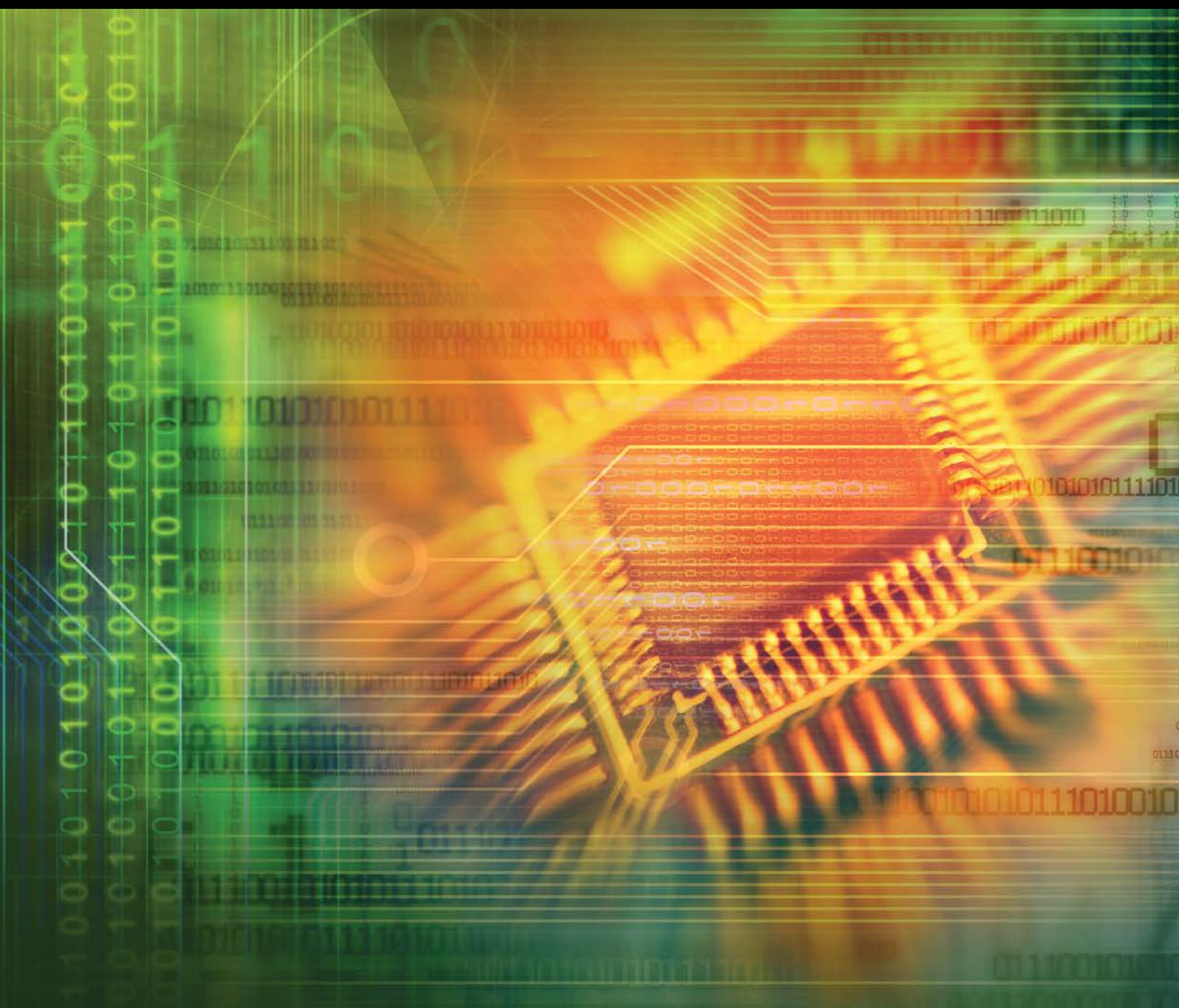
Journal of Electrical and Computer Engineering

System and Network Security: Anomaly Detection and Monitoring

Guest Editors: Michele Vadursi, Andrea Ceccarelli, Elias P. Duarte Jr., and Aniket Mahanti



System and Network Security: Anomaly Detection and Monitoring

Journal of Electrical and Computer Engineering

System and Network Security: Anomaly Detection and Monitoring

Guest Editors: Michele Vadursi, Andrea Ceccarelli, Elias P. Duarte Jr., and Aniket Mahanti

Circuits and Systems

Muhammad Abuelma'atti, KSA
Ishfaq Ahmad, USA
Dhamin Al-Khalili, Canada
Wael M. Badawy, Canada
Ivo Barbi, Brazil
Martin A. Brooke, USA
Tian-Sheuan Chang, Taiwan
M. Jamal Deen, Canada
Andre Ivanov, Canada
Wen B. Jone, USA
H. Kuntman, Turkey
Bin-Da Liu, Taiwan
Shen-Iuan Liu, Taiwan
João Antonio Martino, Brazil
Pianki Mazumder, USA
Sing Kiong Nguang, New Zealand
Shun Ohmi, Japan
Mohamed A. Osman, USA
Ping Feng Pai, Taiwan
Marco Platzner, Germany
Dhiraj K. Pradhan, UK
Gabriel Robins, USA
Mohamad Sawan, Canada
Raj Senani, India
Gianluca Setti, Italy
Nicolas Sklavos, Greece
Ahmed M. Soliman, Egypt
Dimitrios Soudris, Greece
Charles E. Stroud, USA
Ephraim Suhir, USA
Hannu A. Tenhunen, Finland
George S. Tombras, Greece
Spyros Tragoudas, USA
Chi Kong Tse, Hong Kong
Chin-Long Wey, USA
Fei Yuan, Canada

Communications

Sofiène Affes, Canada
Edward Au, China
Enzo Baccarelli, Italy
Stefano Basagni, USA
Jun Bi, China
René Cumplido, Mexico
Luca De Nardis, Italy
M.-G. Di Benedetto, Italy
Jocelyn Fiorina, France
Zabih F. Ghassemlooy, UK
K. Giridhar, India
Amoakoh Gyasi-Agyei, Ghana
Yaohui Jin, China
Peter Jung, Germany
Adnan Kavak, Turkey
Rajesh Khanna, India
Kiseon Kim, Republic of Korea
Tho Le-Ngoc, Canada
Cyril Leung, Canada
Petri Mähönen, Germany
Jit S. Mandeep, Malaysia
Montse Najar, Spain
Adam Panagos, USA
Samuel Pierre, Canada
John N. Sahalos, Greece
Christian Schlegel, Canada
Vinod Sharma, India
Iickho Song, Republic of Korea
Ioannis Tomkos, Greece
Chien Cheng Tseng, Taiwan
George Tsoulos, Greece
Jian-Kang Zhang, Canada
M. Abdul Matin, Brunei Darussalam

Signal Processing

Sos Agaian, USA
Panajotis Agathoklis, Canada
Jaakko Astola, Finland
Anthony Constantinides, UK
Paul Dan Cristea, Romania
Petar M. Djuric, USA
Igor Djurović, Montenegro
Karen Egiazarian, Finland
Woon-Seng Gan, Singapore
Zabih Ghassemlooy, UK
Martin Haardt, Germany
Jiri Jan, Czech Republic
S. Jensen, Denmark
Chi Chung Ko, Singapore
James Lam, Hong Kong
Riccardo Leonardi, Italy
Sven Nordholm, Australia
Cédric Richard, France
William Sandham, UK
Ravi Sankar, USA
Andreas Spanias, USA
Yannis Stylianou, Greece
Ioan Tabus, Finland
Ari J. Visa, Finland
Jar Ferr Yang, Taiwan

Contents

Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2016, Article ID 2093790, 2 pages
http://dx.doi.org/10.1155/2016/2093790

Editorial

System and Network Security: Anomaly Detection and Monitoring

Michele Vadursi,[1] Andrea Ceccarelli,[2] Elias P. Duarte Jr.,[3] and Aniket Mahanti[4]

[1] *University of Naples "Parthenope", 80143 Napoli, Italy*
[2] *University of Florence, 50134 Florence, Italy*
[3] *Federal University of Paraná, 19018 Curitiba, PR, Brazil*
[4] *University of Auckland, Auckland 1142, New Zealand*

Correspondence should be addressed to Michele Vadursi; vadursi@uniparthenope.it

Received 9 May 2016; Accepted 9 May 2016

Large-scale systems and networks often operate under variable and unpredictable conditions, thus requiring efficient and adaptive monitoring and error detection solutions. Furthermore, the increasing complexity and dynamicity of current systems and networks ask for solutions that infer the status by looking for anomalies rather than directly detecting errors. Anomalous behavior is an indication not only of hardware and software faults, but also of security threats including intrusion attempts and frauds, which represent an increasingly relevant challenge from both scientific and socioeconomic point of view. The timely identification of anomalies in dependable systems allows timely error and security threat detection which can trigger appropriate reactions.

This special issue covers a wide range of topics that are of interest to researchers and practitioners in the field of security and anomaly detection in computer systems and networks. The papers contained in this special issue include research articles focused on network intrusion detection, malware detection in mobile devices, clock synchronization vulnerabilities in industrial networks, privacy preservation in IP version 6, and abrupt changes of the available bandwidth.

Distributed Denial of Service (DDoS) attacks are constructed by malicious entities by flooding the target host with traffic thus denying it from servicing legitimate requests. Network intrusion detection systems are deployed to identify and thwart such attacks. Several techniques based on signatures and observed anomalies have been proposed in the literature. The paper by Ö. Cepheli et al. entitled "Hybrid Intrusion Detection System for DDoS Attacks" proposes a hybrid framework combining signature-based and anomaly-based methods for improved DDoS attack detection.

Intrusion detection involves sifting through large amounts of network traffic. Data compression can improve the efficacy of the intrusion detection system. The paper entitled "SVM Intrusion Detection Model Based on Compressed Sampling" by S. Chen et al. presents a Support Vector Machine (SVM) intrusion detection model based on compressive sampling. The paper shows that by using compressed sensing theory the proposed SVM intrusion detection system can utilize a small sample of the network data for training its classifiers and detection time is reduced.

With mobile device sales surpassing those of desktop devices, more people are connecting to the Internet through their smartphones and tablets. This shift to a new platform has attracted the attention of attackers to target mobile devices. O. Somarriba et al. in their paper entitled "Detection and Visualization of Android Malware Behavior" present a monitoring architecture to identify malicious Android applications.

Clock synchronization is an important requirement in several industrial networks such as automation, stock market, and telecommunications. The IEEE 1588 standard allows clock synchronization across the nodes in an Ethernet network; however, this standard does not provide adequate security. In the paper entitled "Protecting Clock Synchronization: Adversary Detection through Network Monitoring" E. Lisova et al. describe clock synchronization vulnerabilities and evaluate solutions to mitigate these attacks.

Entities sharing sensitive information over the Internet should remain anonymous. Address rotation of the sender and receiver can prevent an attacker from discovering the identities of the communicating parties. The Moving Target IPv6 Defense (MT6D) architecture implements user anonymity by automatically changing IP version 6 addresses. D. Basam et al. in their paper entitled "Strengthening MT6D Defenses with LXC-Based Honeypot Capabilities" extend their work on MT6D to study suspicious activity on the discarded addresses and strengthen the MT6D parameters.

Available bandwidth is an important network performance metric, which helps in routing, Quality of Service (QoS), and traffic engineering on the Internet. D. Santoro and M. Vadursi in their paper entitled "Performance Analysis of a DEKF for Available Bandwidth Measurement" present a characterization of a measurement algorithm based on a Discrete-time Extended Kalman Filter (DEKF) for tracking abrupt changes of the available bandwidth.

We sincerely believe this special issue has highlighted relevant emerging issues in security of computer systems and networks, in particular the Internet. We hope the research results presented in this special issue will enable the research community to further the field, by proposing novel and efficient solutions to challenges facing the computer systems and network security community.

Acknowledgments

We thank the authors who made submissions to this special issue and the reviewers for their support and detailed reviews in making this special issue possible.

Michele Vadursi
Andrea Ceccarelli
Elias P. Duarte Jr.
Aniket Mahanti

Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2016, Article ID 6297476, 13 pages
http://dx.doi.org/10.1155/2016/6297476

Research Article

Protecting Clock Synchronization: Adversary Detection through Network Monitoring

Elena Lisova,[1] Marina Gutiérrez,[1,2] Wilfried Steiner,[2] Elisabeth Uhlemann,[1] Johan Åkerberg,[1] Radu Dobrin,[1] and Mats Björkman[1]

[1] *School of Innovation, Design and Engineering, Mälardalen University, Västerås, Sweden*
[2] *TTTech Computertechnik AG, Vienna, Austria*

Correspondence should be addressed to Marina Gutiérrez; marina.gutierrez@tttech.com

Received 10 December 2015; Revised 22 February 2016; Accepted 27 March 2016

Academic Editor: Andrea Ceccarelli

Nowadays, industrial networks are often used for safety-critical applications with real-time requirements. Such applications usually have a time-triggered nature with message scheduling as a core property. Scheduling requires nodes to share the same notion of time, that is, to be synchronized. Therefore, clock synchronization is a fundamental asset in real-time networks. However, since typical standards for clock synchronization, for example, IEEE 1588, do not provide the required level of security, it raises the question of clock synchronization protection. In this paper, we identify a way to break synchronization based on the IEEE 1588 standard, by conducting a man-in-the-middle (MIM) attack followed by a delay attack. A MIM attack can be accomplished through, for example, Address Resolution Protocol (ARP) poisoning. Using the AVISPA tool, we evaluate the potential to perform a delay attack using ARP poisoning and analyze its consequences showing both that the attack can, indeed, break clock synchronization and that some design choices, such as a relaxed synchronization condition mode, delay bounding, and using knowledge of environmental conditions, can make the network more robust/resilient against these kinds of attacks. Lastly, a Configuration Agent is proposed to monitor and detect anomalies introduced by an adversary performing attacks targeting clock synchronization.

1. Introduction

One of the specific characteristics of industrial networks is a high cost of failure, resulting in money loss, environmental threats, or damage to humans. Today, such networks grow extremely fast in complexity and functionality leading to an increasing number of security requirements [1]. Developing a security framework for the whole network implies taking into account application specific features and related systems assets [2]. Clock synchronization is an essential part of all networks with real-time requirements. Basically all industrial networks have real-time requirements, and therefore most messages have deadlines to meet. Consequently, if there is a way to breach clock synchronization, it will disrupt the network functionality, and, moreover, it can be applied to a range of different networks regardless of their specific application area [3]. This fact leads to an increased possibility for an adversary to invest resources in such attacks. Also, it is a motivation to investigate possible ways of clock synchronization protection, which includes adversary detection and consequences mitigation.

Most synchronization algorithms, for example, the IEEE 1588 standard, are vulnerable to delay attacks, as they rely on measuring delays without taking adversary attacks into consideration. A possible way to breach clock synchronization was proposed in [4], namely, a combination of an ARP poisoning attack followed by a delay attack. To conduct a delay attack, an adversary first needs to penetrate the network. A so-called man-in-the-middle (MIM) attack is one possible way to take control over a communication channel. When performing a MIM attack, the adversary is positioned inside the communication channel between benign participants, for example, through ARP poisoning. An ARP poisoning attack takes advantage of the vulnerability in the ARP protocol, which is used for translating between IP and MAC addresses. If an adversary can convince a

benign network participant N_1 to connect the adversary MAC address to the IP address of another benign node N_2, with whom N_1 wants to communicate, the adversary will get all traffic sent by N_1 to N_2. When the adversary controls the channel, the next step in order to break clock synchronization is to perform a selective delay attack. The combination of these two techniques can be used to breach clock synchronization.

An intrusion detection system (IDS) entails monitoring a network for security purposes. This is not a new approach, and a Network Security Monitor (NSM) system was described already in 1994 [5]. Statistics gathered from a network can be used for many functionalities, such as QoS enhancement, network adaptation for having higher resilience levels, and routing optimization. Usually, an adversary penetrating the network changes its behavior in some way. While a passive adversary mainly monitors and analyzes incoming data, the gathered statistics can be used by an active adversary in an upcoming intelligent attack. In case of an active adversary, communication with other network participants is also common. This means that the adversary will introduce new traffic and/or change the traffic pattern. Therefore, if the Monitor, collecting and analyzing the network statistics, can detect a deviation, the adversary presence can be discovered. It is the first step in network protection, which should be followed by adversary isolation or neutralization and consequences mitigation.

The main contribution of the paper is a detailed problem formulation of clock synchronization vulnerabilities along with evaluation of possible techniques to mitigate consequences of clock synchronization breaking. The investigation of the possibility to detect a malicious adversary targeting clock synchronization breaching in industrial networks using IEEE 1588 for clock synchronization was also conducted. The range of possible attacks is narrowed down to a delay attack performed after a successfully conducted MIM attack. Two scenarios for conducting a MIM attack via ARP poisoning with single and multiple network penetrations are considered. Such choice of scenarios allows conducting a discussion about the difference in detection and the following adversary localization once they are formally evaluated in AVISPA [6]. The ARP poisoning attack is well known, but we still conduct a formal evaluation, as this allows evaluating possible mitigation techniques in at the same time and comparing possible solutions. Further, two different ways of performing a delay attack are simulated in OMNeT++ [7] in order to see how these ways can affect traffic characteristics. The results are also discussed from two points of view: the adversary and the Monitor. Our results show that the likelihood of adversary detection depends on many factors, such as the prior knowledge of the network available to the adversary, the knowledge of the network history available to the Monitor, and the ability of the network to switch to Relaxed Mode, that is, allowing additional clock drifts. Environmental conditions are also considered as an additional factor for clock synchronization disturbance and a technique using these conditions for adversary detection is proposed. Finally, another technique that can detect the presence of an adversary, namely, delay bounding, is discussed and evaluated.

The remainder of the paper is organized as follows. Section 2 presents related works and Section 3 introduces some background information regarding clock synchronization, in particular the IEEE 1588 standard, and IDS. Next, the system model including models of the adversary and the Monitor are described in Section 4. In Section 5, the MIM attack is investigated along with the possible consequences of a delay attack for clock synchronization, whereas Section 6 describes the proposed solution for attack detection and countermeasures discussion. Results of the attack evaluation and monitoring simulations are presented in Section 7. Finally Section 8 concludes the paper.

2. Related Works

The initial version of the IEEE 1588 standard does not have any security services. Based on this, the authors of [8] show the effects of a delay attack on the IEEE 1588 Precision Time Protocol (PTP). Our work can be considered as an extension of the above-mentioned paper, as we propose a possible way of conducting the delay attack and widen the scope of possible mitigation techniques. In release 2008, Annex K was added to the standard to provide a set of security solutions [9]. However, the amendment provides only a very limited set of services. Annex K provides guidelines for message integrity protection and group source authentication. These security solutions do not help against a delay attack, as in this case the adversary does not need to change the messages or create new ones—it simply delays them. Mizrahi applied game theoretical approach to analyze the delay attack influence on clock synchronization [10]. As a result, the author proposed to use a multiple-path approach to prevent and mitigate delay attacks. However, this approach is not compatible with IEEE 1588.

In [11], network monitoring has been used in the context of synchronized networks. The concept of a Configuration Agent is introduced: an autonomous entity that learns the characteristics of the network through continuous monitoring, with the goal of facilitating the configuration and reconfiguration of time-triggered networks. The Configuration Agent is composed of four elements: Monitor, Extractor, Scheduler, and Reconfigurator. The duple formed by the Monitor and the Extractor gathers data from the network and distills relevant information from it. In [12], that information is sent to the Scheduler so it can produce a new schedule for the network, with which the network is reconfigured. For this paper, we propose to use a generalized view of the same concept, so we replace the Scheduler with a Diagnoser, which, after the learning phase performed by the Monitor and the Extractor, will use the distilled information to decide what actions to take. Finally the Reconfigurator carries out those actions.

An example of a Network Security Monitor running on Ethernet and applied for LANs is presented in [13]. The authors proposed a hierarchical model of data analysis that allows separation of network activities into host-to-host, services, and connections groups. In [14], traffic patterns are proposed to be used for detection of periodic communication of Botnets, subnetworks consisting of infected devices

remotely observed by the adversary. These two examples are from different areas and are separated significantly in time, demonstrating that security and monitoring can complement each other in an efficient way.

As mentioned above, applying network monitoring techniques to security issues leads to the development of an IDS. There are two main types of IDS depending on the logic of detection [15]. The first one is a signature-based network IDS. In this approach, there is a set of known attacks together with their corresponding patterns. The IDS is monitoring the system and raises an alarm, when there is a system pattern matching the one from the set. This approach has an obvious limitation: if there is an attack that was not considered at the development phase, it will not be detected. The second group is called heuristic or anomaly-based IDS. With this approach, the system instead knows some standard ways of behavior of the network and searches for any anomaly, anything that does not match the standard pattern. The advantage of this method is the possibility to detect a previously unknown attack. On the other hand, if the intruder knows the specific network patterns, the adversary actions can be masked and indistinguishable from the normal network behavior. The patterns of communication in conjunction with clock synchronization algorithms, which are in the main scope of this paper, are well known. Therefore, in our case we are targeting a hybrid technique, that is, combining a heuristic and a signature-based method. In this case, the Monitor can be more flexible and have a higher probability of an attack detection.

When evaluating a new solution or mitigation technique, it is essential to have some benchmarks and evaluation criteria. In [16], the authors propose a metric-based approach for IDS evaluation. Logical, architectural, and performance metrics were presented as the main groups of criteria. Logical metrics imply such characteristics as cost, maintainability, and manageability. Adjustable sensitivity, data pool scalability, data storage, and similar characteristics can be considered as architectural metrics. Finally, error reporting and recovery, induced traffic latency, operational performance impact, and observed false positive and negative ratios represent performance metrics. The scope of Monitor evaluation used in this paper is to show the impact of the Monitor on the network and the efficiency in attack detection.

3. Background

3.1. Clock Synchronization. In order to be able to cooperate, nodes of industrial networks have to share the notion of time, that is, be synchronized. In the ideal case, every node has perfect clock and simply follows the schedule based on its time. In reality, each clock has a natural drift. This drift can be different, mostly depending on the cost of the clock: usually the more expensive the clock is, the more accurate it is. Clock synchronization algorithms are used to assist the nodes with clock correction. Clock drift is a natural characteristic caused by the underlying physical oscillators. Therefore, it cannot be completely eliminated, only mitigated, that is, periodically compensated for. In industrial applications, often only the relative clock differences are important for network correct performance. Consequently, such clocks should be synchronized with each other rather than with an external time reference (e.g., such as GPS).

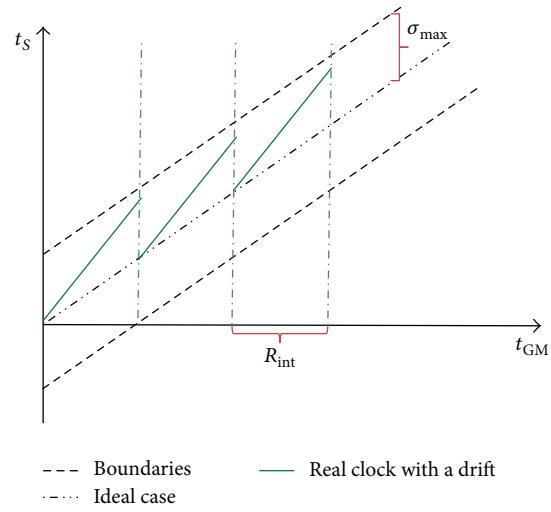


Figure 1: Periodic correction of slave clock time, t_S, according to grandmaster clock time, t_{GM}.

Clock correction is done periodically as is shown in Figure 1. The purpose of clock correction is to keep clock time within acceptable boundaries (dashed black lines). The ideal case would be that times provided by the grandmaster clock and the slave clock are always the same (black line with two dots and a dash). But, in reality, the slave clock will drift apart from the grandmaster clock (solid green line) and, therefore, it needs to be periodically corrected.

3.2. IEEE 1588 Standard. Given two clocks, A and B, in a network such that they have been synchronized in a moment in the past, that is, $t_A = t_B$, in a later moment the time values provided by these clocks will have drifted apart as follows:

$$\left|t_A - t_B\right| = \sigma. \tag{1}$$

This drift is caused by the nonideality of the clocks and environmental conditions, for example, heat affecting the frequency of the oscillators.

To avoid failures, caused by inability of applications to meet their deadlines, clock synchronization protocols are used. Among those, IEEE 1588 is a standard widely used in industrial applications for providing and maintaining clock synchronization [17]. In the IEEE 1588 standard, one of the nodes is chosen as a grandmaster GM, and the rest of the nodes are referred to as slaves S_i. Slaves are synchronized to the grandmaster, such that the differences in time values provided by the local clocks in the nodes, as expressed in (1), are bounded. This is expressed in the synchronization condition

$$\left|t_{S_i} - t_{\text{GM}}\right| < \sigma_{\max} \tag{2}$$

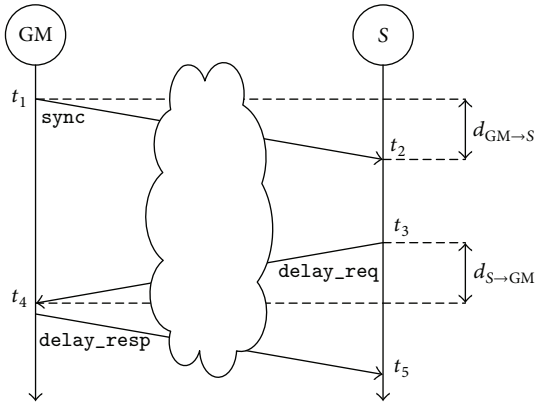


Figure 2: Clock synchronization protocol.

that should be constantly preserved. Here, σ_{max} is a parameter that should be chosen so that the application requirements are satisfied. The minimum value of σ_{max} can be calculated as

$$\min(\sigma_{max}) = 2\max(\rho_i) R_{int}, \tag{3}$$

where ρ_i are the clock drifts of the clocks in the network and R_{int} is the resynchronization interval. In Figure 1, it can be seen how IEEE 1588 works by periodically adjusting the value of the slave clock. The period of these corrections is the resynchronization interval. The value of the offset used to correct the slave clock is calculated applying the protocol depicted in Figure 2. The clock synchronization protocol consists of a series of messages being exchanged and time-stamped between the grandmaster and the slave in order to gain enough information to calculate the offset. This process is repeated periodically in every resynchronization interval R_{int}. The message exchange is as follows:

(1) At $t = t_1$ the grandmaster sends a synchronization message (`sync` in Figure 2) containing t_1 to the slave.

(2) At $t = t_2$ the slave receives the `sync` message. Now both t_1 and t_2 are recorded in the slave.

(3) At $t = t_3$ the slave sends out the delay request message (`delay_req` in Figure 2). t_3 is also recorded in the slave.

(4) At $t = t_4$ the grandmaster receives the `delay_req` message.

(5) At $t > t_4$ the grandmaster sends the delay response message (`delay_resp` in Figure 2) containing t_4 to the slave. When the slave receives the `delay_resp` message, lastly, t_4 is recorded.

Finally when all the time-stamps have been collected by the slave, the offset, σ_{meas}, can be calculated according to

$$\begin{aligned} d_{GM\to S} + \sigma_{meas} &= t_2 - t_1, \\ d_{S\to GM} - \sigma_{meas} &= t_4 - t_3, \end{aligned} \tag{4}$$

where $d_{GM\to S}$ and $d_{S\to GM}$ are the transmission delays of a message going from the grandmaster to the slave and from the slave to the grandmaster, respectively. Now if we assume that the transmission delay is symmetric, that is, it is the same in both directions, then $d_{GM\to S} = d_{S\to GM} = d_0$ and the measured offset is

$$\sigma_{meas} = \frac{1}{2}\left((t_2 - t_1) - (t_4 - t_3)\right). \tag{5}$$

Ideally, this value of the measured offset reflects the difference between the grandmaster clock and the slave at the moment when the offset is measured

$$t_S - t_{GM} = \sigma_{meas}. \tag{6}$$

In the IEEE 1588 standard, the equation for offset calculation is more complicated, as there are parameters compensating the propagation delays. For the sake of simplicity they are omitted here, as they do not change the logic of the protocol and do not make any significant difference for conducting a delay attack.

The standard defines three types of clocks; they are *transparent*, *boundary*, and *ordinary*, respectively. A transparent clock performs hardware time-stamping of synchronization messages and updates the corresponding fields in them. A boundary clock has one of its ports in slave mode and gets the time from the grandmaster via this port. It does not update synchronization messages but can create new ones with the time-stamps according to the information provided by the slave port and sends them out. An ordinary clock is a clock without any specific additional functions.

In addition, the standard defines two possible operation modes and two modes for synchronization messages exchange. The following operational modes are possible: end-to-end and peer-to-peer. In the first mode, clocks get information about the delays in links from the exchange of synchronization messages each time they want to make a correction. In the second mode, this exchange of messages is performed for all links regularly and without relation to the clock correction events. Each time a clock wants to correct its time, it has information about all delays with all neighbours. The end-to-end operational mode is suitable for networks where it cannot be guaranteed that all devices in the network support IEEE 1588. The peer-to-peer mode implies that the IEEE 1588 standard is supported by all devices in the network. All above-mentioned synchronization message exchanges can be performed in two or in four steps. In the second variant, follow-up messages are used to provide more precise time-stamps.

4. System Model

Considering possible ways of attacking the system and analyzing the system reaction to the intrusion, it is important to set the limits and assumptions of the investigated scenarios. There are many possible ways of interaction between the adversary and the system depending on the assumptions made for both separately and jointly. In this paper we consider wired networks, wherein synchronization is established and maintained according to IEEE 1588. Using peer-to-peer mode in the network implies that network participants periodically

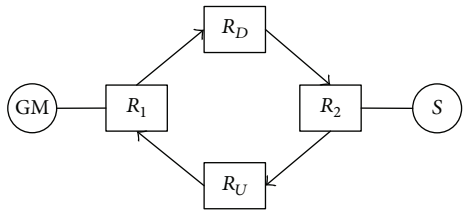


Figure 3: The network topology used for the simulations.

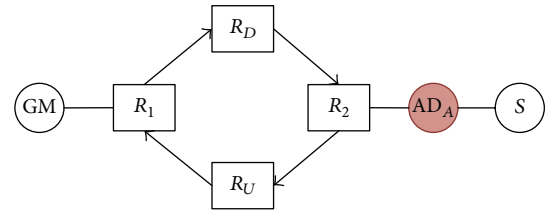


Figure 4: Scenario A for the network, AD_A—an adversary.

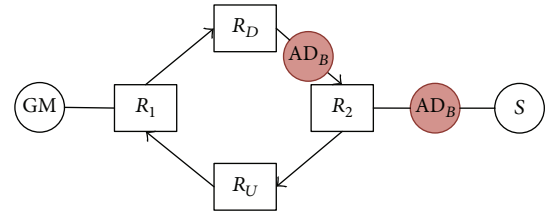


Figure 5: Scenario B for the network, AD_B —an adversary.

exchange messages to be aware of delays in the channels between them and their neighbours. The networks consist of routers, capable of messages time-stamping, and nodes, particularly grandmasters and slaves. Routers and nodes have the transparent type of clocks, which means that they perform hardware time-stamping of synchronization messages upon arrival and transmission, via an update of the correction field in the follow-up messages.

Figure 3 shows the sample topology. The network consists of a grandmaster, a slave, and a set of routers. There are two separate communication channels between the grandmaster and the slave, namely, downlink and uplink.

4.1. Adversary Model and Goal. To perform an attack analysis, the adversary model should be specified first. We assume that the adversary has access to and initial knowledge about the network. The adversary knows which node or communication link it is going to attack. The adversary primarily targets clock synchronization breaking; therefore, it attacks a link which is involved in the synchronization procedure. In addition to the assumptions mentioned above, the adversary choices can be random or based on an analysis of the network conducted in advance. We consider a case when the adversary attacks only communication channels and links. In this case, the adversary is capable of receiving, transmitting, and delaying messages. At this point, the capability of learning (i.e., the possibility to analyze the reaction of the opponent and connect consequences with causes) and behavioral adaptation is not considered. The main adversary goal is system disruption, that is, the adversary intends to cause system error and propagate it as much as possible, so that it leads to a system failure. Also, the adversary aims to prolong its influence and stay undetected. We assume that the behavior is rational in pursuing the above-mentioned goals.

We investigate the case when the adversary uses an ARP poisoning attack to penetrate the network and take control over the communications in the targeted channel; that is, the adversary conducts a man-in-the-middle attack. The next phase for the adversary is to perform a selective delay attack targeting synchronization. As specified in the adversary description above, the objective of the attack is a link. In this paper, we consider two scenarios as shown in Figures 4 and 5. Scenario A is a case when one link is under attack. Here the adversary controls the communication between a router R_2 and a slave S. The second considered case, scenario B, is a consecutive attack on two links. This case is represented by AD_B in Figure 5; namely, the adversary controls the communication channels between the router R_D and the router R_2 and between the router R_2 and the slave S. These two links under attack are parts of one logical channel between the grandmaster GM and the slave S. This scenario implies that the adversary can interfere with the targeted message in different parts of its propagation path ($R_D \rightarrow R_2$ or $R_2 \rightarrow S$). This difference is important for the choice of mitigation techniques. Even though the results will look exactly the same for the slave, it can be more difficult to distinguish the adversary from natural network disturbances during the delays analysis. Furthermore, following adversary detection, it can be more difficult to locate the adversary. Also, such a scenario can serve as a basis for modeling of a compromised router. It can be achieved if the possibility to change the contents of the messages is added to the adversary skills set, as then the adversary can actually replace the router from a functional point of view.

4.2. Configuration Agent Model. The four elements that comprise the Configuration Agent, which we propose to introduce in the network to detect possible attacks, can be seen in Figure 6. First, the Monitor gathers traffic measurements. Next, the Extractor transforms these traffic measurements into relevant traffic parameters. Further, by analyzing the traffic parameters, the Diagnoser is able to detect if there are anomalies in the traffic patterns and determine which are the correct actions to take. Finally, the Reconfigurator changes the configuration of the network to introduce the changes proposed by the Diagnoser.

In this paper, we assume that all the functionalities of the Configuration Agent take place locally in every slave in the network. This means that the only information that the Configuration Agent has is that gathered by the Monitor in a given slave. This approach presents both limitations and advantages. On the one hand it guarantees that the detection and mitigation processes are not affected by the same attack that we are trying to detect. On the other hand, having a global view of the network allows us to combine the information gathered in every slave, which can certainly help

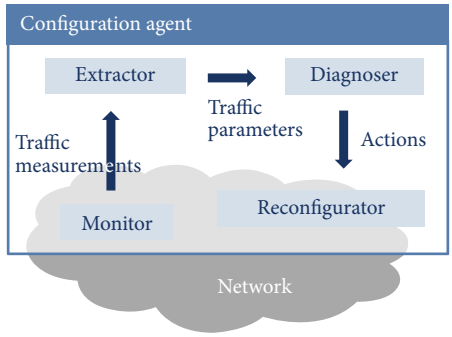


Figure 6: Configuration Agent overview.

with the detection of the attack. That scenario will however not be explored until future work.

5. Vulnerability Analysis

5.1. ARP Poisoning as a Method of Performing a MIM Attack. ARP is a network layer protocol used to define the correlation between MAC and IP addresses for the network participants. When a node N_1 wants to send a message to a node N_2, N_1 knows the IP address of N_2, but to send the message it also needs to know the related MAC address. N_1 first checks its table of IP and associated MAC addresses. If it cannot find the MAC address of N_2 in the table, it sends out a broadcast message requesting the node that has the IP address of N_2 to reply and send back its related MAC address. This communication is completely unprotected and hence vulnerable. Whenever any node receives an ARP reply, it overwrites its table even if it has not sent the request. A MIM attack is an attack when an adversary controls the communication channel between two parties. The parties believe that they communicate with each other directly, but in reality all data exchange is going through the adversary, who possibly can influence the data. An ARP poisoning attack is an attack using the ARP algorithm vulnerability to perform a MIM attack. A malicious adversary can send an ARP response to N_1 pretending to be N_2, and one to N_2 pretending to be N_1. As a result, communication between N_1 and N_2 will go through the adversary. Even though this is a well known attack, it still remains valid and possible, and is used for network penetrations [18].

5.2. Possible Targets of the Attack. ARP poisoning redirects traffic and allows the adversary to control communications between specific sets of MAC addresses. Depending on the adversary goal and the specific application, there are two possible targets of such an attack. Either the adversary can target a concrete communication link, hereafter referred to as scenario *A*, or it can target a specific device in the network, implying that more than one link needs to be compromised, termed scenario *B*. In case of scenario *A*, the adversary controls the communication between the two devices on each side of the link. The adversary can influence both devices in a harmful way if its presence is undetected. Alternatively, a specific device in the network can be chosen as a target. Scenario *B* highlights how an adversary can gain additional advantages, for example, making its localization more complicated. If the adversary can perform several simultaneous MIM attacks such that it is able to control all incoming and outgoing traffic for one specific device, it simulates the situation of a compromised device through ARP poisoning. Depending on the topology this can have different levels of complexity. The most appealing target for the adversary is most likely a grandmaster clock, as through this clock it can influence all slave clocks connected to it. According to the adversary model, if a clock is compromised, the adversary is capable of creating new synchronization messages, as well as delaying or accelerating messages it forwards. The latter can be achieved by changing the schedule of the clock. In this paper it is assumed that the adversary only performs attacks targeting links, that is, scenario *A*.

5.3. Consequences of the Delay Attack. In the two scenarios *A* and *B* described above, once the first stage of the attack has been performed, that is, the link has been compromised, the attacker performs the same action in both cases: it introduces a delay in the synchronization messages. In this section, an analysis of the consequences of introducing delays on the time synchronization in the network is shown.

If we use the value of the offset as obtained in (5) to correct the slave clock, we have

$$t_S^{\text{old}} \longrightarrow t_S^{\text{new}} + \sigma_{\text{meas}}. \tag{7}$$

Using (7) in (6) we obtain $t_S^{\text{new}} - t_{\text{GM}} = 0$, making it the best possible value for the offset. This is the value that we would obtain in an attack-free situation; therefore henceforth it will be referred to as σ_{af}:

$$\sigma_{\text{af}} = \frac{1}{2}\left((t_2 - t_1) - (t_4 - t_3)\right). \tag{8}$$

The value of the offset obtained above assumes that the transmission delay is symmetric. However, the adversary in the attack that we are considering (Figure 7) introduces an asymmetric delay, d_{adv}, that affects the synchronization messages in the following way:

$$\begin{aligned} d_{\text{GM}\to S} &= d_0 + d_{\text{adv}}, \\ d_{S\to\text{GM}} &= d_0. \end{aligned} \tag{9}$$

Now the time-stamps collected by the slave through the synchronization protocol described in Section 3.2 are

$$\begin{aligned} t_1' &= t_1, \\ t_2' &= t_2 + d_{\text{adv}}, \\ t_3' &= t_3 + d_{\text{adv}}, \\ t_4' &= t_4 + d_{\text{adv}}. \end{aligned} \tag{10}$$

Substituting (10) to (5) and using (8), we obtain the value of the measured offset when there is a delay introduced by the adversary according to

$$\sigma_{\text{meas}} = \sigma_{\text{af}} + \frac{1}{2}d_{\text{adv}}. \tag{11}$$

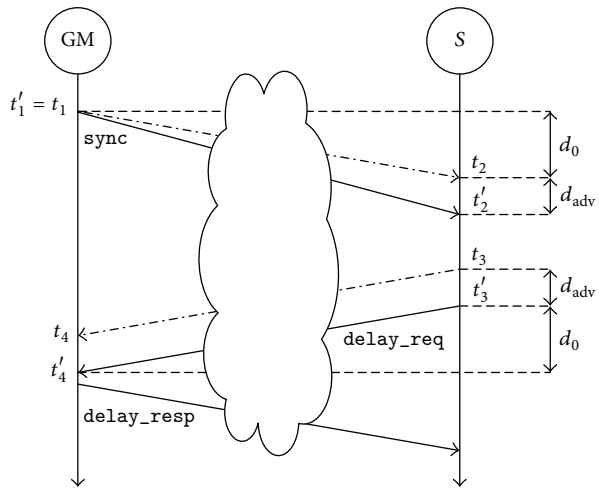


Figure 7: Clock synchronization protocol under attack.

If the adversary delays the delay request message instead of the synchronization message, only the sign of d_{adv} in (11) will change. The choice of message to delay does not affect the following reasoning. If σ_{meas} is used to correct the slave clock, then $t_S^{old} \rightarrow t_S^{new} + \sigma_{meas}$ and because initially $t_S - t_{GM} = \sigma_{af}$, the difference between the slave clock and the grandmaster is now

$$t_S - t_{GM} = \sigma_{af} - \sigma_{meas} = \frac{1}{2} d_{adv}. \tag{12}$$

Take into account that this is the value just after the correction, from where the slave clock will start drifting again, and thus for the next resynchronization interval it will be

$$t_S - t_{GM} = \frac{1}{2} d_{adv} + \sigma_{meas}. \tag{13}$$

The worst case would be if $|\sigma_{meas}| = \sigma_{max}$, combining this result with the synchronization condition (2), we can conclude that an attack that introduces an asymmetric delay, no matter how small, can break the synchronization. However, this holds true only as long as the chosen σ_{max} for the network is actually the minimum value possible as expressed in (3). In order to make the network more resilient to possible attacks, we can relax this assumption and a longer value can be chosen. Thus, let σ_{rel} be the maximum allowed offset; the synchronization condition would be

$$\left| t_{S_i} - t_{GM} \right| < \sigma_{rel}. \tag{14}$$

Then we obtain a relation between d_{adv}, σ_{max}, and σ_{rel} that states that, in order to break the time synchronization, an attacker has to introduce a delay twice as long as the difference between σ_{rel} and σ_{max} or, stated differently, a network can tolerate attacks that introduce a delay twice as long as the difference between σ_{rel} and σ_{max} before the time synchronization is broken; thus

$$\left| \frac{1}{2} d_{adv} + \sigma_{max} \right| < \sigma_{rel}. \tag{15}$$

According to the defined adversary model, the adversary strives to keep the network penetration unnoticeable. Furthermore, in some cases, the adversary can succeed in keeping the slave ignorant of the synchronization breaking whenever protection techniques are lacking. The best scenario for the adversary is to break the clock synchronization, while letting the slave think that it still is in a synchronized state. If, as a result of the attack, the slave still thinks that (14) holds for it, but in reality the offset between the grandmaster and the slave is bigger than σ_{rel}, the adversary has succeeded. The advantage, from the adversary point of view, of such an outcome is that the system remains oblivious of its failure. This means that the system will not apply any countermeasures to mitigate the consequences and consequently will not be able to return to a safe state.

6. Potential Solutions and Mitigation Techniques

In this paper we propose the use of a Configuration Agent to detect network penetration via a MIM attack as described above. The idea is that the Configuration Agent will be able to detect the traffic anomalies associated with the attack and use these to diagnose what is happening.

In addition, we have identified some mitigation techniques that can be used alone or in conjunction with the Configuration Agent to strengthen the IEEE 1588 against delay attacks. These mitigation techniques are not enough by themselves to prevent, protect against, or detect an attack, but they can be used to put some boundaries to the damage caused by the attack, thus increasing the resilience of the system.

6.1. Attack Detection. The synchronization messages are sent from the grandmaster with a period equal to the resynchronization interval, R_{int}, and the use of transparent clocks eliminates any possible interference of the rest of the traffic in the network. This means that any variation in R_{int} of the synchronization messages as perceived in the slave could be a hint of something happening in the network.

To detect these anomalies, a Monitor should be placed in the slaves. There the Monitor will be tracking the arrival times of synchronization messages to the slave. The interarrival time between two consecutive messages is

$$\Delta t_i = t_{i+1} - t_i. \tag{16}$$

Although the synchronization messages are sent periodically, some variations of the interarrival time should be expected. Nevertheless, an abrupt and sudden change in the interarrival time could be an indication of an attack happening.

Of course, other subtler, smarter attacks are likely not to be detected just by inspection of the interarrival times. For those, we should use some previous knowledge of the network. Here, we will assume that the Configuration Agent has been active in the network for some time before the attack starts and thus we have a history of the arrival times of synchronization messages to the slave. With a set of n values

we calculate the average of the interarrival time, that would be the resynchronization interval, R_{int}, as perceived by the slave:

$$R_{\text{int}} = \frac{\sum_{i=1}^{n-1} (t_{i+1} - t_i)}{n-1} = \frac{t_n - t_1}{n-1}. \tag{17}$$

The main difference between using the interarrival times or the resynchronization interval as a parameter to detect an attack is that the first is an instantaneous measure that shows right away if something is happening but a smarter attack can go undetected. On the other hand, using the resynchronization interval, it is possible to spot more subtle attacks but some data need to be accumulated before a trend arises.

6.2. Mitigation Techniques

6.2.1. Bounding the Breach. Recall that the IEEE 1588 clock synchronization protocol is based on the exchange of messages between the grandmaster and the slaves. Concretely, in Figure 2, it can be seen how the message `sync` is sent at t_1 from the grandmaster to the slave and, as a response, the message `delay_req` is sent from the slave at t_3, arriving to the grandmaster at t_4. Similarly, the slave is waiting for the response from the grandmaster, the `delay_resp` message that arrives at t_5. These two request-response relations can be used to prevent delay attacks from taking the clock in the slave irreversible away from the grandmaster clock. To do so, we define t_{ret} for the grandmaster and the slaves as the timespan between sending the message and receiving the corresponding response message:

$$\begin{aligned} t_{\text{ret}}^{\text{GM}} &= t_4 - t_1, \\ t_{\text{ret}}^{S} &= t_5 - t_3. \end{aligned} \tag{18}$$

The minimum value for these is the transmission time of the message in the best case, that is, when it does not suffer any intervention in terms of delay attacks, queuing delays, or MAC layer contention. Thus, let n be the number of hops that the message goes through; then the value of t_{ret} can be calculated as

$$\min(t_{\text{ret}}) = \frac{\text{messageLength}}{\text{dataRate}} \times 2n. \tag{19}$$

To calculate the maximum t_{ret}, contention and execution times in the nodes must be taken into account. For the contention we assume that the message can be delayed by at most one message of maximum length in each hop. For the execution time we assume the worst case execution time (t_{WCET}):

$$\max(t_{\text{ret}}) = \min(t_{\text{ret}}) + \frac{\max \text{MessageLength}}{\text{dataRate}} \times 2n + t_{\text{WCET}}. \tag{20}$$

In a small network as the one depicted in Figure 3 with just four hops between the grandmaster and the slave and keeping aside the execution time, the range of t_{ret} is (8, 104) μs (assuming synchronization protocol messages of 126 bytes and a dataRate of 100 Mbps and for the contention using the maximum length for an Ethernet message, 1522 bytes). This means that any value that exceeds that range can imply that the network is under attack. Note, however, that with this method only the delay attacks that introduce a delay longer than $\max(t_{\text{ret}})$ are detected each time. An attack that introduces a delay of, for example, 50 μs will not be detected in a situation of low contention even though it is clear from Section 7.2 that this is a delay large enough to break synchronization. Hence, this method can not be used alone as a detection mechanism, but only as a mitigation technique to prevent the attack from causing excessive clock drifts.

6.2.2. Relaxed Mode. One of the possible network reactions to the detection of an attack is to switch to a relaxed synchronization condition mode. This means that $\sigma_{\max}$ in (2) is increased. This Relaxed Mode leads to degradation of the network service quality but may enable fast network recovery. Obviously, the applicability of such an approach depends on the criticality level of the application and the estimated time needed for recovery. It should be mentioned that the ability of the system to switch to the relaxed synchronization condition mode should be considered already during the system development phase.

6.2.3. Using Environmental Conditions. IEEE 1588 targets industrial applications that imply coping with related environmental conditions (e.g., temperature, humidity). These conditions can influence hardware and particularly the clock crystals. To investigate possible consequences for clock synchronization, the message exchange between a grandmaster GM and a slave S through a set of routers $R_1, \ldots, R_n$ is considered, Figure 8. At each message exchange chain, an error δ, which is caused by hardware time-stamping inaccuracy, is also considered.

For simplicity first we consider a synchronization message exchange without intermediate nodes, routes, as depicted in Figure 7. In this case, in order to take into account additional deviations caused by environmental fluctuations, the following substitutions are required:

$$\begin{aligned} t_1'' &= t_1 + \delta_1, \\ t_2'' &= t_2 + \delta_2, \\ t_3'' &= t_3 + \delta_3, \\ t_4'' &= t_4 + \delta_4. \end{aligned} \tag{21}$$

Then by using (5) we can see that the resulting value of the new offset $\sigma_{\text{meas-envir}}$ can be obtained as

$$\sigma_{\text{meas-envir}} = \sigma_{\text{af}} + \frac{(\delta_2 - \delta_1 + \delta_3 - \delta_4)}{2}. \tag{22}$$

The values constituting the error δ_i can be grouped accordingly by which node they are produced. Actually, errors made by the same node are similar if we consider events occurring close in time, implying that the events of sending

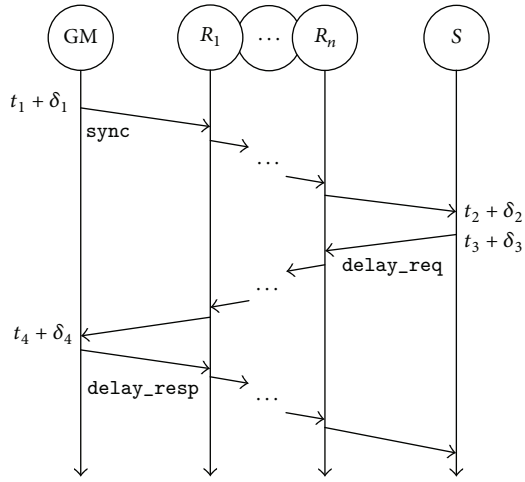


Figure 8: Synchronization protocol considering additional clock drifts introduced by environmental conditions.

the sync and the delay_req messages are likely to be subject to the same environmental delay, and, conversely, the event of receiving the sync and the delay requests messages are likely subject to the same delay, according to

$$\delta_{GM} = \frac{(\delta_1 + \delta_4)}{2}, \qquad \delta_S = \frac{(\delta_2 + \delta_3)}{2}. \tag{23}$$

This yields

$$\sigma_{\text{meas-envir}} = \sigma_{af} + \delta_S - \delta_{GM}. \tag{24}$$

If we add intermediate nodes, routers, to the chain of synchronization messages exchange, their corresponding errors will be included in (24) twice (once per the link they are connected to) but with difference signs. Each intermediate node is an end of the first related link and a beginning of the second related link; that is, as we consider the difference of these errors for each link, they will have different signs in the final expression. Strictly speaking, when we subtract the two errors associated with the same node, the result is not exactly zero, but it is negligibly small. Therefore, when considering the errors caused by environmental conditions only the grandmaster and the slave errors are significant even if intermediate routes are included in the path. This leads to the conclusion that the most significant influence from the environment occurs when the grandmaster and the slave are separated far enough, such that they can have different environmental conditions.

This knowledge can also be used for detecting abnormal delays in the communication that cannot be explained by the environmental conditions alone. If nodes have sensors collecting data about main factors influencing clocks crystals, it is viable to calculate possible clock offset between nodes caused by different environmental conditions. If the observed shift is bigger than what can be expected by environmental conditions, this can indicate the presence of an adversary. Having a clock offset bigger than what was estimated can trigger additional checking of, for example, the links for asymmetry delay detection. Under the assumption of having 5 μs offset with 50 ppm drift, environmental conditions can add 10 ppm to the drift and result in 6 μs offset [19]. This number shows that if the clocks are without temperature compensation, they can affect clock synchronization quite significantly.

6.3. Countermeasures Discussion. When an attack is detected, while the system is in the Relaxed Mode, it should first try to mitigate existing consequences, that is, the synchronization breaching, and second try to prevent the propagation of further consequences. To complete the first goal, related network participants should be informed that there are compromised links. Once the attack is detected, the Monitor could simply indicate between which grandmaster and which slave that there is a breach. The Monitor typically knows the path on which this breach occurred, but it is not known where exactly the adversary is. In the worst case, the whole path from the considered grandmaster to the slave should be eliminated from the clock drift calculations. In scenario *A*, the adversary localization can be made by checking, one by one, all the links in the path under the assumption that there is a technique for checking the suspicious link without letting the adversary know about the check. It can be forged synchronization messages, where delaying would directly reveal the presence of the adversary. It is a challenge, as there are many parameters to consider and assumptions to validate. In scenario *B*, additional measures should be applied for the adversary localization. In this case, the adversary can act on different links in different order. The possibility of this scenario is as high as the first one, as the adversary does not need any additional techniques for switching from scenario *A* to scenario *B*. Such switching will bring only benefits to the adversary, as it increases the chances for a successful attack and a longer undiscovered period which in turn means more serious consequences for the network.

7. Results

7.1. Attack Evaluation. In this subsection, an evaluation of the attack targeting clock synchronization is presented. The attack consists of two phases. The first phase is a MIM attack via ARP poisoning and the second phase is a delay attack. The first phase is evaluated by formal specification of the conducted attack, and the second phase is evaluated by means of logical reasoning in Section 5.3. ARP poisoning is not a newly discovered type of the attack and the evaluation made in this paper is an extension of [4], which only considered scenario *A*. We conduct a formal evaluation of this attack keeping in mind future work where we need a tool for investigating comparison mitigation and prevention techniques.

For the formal attack description and evaluation, the Automated Validation of Internet Security Protocols and Applications (AVISPA) tool was used. This tool is used for protocols analysis from a security point of view. It

has several possible techniques for evaluation of security properties of the considered protocol; they are the On-the-Fly Model Checker (OFMC), the Constraint-Logic-Based Attack Searcher (CL-AtSe), the SAT-Based Model Checker (SATMC), and the Tree Automata Based on Automatic Approximations for the Analysis of Security Protocols (TA4SP).

The OFMC [20] mode was used in the evaluations, as it allows including an adversary in the list of network participants and specifying the goal in the correct way, to check the knowledge of all participants in the end of the message exchange.

AVISPA uses High-Level Protocol Specification Language (HLPSL) to interact with a user, and there is also a possibility to use the Security Protocol Animator (SPAN) [21] tool to simplify this interaction. Working with SPAN, the user needs to specify several categories of protocol components: identifiers, messages, knowledge, and goals. Below, the formalisation of scenario B is considered. Formal analysis of both scenarios was performed, but as scenario B can be represented as an extended scenario A, only the evaluation of scenario B is described here.

Identifiers. Two types of identifiers were used; they are users and numbers (see Table 1). In scenario B (Figure 5) there are 4 users: R_D, R_2, and S are benign network participants and AD_B is an adversary. IP and MAC addresses of these four users are represented as numbers.

Messages. ARP requests and responses are specified through messages. Intruder AD_B sends a message (ARP request) to R_D; this message contains the IP address of R_D and any other IP address of a benign networks participant. Node R_2 answers to S with a message that contains the MAC address of R_2. After such a manipulation, the adversary obtains the MAC address of R_2. In a similar manner, AD_B obtains the MAC addresses of R_D and S; now AD_B can send messages to all of them. In the next step, AD_B sends to R_D a message (ARP reply) containing the IP addresses of R_D and R_2, plus the MAC address of R_2. After this, for R_D, the IP address of R_2 is associated with the MAC address of AD_B. This procedure needs to be repeated for R_2 and S. As a result, R_2 has both the IP addresses R_D and S associated with the MAC address AD_B and S has the IP address R_2 associated with the MAC address AD_B.

Knowledge. Each participant knows its IP and MAC addresses and other benign network participants (see Table 2).

Goals. The goal was specified as keeping the MAC address of R_2 secret from R_D and S. If this condition is fulfilled, it means that the attack was performed successfully, as R_D and S possess only the MAC address of AD_B while they think that they communicate with R_2. Therefore, if the tool shows that the secret holds, this means that the adversary wins.

OFMC analysis showed that, specified in this way, the protocol is safe for the goal of keeping the MAC address of R_2 secret. This means that the described attack scenario is possible and can be performed. The second step is performing the delay attack. After successfully performing the MIM attack, R_D and R_2 communicate through AD_B, and R_2 and S communicate through AD_B. This means that AD_B can delay selective messages in these two communication channels. As it was shown in Section 4 this can lead to clock synchronization breaking.

Table 1: Identifiers.

Type	Identifier
User	R_D, R_2, S, AD_B
Number	IP_{R_D}, IP_{R_2}, IP_S, MAC_{R_D}, MAC_{R_2}, MAC_S, MAC_{AD_B}

Table 2: Knowledge.

User	Knowledge
R_D	R_2, S, IP_{R_D}, MAC_{R_D}
R_2	R_D, S, IP_{R_2}, MAC_{R_2}
S	R_D, R_2, IP_S, MAC_S
AD_B	R_D, R_2, S, IP_{R_D}, IP_{R_2}, IP_S, MAC_S, MAC_{AD_B}

7.2. Simulations with OMNeT++. To evaluate the proposed approach, we have created a network simulation using OMNeT++ [7] together with the INET framework [22]. For the concrete modules needed for the simulation of the clock synchronization protocol, the implementation made by Levesque and Tipper has been used [23]. The goal of these simulations is first to demonstrate that the delay attack can indeed break the clock synchronization. The simulations will also fulfill the role of the Monitor as part of the data that we obtain from them is the same data that a real Monitor would gather in a real network.

Figure 3 shows the topology of the simulated network. The communication starting in the grandmaster GM goes to the slave S through the downstream router R_D. Communication starting in the slave goes to the grandmaster through the upstream router R_U. The resynchronization interval R_{int} is set to 100 ms. We assume that the drift rate of the slave clock is 50 ppm and, therefore, applying (3), $\sigma_{\max} = 10\,\mu$s. And we chose $\sigma_{\text{rel}} = 20\,\mu$s, thus implying that the system has been designed to work properly with this synchronization accuracy. Without loss of generality, just in order to simplify the explanations of the simulations, we assume that the master has a perfect clock; that is, it does not drift. However, the slave is, of course, not aware of this fact.

We simulate the effects of an attack that breaks the time synchronization as shown in Section 5.3. For that we use different models for the delay: a constant delay and a linearly increasing delay. Once an adversary penetrated the network, it can use different techniques for messages delaying. The goal is to investigate different cases going from the simplest one to more complex and try to analyze the differences from the detection point of view.

7.2.1. Constant Delay, $d_{adv} = 50\,\mu$s. In Figure 9 the variations of the difference between the slave clock and the grandmaster clock with time are shown. Before the attack the difference was oscillating between 0 and $-5\,\mu$s, as the result of the clock

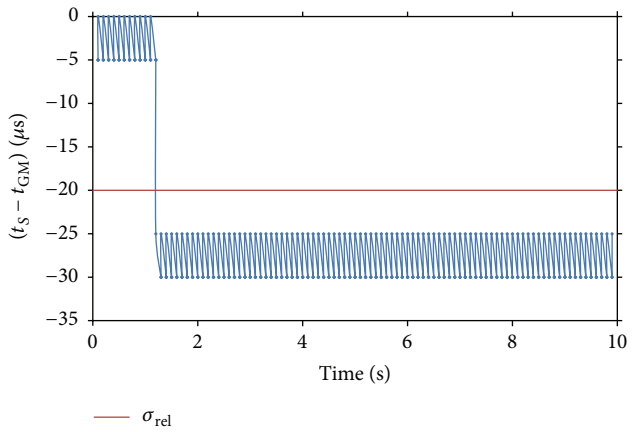


FIGURE 9: Time deviation between the grandmaster clock and the slave clock. Constant delay, $d_{adv} = 50\,\mu$s.

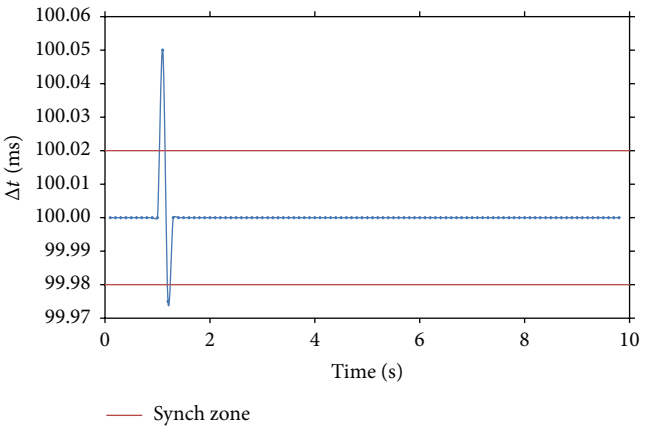


FIGURE 10: Interarrival times of sync messages to the slave. Constant delay, $d_{adv} = 50\,\mu$s.

synchronization protocol performance. After the attack, the difference grows and oscillates between 25 μs and 30 μs. Because these values are bigger than σ_{rel}, we conclude that this attack breaks the clock synchronization. This is the expected result as the value chosen for the delay satisfies (15).

Although Figure 9 is useful to show how the time synchronization is broken, it can be obtained just in the context of this simulations and not in a real-life situation. To detect the attack we must restrain the information used to that available to the slave.

As it was explained in Section 6.1, the Monitor in the slave collects the arrival times of synchronization messages. Figure 10 shows the interarrival times of synchronization messages to the slave as obtained with (16). Before the attack the interarrival times were constantly equal to the resynchronization interval. The peek in that figure is the first message affected by the delay. However, all the following messages are also affected by the delay; we can not see it in the figure because the delay is constant; therefore it only shifts the arrival time of the messages but not the distance between them.

The value of the peak in Figure 10 ($d_{adv} = 50\,\mu$s) is longer than the maximum possible value ($2\sigma_{max} = 20\,\mu$s); therefore, this attack will be easily detected just by analyzing the interarrival times.

After the attack has been detected, some mitigation techniques can be applied. For example, if the system has been designed to function in a Relaxed Mode such as $\sigma_{rel} > 30\,\mu$s then the Configuration Agent can carry out this change of mode. Thus, even though the synchronization is deteriorated the system still behaves in a predictable manner.

7.2.2. Linearly Increasing Delay. We now simulate a delay that increases linearly with every synchronization message that arrives to the router:

$$d(n) = d_{adv} n, \tag{25}$$

where $d_{adv} = 1\,\mu$s and $n = 1, 2, \ldots$ for all messages after the attack starts.

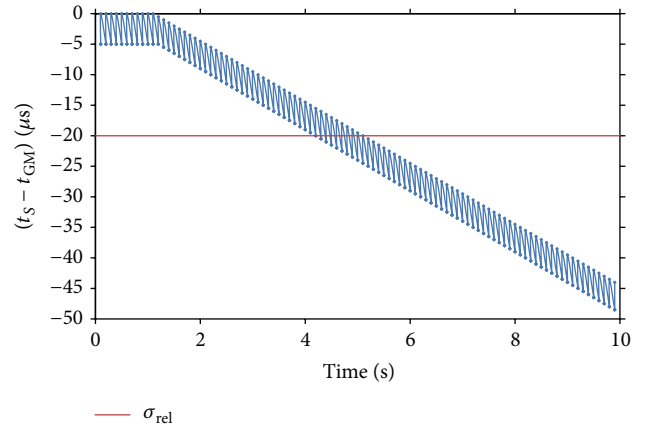


FIGURE 11: Time deviation between the grandmaster clock and the slave clock. Linearly increasing delay.

In Figure 11 it can be seen how the initial delay $d(1) = d_{adv}$ is not big enough to break the synchronization, but after enough resynchronization intervals it does. However, in this case, as compared to the previous one, the attack cannot be detected just by inspecting the interarrival times of synchronization messages to the slave. This can be seen in Figure 12: the effect of the attack on the interarrival times is so small that the slave might as well confuse it with a drifting in the grandmaster clock. This attack puts the slave in a state in which it is not aware of the fact that it is going out of synchronization when, indeed, it is.

For this case we conclude that other parameters, different than the interarrival times, should be used to be able to detect the attack. If we want to keep the detection local to the slave, we can assume that the Monitor has been gathering data for some time before the attack happens and use those values to obtain statistical parameters.

For example, we can examine the variations of the calculated value of the resynchronization interval obtained using (17). In Figure 13 it is shown how this value is consistently increasing. Therefore, in order to detect the attack, we need to be able to identify this kind of patterns. Because the increase

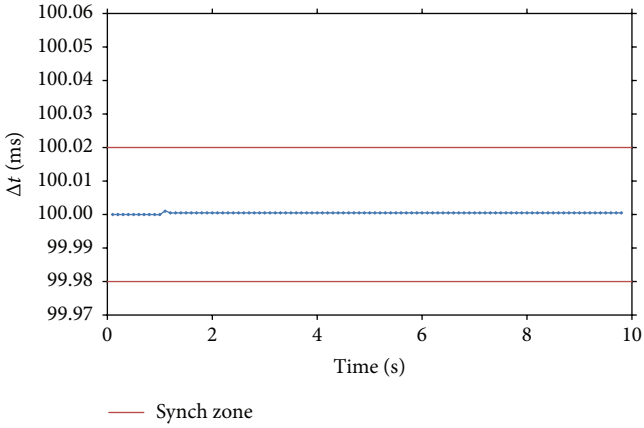


Figure 12: Interarrival times of sync messages to the slave. Linearly increasing delay.

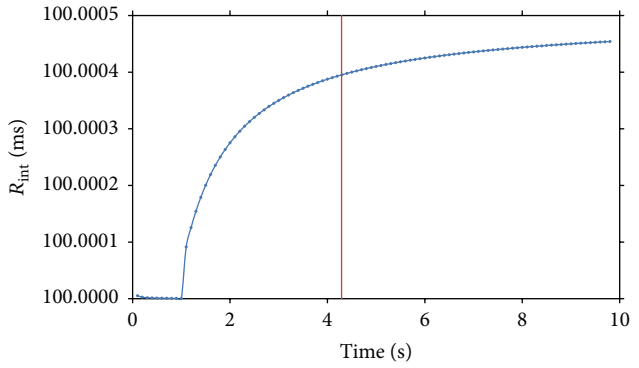


Figure 13: Resynchronization interval as obtained in the slave. Linearly increasing delay.

in the value is so small, we could probably not use it as a sole criterion to detect an attack, but it can be one of a multicriteria detection method. A further study could be done comparing the amount of resynchronization intervals that the attack will need to break the time synchronization with the number of resynchronization intervals that the Configuration Agent needs to detect the attack.

Independently of the detection capabilities, here we see again how choosing a σ_{rel} longer than σ_{max} gives the network some time to react to the attack even before the synchronization has been broken.

We showed the inner difficulty of attack detection only by means of local monitoring, especially in the case of a smarter attack that introduces a linearly increasing delay. Our proposal is to use mitigation techniques, for example, like the ones presented in Section 6.2, to cope with attacks that cannot be detected by distributed monitoring in nodes. If we apply the mitigation technique described in Section 6.2.1 to the topology used in the simulations, we see that in the worst case scenario (i.e., low contention in the network) this method is able to detect attacks introducing delays longer than 100 μs. This value is way above the minimum delay required to break the synchronization and, therefore, the technique cannot be used to prevent time synchronization breaching. Nevertheless, it has two important benefits in the case of the linearly increasing delay. First, it actually allows the slave to detect an attack that is not possible to detect with the monitoring approach. And secondly, by including the knowledge of this upper limit for the delay in the design, the system can be prepared for this scenario, that is, has an approach for returning into safe mode.

8. Conclusions and Future Work

In this paper, possible strategies of breaching clock synchronization together with techniques on how to detect it were investigated. First, the possibility of conducting the proposed attack breaching clock synchronization was proven by evaluation in AVISPA and through logical reasoning. This conclusion demonstrates the necessity to provide industrial networks with appropriate protection measures. Next a traffic monitoring approach was proposed as a way of detecting the delay attack. Simulation of the Monitor approach showed the possibility to detect the delay attack in case of the adversary imposing a constant delay. The simulation results also showed that if the system is designed with a relaxed synchronization condition mode, it can help mitigating the consequences of a delay attack once it has been detected. The efforts required for localization of the adversary depends on the way of performing the ARP poisoning attack, for example, in the scenario when an adversary takes control over several communication channels, it is more challenging to define links that are compromised. Furthermore, the result also demonstrates that in the case of introducing a linearly increasing delay, the adversary influence can remain undetected. Therefore, more sophisticated detecting techniques are needed to detect such attacks. Algorithms for growing trend detection can then be a possible solution for coping with nonconstant delays.

Clock synchronization is an essential part of all networks with real-time requirements. Basically all industrial networks have real-time requirements, and thus if there is a way to breach clock synchronization, the method will disrupt the network functionality and moreover it is applicable for a range of use cases. IEEE 1588 that is typically used to provide clock synchronization lacks security mechanisms. Not even Annex K, which has been introduced to enhance security, is capable of handling delay attacks such as the ones evaluated in this paper. However, there is nothing in IEEE 1588 which prevents using a Monitor and thus our proposed solution can easily be used to enhance the standard.

There is a high potential for future work in this area. We plan to consider more diverse attack scenarios, which include compromised devices and cases with clock acceleration and deceleration. Further, different detection and mitigation strategies, such as distributed monitoring to help locating the adversary together with algorithms for trend detection, are to be considered. Furthermore, we want to add learning and adaptation abilities of an adversary and of the Monitor to analyze their interactions.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

The research leading to these results has received funding from the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme FP7/2007–2013/under REA grant agreement 607727.

References

[1] D. Dzung, M. Naedele, T. P. von Hoff, and M. Crevatin, "Security for industrial communication systems," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1152–1177, 2005.

[2] E. Lisova, E. Uhlemann, J. Akerberg, and M. Bojrkman, "Towards secure wireless TTEthernet for industrial process automation applications," in *Proceedings of the IEEE Emerging Technology and Factory Automation (ETFA '14)*, pp. 1–4, Barcelona, Spain, September 2014.

[3] E. Lisova, E. Uhlemann, W. Steiner, J. Akerberg, and M. Bjorkman, "A survey of security frameworks suitable for distributed control systems," in *Proceedings of the International Conference on Computing and Network Communications (CoCoNet '15)*, pp. 205–211, Trivandrum, India, December 2015.

[4] E. Lisova, E. Uhlemann, W. Steiner, and J. Åkerberg, "Risk evaluation of an ARP poisoning attack on clock synchronization for industrial applications," in *Proceedings of the IEEE International Conference on Industrial Technology (ICIT '16)*, Taipei, Taiwan, 2016.

[5] B. Mukherjee, L. T. Heberlein, and K. N. Levitt, "Network intrusion detection," *IEEE Network*, vol. 8, no. 3, pp. 26–41, 1994.

[6] A. Armando, D. Basin, Y. Boichut et al., "The AVISPA tool for the automated validation of internet security protocols and applications," in *Computer Aided Verification*, K. Etessami and S. K. Rajamani, Eds., vol. 3576 of *Lecture Notes in Computer Science*, pp. 281–285, Springer, Berlin, Germany, 2005.

[7] OMNeT++, January 2015, http://www.omnetpp.org/.

[8] M. Ullmann and M. Vogeler, "Delay attacks—implication on NTP and PTP time synchronization," in *Proceedings of the International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS '09)*, pp. 1–6, October 2009.

[9] B. Hirschler and A. Treytl, "Validation and verification of IEEE 1588 annex K," in *Proceedings of the International IEEE Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS '11)*, pp. 44–49, Munich, Germany, September 2011.

[10] T. Mizrahi, "A game theoretic analysis of delay attacks against time synchronization protocols," in *Proceedings of the International IEEE Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS '12)*, pp. 1–6, San Francisco, Calif, USA, September 2012.

[11] M. Gutiérrez, W. Steiner, R. Dobrin, and S. Punnekkat, "A configuration agent based on the time-triggered paradigm for real-time networks," in *Proceedings of the IEEE World Conference on Factory Communication Systems (WFCS '15)*, pp. 1–4, IEEE, Palma, Majorca, Spain, May 2015.

[12] M. Gutiérrez, W. Steiner, R. Dobrin, and S. Punnekkat, "Learning the parameters of periodic traffic based on network measurements," in *Proceedings of the IEEE International Workshop on Measurements & Networking (M&N '15)*, pp. 1–6, October 2015.

[13] L. Heberlein, G. Dias, K. Levitt, B. Mukherjee, J. Wood, and D. Wolber, "A network security monitor," in *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 296–304, IEEE, Oakland, Calif, USA, May 1990.

[14] M. Eslahi, M. S. Rohmad, H. Nilsaz, M. V. Naseri, N. Tahir, and H. Hashim, "Periodicity classification of HTTP traffic to detect HTTP Botnets," in *Proceedings of the IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE '15)*, pp. 119–123, Langkawi, Malaysia, April 2015.

[15] M. Garuba, C. Liu, and D. Fraites, "Intrusion techniques: comparative study of network intrusion detection systems," in *Proceedings of the 5th International Conference on Information Technology: New Generations (ITNG '08)*, pp. 592–598, Las Vegas, Nev, USA, April 2008.

[16] G. Fink, B. Chappell, T. Turner, and K. O'Donoghue, "A metricsbased approach to intrusion detection system evaluation for distributed real-time systems," in *Proceedings 16th the International Parallel and Distributed Processing Symposium (IPDPS '02)*, Abstracts and CD-ROM, 8 pages, Ft. Lauderdale, Fla, USA, April 2002.

[17] IEEE, "IEEE standard for a precision clock synchronization protocol for networked measurement and control systems," IEEE Std 1588-2008, Revision of IEEE Std 1588-2002, 2008.

[18] B. Kang, P. Maynard, K. McLaughlin et al., "Investigating cyber-physical attacks against IEC 61850 photovoltaic inverter installations," in *Proceedings of the IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA '15)*, pp. 1–8, Luxembourg City, Luxembourg, 2015.

[19] TCXO, Temperature Compensated Crystal Oscillator, http://www.radio-electronics.com/info/data/crystals/tcxo.php.

[20] D. Basin, S. Modersheim, and L. Vigano, "OFMC: a symbolic model checker for security protocols," *International Journal of Information Security*, vol. 4, no. 3, pp. 181–208, 2005.

[21] Y. Glouche, T. Genet, O. Heen, and O. Courtay, "A security protocol animator tool for AVISPA," in *Proceedings of the ARTIST2 Workshop on Security Specification and Verification of Embedded Systems*, Pisa, Italy, May 2006.

[22] INET Framework, June 2015, https://inet.omnetpp.org/.

[23] M. Levesque and D. Tipper, "ptp++: a precision time protocol simulation model for OMNeT++/INET," http://arxiv.org/abs/1509.03169.

Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2016, Article ID 5212314, 13 pages
http://dx.doi.org/10.1155/2016/5212314

Research Article

Strengthening MT6D Defenses with LXC-Based Honeypot Capabilities

Dileep Basam, J. Scot Ransbottom, Randy Marchany, and Joseph G. Tront

Bradley Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA 24061, USA

Correspondence should be addressed to Dileep Basam; dileepba@vt.edu

Received 5 November 2015; Revised 8 February 2016; Accepted 6 March 2016

Academic Editor: Elias P. Duarte

Moving Target IPv6 Defense (MT6D) imparts radio-frequency hopping behavior to IPv6 networks by having participating nodes periodically hop onto new addresses while giving up old addresses. Our previous research efforts implemented a solution to identify and acquire these old addresses that are being discarded by MT6D hosts on a local network besides being able to monitor and visualize the incoming traffic on these addresses. This was essentially equivalent to forming a darknet out of the discarded MT6D addresses, but the solution presented in the previous research effort did not include database integration for it to scale and be extended. This paper presents a solution with a new architecture that not only extends the previous solution in terms of automation and database integration but also demonstrates the ability to deploy a honeypot on a virtual LXC (Linux Container) on-demand based on any interesting traffic pattern observed on a discarded address. The proposed architecture also allows an MT6D host to query the solution database for network activity on its relinquished addresses as a JavaScript Object Notation (JSON) object. This allows an MT6D host to identify suspicious activity on its discarded addresses and strengthen the MT6D scheme parameters accordingly. We have built a proof-of-concept for the proposed solution and analyzed the solution's feasibility and scalability.

1. Introduction

Nodes participating in MT6D periodically relinquish IP addresses and hop on to new addresses. This address-hopping trait of MT6D improves the security and privacy of IPv6 networks; however, there is no feedback mechanism in the current implementation; that is, a node engaged in a MT6D conversation has no means of realizing if there is an attacker uncovering its MT6D addresses and trailing the MT6D node along its address hops.

(i) Challenge. In this work, we propose a mechanism to use relinquished MT6D addresses in order to discover any anomalies and gather intelligence on attacker methods. We also build a proof-of-concept solution to impart darknet and honeypot capabilities to MT6D scheme.

(ii) Proposal. Our approach to solve this problem is to design a solution that identifies and acquires the addresses discarded by the MT6D hosts and enumerating incoming traffic on these relinquished addresses in addition to providing the ability to deploy a virtual container-based honeypot configured with a specific discarded address upon detecting a suspicious traffic pattern. Our previous effort [1] included building a Central Node (CN) that passively listens to local link traffic in promiscuous mode, identifying and acquiring discarded addresses by MT6D nodes and performing traffic enumeration on these addresses. The CN uses typical IPv6 neighbor discovery protocol (NDP) messages like Neighbor Solicit (NS) and Multicast Listener Discovery (MLD) messages to reconstruct who is relinquishing which addresses and when to acquire them.

The solution's ability to deploy a honeypot tied to a discarded MT6D address of interest can potentially take up the conversation forward with the attacker to gather intelligence on attacker methods besides collecting attack traffic samples for further analysis. The solution also allows an MT6D node to query the database for incoming traffic on its discarded addresses, thus giving the MT6D node the ability to analyze the activity to identify malicious traffic or an attacker persistently trailing on its discarded addresses, to change

the scheme parameters, for example, move to a stronger secret-key or a faster hopping interval to evade the attacker.

(iii) Threat Model and Assumptions. We assume a threat model involving an attacker with infinite resources (compute cycles and time), with an ability to brute-force the scheme parameters, for example, the secret key involved in address computation (hashing scheme), to discover the MT6D addresses. But the attacker validating these uncovered MT6D addresses will generate some traffic, leaving a trail of his/her activity on these discarded MT6D addresses.

(iv) Goals. In this paper, we seek to impart honeypot capabilities to Moving Target IPv6 Defense (MT6D) in order to provide a feedback mechanism for evaluating the strengths of MT6D scheme by analyzing the activity on discarded MT6D addresses. Among the goals of this effort, the first goal is to implement the database integration into CN along with developing a way to build web server to offer real-time visualization of the local MT6D addresses that are being relinquished and incoming traffic on these discarded addresses. Our next goal is to implement a capability of deploying honeypot service on a virtual container on-demand bound to a discarded MT6D address of interest with minimal interruption observable to an attacker. Lastly, we want to analyze the feasibility and scalability of the proposed solution by analyzing the CPU and memory overhead trends while scaling to multiple honeypot containers.

The paper is organized as follows. First, we provide background on Moving Target IPv6 Defense (MT6D), Linux Container (LXC), Honeypots, Dionaea, and various components used in building the solution in Section 2. Related work on virtual honeypot based solutions and adaptive/dynamic honeypot architectures is discussed in Section 3. Section 4 explains our idea and high level approach. Section 5 describes the solution testbed and implementation details. In Section 6, we present our results, visualizations, and our observations. Sections 7 and 8 present future work and conclusion.

2. Background

In this section, we present background, starting with IPv6 followed by Moving Target IPv6 Defense (MT6D), Dionaea, LXC, and other components used in building the solution.

2.1. IPv6. On September 24, 2015, the American Registry for Internet Numbers (ARIN) announced that it allocated the final IPv4 address blocks in its free pool and the IPv4 address space is completely exhausted. The IPv6 protocol by design overcomes the address exhaustion limitation with its address length of 128 bits. This new address size allows for 2^{128} possible addresses, approximately 7.92×10^{28} addresses for every possible IPv4 address.

2.2. Moving Target IPv6 Defense (MT6D). The MT6D [2, 3] involves nodes that change network addresses while maintaining active sessions. Leveraging the large address space in IPv6, the MT6D protocol changes IP addresses for communicating hosts in synchronization. This achieves an effect similar to frequency hopping in radio networks, for an attacker passively listening to the conversation of two MT6D hosts will see multiple hosts communicating with each other rather than a pair of hosts. MT6D achieves synchronization between the involved nodes by having both ends of a communication pair compute both their own IP address and their remote-peer's IP address during the particular time window "t". The scheme involves computation of the interface-identifier part of the address, that is, IID$'$ extracted from first 64 bits from a hash digest (H) of concatenated string made of a symmetric Secret-Key (SK), initial IID, and current time window "t":

$$\text{IID}' = H\left[\text{IID\_initial} \parallel \text{SK} \parallel t\right]_{0 \to 63}. \quad (1)$$

MT6D mechanism offers privacy and anonymity by hiding the original network layer addresses of involved IPv6 nodes with dynamically generated addresses besides protecting against intrusion-based network attacks. MT6D protects against address tracking and traffic correlation, thus helping hosts keep their network identities private while conducting sensitive communications.

2.3. Dionaea and DionaeaFR. Dionaea is a low-interaction honeypot offering IPv6 support with an ability to capture malware in addition to logging suspicious traffic. Dionaea offers flexibility to configure services and interfaces to listen on. Dionaea uses Sqlite as the backend database allowing custom queries to be made to extract interesting information from logs stored in the database [4].

DionaeaFR is an open-source library that offers front-end visualization of Dionaea's logs that are stored in the Sqlite database. DionaeaFR's web server is implemented in Python and Django framework. This front-end console allows retrieving statistics of traffic hitting the honeypot in addition to offering a way to download attack traffic samples, for example, malware [5].

2.4. LXC. LXC is a light-weight virtualization technology that relies on isolated user spaces to create virtual containers that share same kernel. LXC uses Linux kernel features like namespaces, Chroots, CGroups, and so forth, to isolate the container processes. LXC differs from the concept of standard virtual machines (VM) by sharing the same kernel and underlying hardware which obviates the need for a hypervisor [6].

2.5. Other Components Used in Building the Solution

2.5.1. Scapy, Pcapy, Impacket, Pyroute2, and Wireshark. Scapy is a Python module for packet crafting and manipulation tool for computer networks [7]. Pcapy and Impacket are python libraries for capturing and decoding raw network traffic. Pyroute2 is a network configuration library for binding IP addresses to a Linux host. Wireshark is a popular packet capture and analysis tool [8].

2.5.2. MongoDB and PyMongo. MongoDB is a NoSQL database that stores each record in the database as a JSON object and PyMongo is a Python library that acts as an application programming interface (API) to interact with MongoDB from Python code.

2.5.3. D3.js. D3.js is a JavaScript library that leverages HTML, SVG, and CSS to produce complex visualizations on a web browser [9].

2.5.4. Dstat. Dstat is a free tool that combines vmstat, iostat, netstat, and ifstat to view system resources in real-time. The tool is used in this work to collect real-time CPU (usr/sys) and free-memory (RAM) statistics during the experiments [10].

3. Related Work

A honeypot is a security resource that delivers insights by getting probed, attacked, or compromised by malicious entities [11] and subsequently reports the characteristics of the attack. Honeypots are classified as either low or high interaction based on their designed level of interaction with the potential attacker. Low-interaction honeypots, as their name suggests, are often limited by the degree of interaction. Examples of low interaction honeypots include Honeyd and Dionaea. On the other hand, high-interaction honeypots support complex behavior by emulating a real operating system with a full suite of applications. High-interaction honeypots carry more risk by allowing full compromise by the attacker. Honeypots have no legitimate production use for incoming traffic, so in theory any traffic hitting a honeypot can be deemed suspicious and warrant inspection.

On the other hand, a Honeynet is a basic network of commonly used operating systems in their default configurations. As a Honeynet does not broadcast its identity, all incoming traffic is deemed illegitimate and further investigated. Kuwatly et al.'s work [12] discusses a dynamic honeypot solution built from the fingerprinting tool (p0f) and Nmap to dynamically configure and leverage Honeyd-based emulated virtual hosts and a physical high-interaction honeypot cluster to capture and analyze attacker traffic. Hecker et al.'s paper [13] proposes a solution that uses nmap for fingerprinting the local network (topology, hosts, ports, etc.) and Honeyd-configuration manager for dynamically building honeypots.

Kishimoto et al.'s work [14] on dynamic honeypot commissioning involves detection of incoming address scans targeting an unallocated IP address. Research work done by [14] on commissioning honeypots based on incoming address scans shows that disabling Duplicate Address Detection (DAD) makes the address acquisition faster. During the prework tests, we analyzed the relevant network captures and found that we can significantly reduce the delay further by enabling a node to proactively claim ownership of the acquired address without waiting for NS messages from router. More details are presented in Section 4. Hieb and Graham [15] employ dynamic honeypots and monitoring network activity of deployed honeypots to set up anomaly-based intrusion detection for the network. Brzeczko's work [16] on Turnkey Honeynet framework involved automatic commissioning of Honeypots (Dionaea, Kippo, and Glastopf) depending on the composition of live attack traffic.

Memari et al. [17] built a virtual Honeynet and compared LXC virtualization with other virtualization methods including VMware, VirtualBox, and KVM and concluded that the LXC approach provides better performance. Work by Sokol and Pisarcik [18] on Distributed Virtual Honeynets framework talks about a master control center and a network of high-interaction virtual Honeynets based on OpenVZ and LXC virtualization.

Previous research work in this area involved creating Darknets, Network Telescopes, and Honeynets, with no advertised services and listen for illegitimate traffic. There are no current implementations adapting such a scheme to moving target defense schemes like MT6D. Also unlike a static darknet that passively listens on unallocated address space, our solution is dynamic as it learns IP-address and MAC address associations from live network traffic to actively acquire the addresses being purged in order to gather intelligence on attacker methods. Work presented in this paper extends the related work discussed above in terms of a full IPv6 implementation framework of on-demand honeypot commissioning using Linux containers (LXC) in addition to proposing a method to leverage such a solution to fortify MT6D defenses. This solution presents a simplified visualization of attacker traffic by ignoring the MT6D nodes that do not have interesting incoming traffic to enable an uncluttered view of the suspicious traffic. This work also presents a memory and CPU overhead analyses of the proposed solution while scaling to multiple honeypot containers.

4. Idea and Approach

The central idea of this research effort is to devise a mechanism to gather intelligence on attacker methods by watching for suspicious traffic on relinquished MT6D addresses in addition to be able to deploy a honeypot on a specific discarded address upon identifying a suspicious traffic pattern.

To implement such a scheme, as shown in Figure 1, we have a MT6D-host that periodically hops onto new addresses while relinquishing old addresses. A CN passively listens to the network traffic in promiscuous mode to parse the NS and MLD messages from MT6D hosts to populate a database collection of who is relinquishing what address, identifying the right time-instant to acquire these discarded addresses and subsequently binding these addresses to its local network interface. After the CN binds these discarded addresses to its network interface, it analyzes the incoming traffic on these discarded addresses and may place a request to the honeypot-host for a honeypot container to be deployed on a specific IPv6 address. The challenge is to migrate the IP address from the CN to virtual LXC container on the honeypot-host (honeypot container) with minimal interruption observable to an external attacker who is sending the traffic to the discarded address. The steps involved in this process are unbinding a specific IP address on the CN, binding this address on honeypot container, and finally invoking and

binding Dionaea and DionaeaFR services to this IP address on the honeypot container.

We propose an efficient and quick approach for honeypot deployment by maintaining hot spares of honeypot containers. To facilitate the hot spares approach, we packaged an LXC container with all prerequisites such as Dionaea, DionaeaFR, and supporting python libraries to act as a honeypot LXC-template. The honeypot-commissioning-module on honeypot-host clones a fixed number of containers from the honeypot LXC-template in advance and maintains a list of available containers. By keeping honeypot containers waiting in a queue, deploying a honeypot configured with a target IPv6 address requires that the only configuration needed on the container-side is binding the IP address to the network interface of an available container and starting Dionaea honeypot and DionaeaFR (management web server) services. This minimal configuration requirement ensures minimal delay in bringing up honeypot service bound to the desired IPv6 address.

In order to minimize the interruption window observable by an attacker during the migration of IP address from CN to honeypot container, we followed the approach of holding the address for a fixed-period after receiving the message that a honeypot is being deployed from the honeypot-host. For a smoother IP transition from CN to the honeypot container, based on our initial tests, we implemented the heuristic of CN holding the address for one second. This address-holding-period compensates for the time it takes the IP address to be bound on container's network interface and to start Dionaea and DionaeaFR services on the honeypot container.

We went with the approach of using the honeypot-host instead of running a honeypot service on the CN itself for two reasons. First, running honeypot on the CN may not be a secure approach as the CN not only maintains a database of the discarded addresses but also actively collects the addresses that are currently active on MT6D hosts from the parsing of NS and MLD messages. Secondly, a design including a dedicated honeypot host with its own database allows flexibility for such a honeypot-host running virtual machines with vulnerable-services to sit in a segregated zone like a DMZ in the future while its partner CN can still be on the local-network with MT6D nodes.

Even though we used a low-interaction honeypot, Dionaea, for experiments, by the virtue of using LXC containers in our solution, the solution allows high-interaction honeypot deployment on the container. So we are not limited to emulated services or virtual hosts in future experiments.

5. Testbed and Implementation

The test setup as shown in Figure 1 involves three Linux desktops running the Ubuntu 12.04 operating system connected to a layer-2 switch with an uplink to the university network router. Virginia Tech has fully operational production IPv6 network. The MT6D host periodically acquires new addresses while relinquishing old addresses. The CN identifies and acquires the discarded addresses on the local network and is also responsible for performing traffic enumeration, analysis,

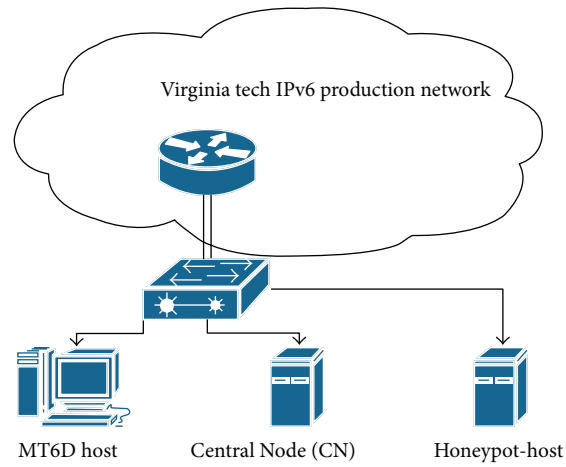


Figure 1: Block diagram of test setup.

and visualization. Third Linux machine is the honeypot-host system that is installed with LXC software to support on-demand creation of containers to act as individual honeypots. To set context for the memory and CPU overhead analyses in Section 6, the honeypot-host system is a Dell Optiplex Desktop running 64-bit 12.04 Ubuntu operating system installed with Intel Core-2 6700 2.66 GHz CPU, 4 GB RAM, and 80 GB of secondary memory.

Figure 2 depicts the architecture of our solution. The solution consists of CN block integrated with the database (MongoDB) and web server (Node.js) modules for inter-module communication and real-time visualizations. The honeypot-host block is implemented with on-demand honeypot deployment capability integrated with its dedicated (MongoDB) database.

The implementation involves building honeypot-host block and implementing database integration for CN and honeypot-host modules. The CN includes traffic-parsing, address-acquisition, enumeration, analysis, and visualization modules. The CN uses Pcapy and Impacket libraries in its traffic-parsing module to listen to neighbor discovery (NS and MLD messages) conversations of all MT6D nodes on local link to learn the addresses that are being relinquished by the MT6D nodes and stores these associations in a database. It employs the Pyroute library in address-acquisition module to acquire and bind these learned addresses to local host and Scapy to send out a forced NA claiming ownership of the bound address as an unicast to the local-router. The CN's traffic-enumeration module uses the same Pcapy and Impacket libraries to parse the incoming traffic on these acquired addresses and does traffic enumeration (Source IP address, Destination Port, and Packet Count), storing this data in the form of a native python dictionary. These python-dictionaries holding enumerated traffic data get converted to JSON objects for facilitating the visualizations. A node.js web server has been built on the CN to offer visualizations based on real-time traffic (MT6D addresses that are being spawned and incoming traffic on acquired addresses). The visualized data offers two views, MT6D_view and attacker_view (Figures

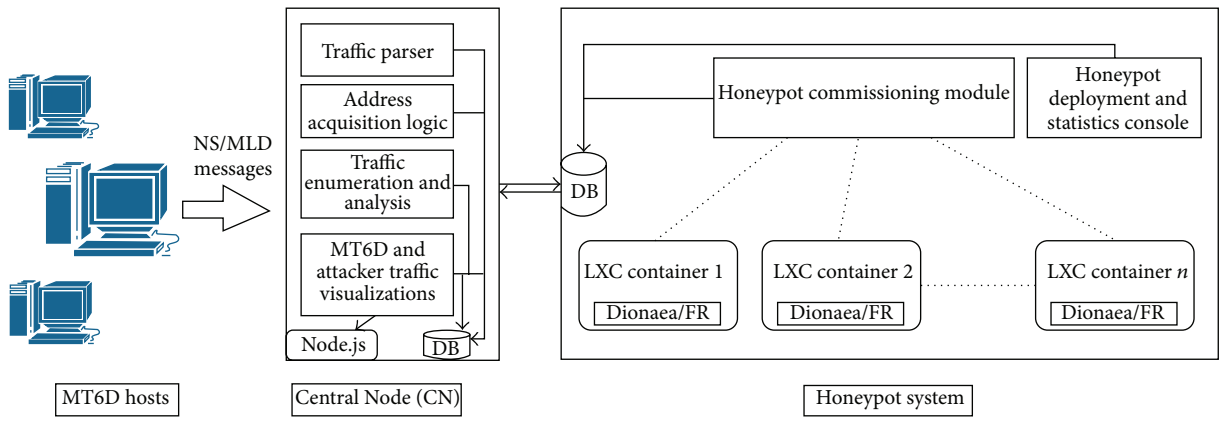


Figure 2: Solution architecture.

4, 7, and 8), allowing an administrator to see whether an attacker is able to trail the MT6D node along the spawned addresses.

The honeypot-host block includes the honeypot-commissioning module, the LXC, and a database. The honeypot-host database hosts two collections, hpot_requested_collection and hpot_deployed_collections, for exchanging information on addresses on which the honeypot is being requested or deployed. Figure 3 depicts the flow-diagram that explains the sequence of messages exchanged between the blocks and actions performed during the operation. Step 1 involves the CN listening to and parsing local link NS/MLD messages to keep a list of MT6D addresses being relinquished on the local network. In Step 2, upon seeing an MT6D node relinquish an address, the CN assigns the address to itself to facilitate traffic enumeration on this address. In Step 3, we have interesting incoming traffic hitting a discarded MT6D address (IPv6-address-of-interest) that is currently bound to the CN. Step 4 involves the CN requesting that a honeypot be deployed on a specific IPv6 address by writing the IPv6-address-of-interest to hpot_requested_collection on honeypot-host's database. In Step 5, the honeypot-host periodically checks hpot_requested_collection, retrieves the IPv6-address-of-interest, and acknowledges the CN by writing the IP address to hpot_deployed_collection in honeypot-host database.

In Steps 6 and 7, the CN continually checks for records in hpot_deployed_collection and upon finding a record the CN waits for a fixed time (the address-holding-period) and unbinds the address retrieved from hpot_deployed_collection's record. In the mean-time, as shown in Step 8, the honeypot-host checks for an available spare LXC container in the queue and invokes a shell-script to run lxc-attach commands to bind the IP-address on which honeypot is requested to the container's network interface, start the Dionaea service to bind the honeypot service to the newly assigned IPv6 address, and run the python web-server for launching the DionaeaFR front-end console. To speed up the address binding process, as specified in previous research effort [1], we disable Duplicate-Address-Detection (DAD) and use the concept of a forced Neighbor Advertisement (NA) thus advertising the new IPv6 address with the container's mac-address to the local router for quick address-acquisition. This completes the deployment of a honeypot tied to a desired IPv6 address and all future attacker traffic hits the honeypot container.

The MT6D_view visualizations (Figures 4 and 7) show each MT6D host represented by its mac address and all the addresses relinquished by each MT6D host and incoming traffic on each of these spawned addresses by source IP address and Protocol. To simplify visualization of incoming traffic without getting cluttered by new addresses that are being discarded by MT6D nodes and to observe the incoming traffic from the attacker perspective, the attacker_view visualization (as shown in Figure 8) view would show Source IP (potential attacker) of incoming traffic at its root and the visualization would then branch from each of these source-IPs of incoming traffic to a MT6D mac address and then branching to MT6D IPv6 addresses and incoming traffic composition in terms of protocol and port numbers and corresponding packet-counts.

In this chapter, we have discussed how we implemented various components of the proposed solution and explained how information flows between different modules. The next section presents evaluation of the solution in terms of our observations and results.

6. Results

Our solution at its heart involves efficiently spinning up containers on the honeypot-host and migrating the IP addresses from the CN to honeypot containers. To characterize such a solution, we will evaluate it in terms of interruption window observable by an attacker, as well as CPU and memory overheads. The interruption window observable to an attacker is a critical factor for judging the feasibility of the solution, since the duration of interruption window may offer a cue to the attacker about migration of his/her traffic from one machine (the victim) to another (the honeypot). CPU and memory overhead analyses offer valuable insights into scalability of the solution, that is, the number of honeypot containers that can be accommodated for a specific honeypot-host system

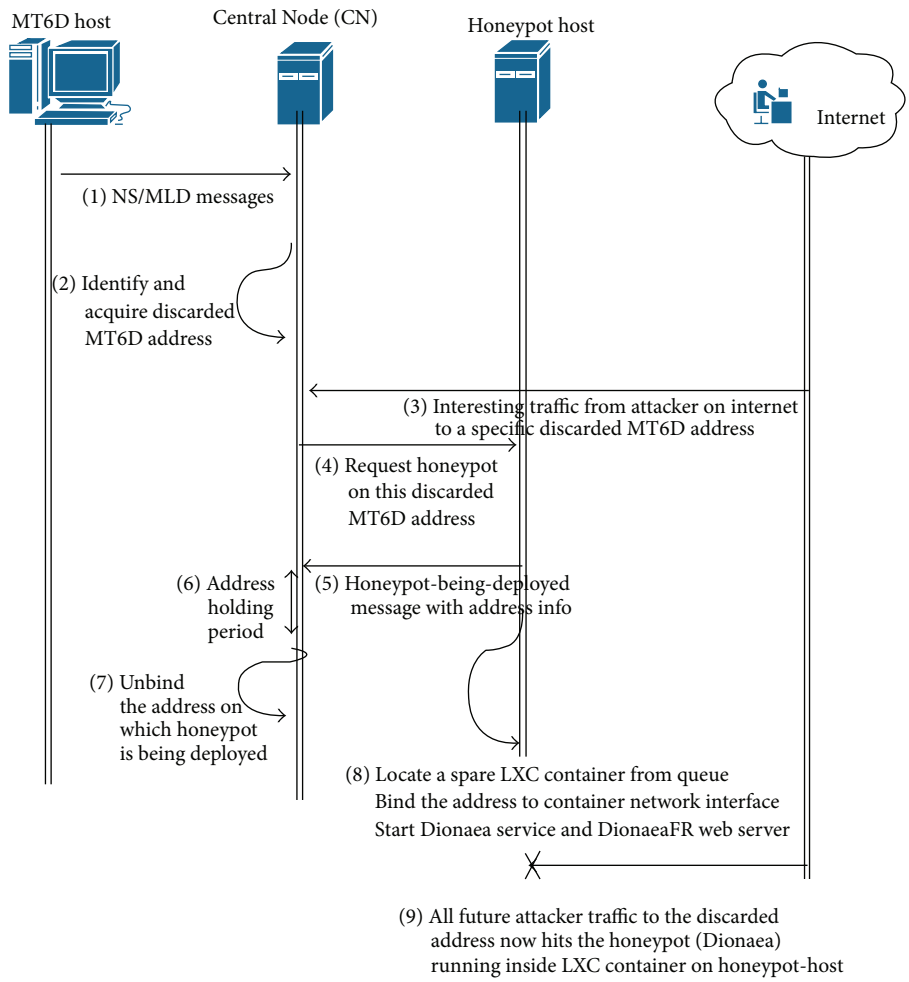


Figure 3: Flow diagram: sequence of messages exchanged between various components.

(CPU and memory) configuration. We will also discuss a mechanism for MT6D nodes to leverage our solution. In this section, we will explain the methodology of the testing and present our results.

6.1. Experiments and Results

6.1.1. Test Scenario. We first present a particular instance of potential attack-traffic hitting a discarded MT6D address and launch a honeypot-service bound to the specific address on an LXC container and supporting traffic visualizations. We then present results detailing the address-migration window times in terms of hold-packet-count and lost-packet-count for a ten-honeypot container deployment trial. We also present our observations from CPU and memory overhead analyses for single, five, and ten-honeypot container deployment trials. Hold-packet-count refers to the number of packets still hitting the CN since the trigger that deploys the honeypot, that is, the first ICMP probe hitting the CN. Lost-Packet-Count refers to the number of packets lost as seen by the attacker during the migration of an IP address to the honeypot container from the CN.

The experiment starts with an MT6D host with mac address 00:24:e8:42:c4:7a spinning off new addresses while dropping its old addresses. The CN listens to MT6D host's NS and MLD messages and parses them to identify and acquire the addresses that are being discarded by the MT6D host and visualizes the incoming traffic on these discarded addresses. Figure 4 shows the first level of a MT6D_view visualization with the MT6D host represented by its mac address, 00:24:e8:42:c4:7a, at the center, with each newly generated MT6D address (e.g., 2001:468:c80:c111:8c23:9665:ae9a:c757) represented by the last 64 bytes (e.g., 8c23:9665:ae9a:c757). The /64 subnet prefix remains the same, that is, 2001:468:0c80:c111 for all the MT6D address children nodes.

The next step involves sending ICMP echo probes (simulated attack traffic) every 50 ms from the machine (at 2601:5c0:c000:773e:1565:8df2:1408:2077) on a remote Internet connection to a particular discarded MT6D address, 2001:468:c80:c111:8c23:9665:ae9a:c757 that is now assigned to the CN. The first ICMP echo request packet serves as a trigger (as per configuration in the traffic enumeration and analysis module) for the CN to flag this activity and request a honeypot be deployed on this specific address.

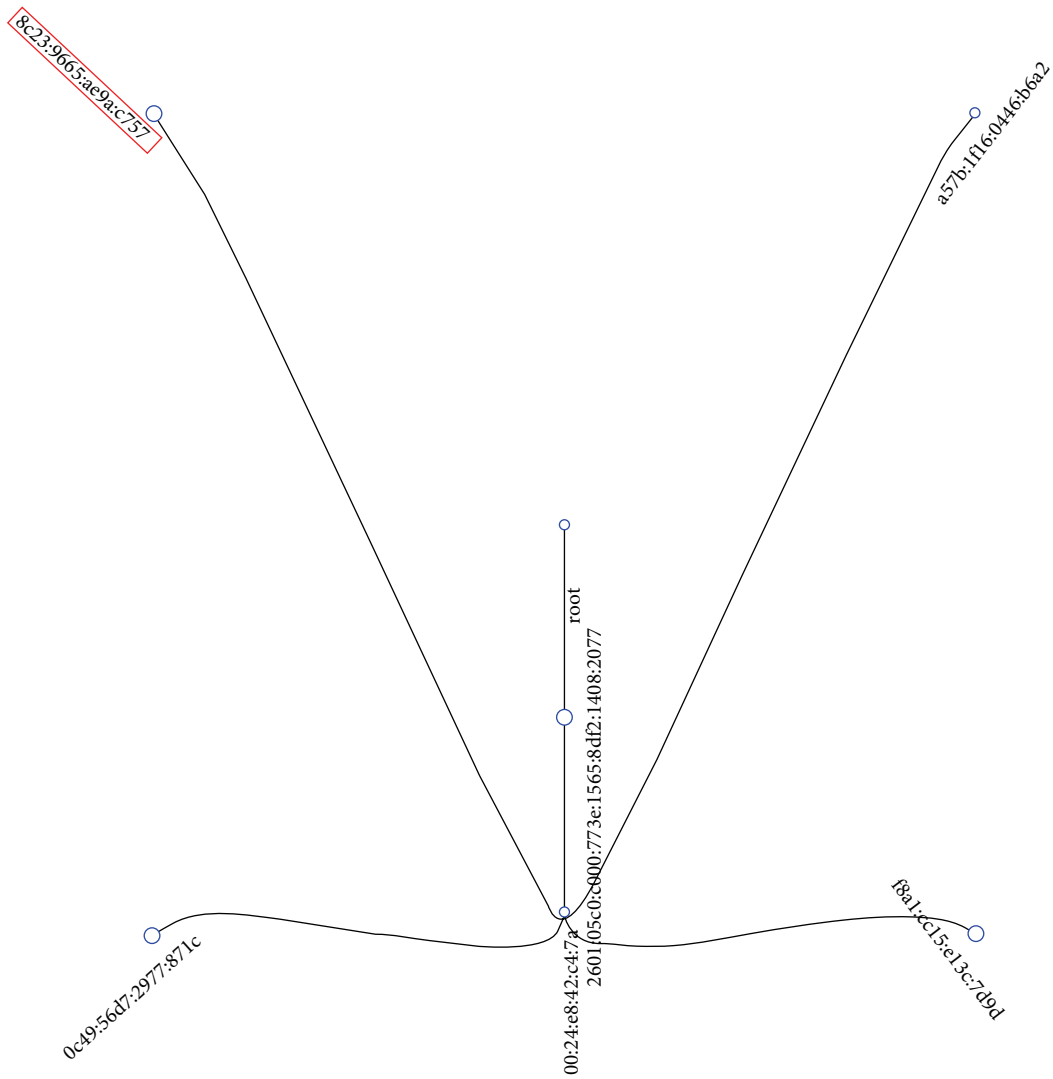


Figure 4: MT6D view depicting addresses spawned by a MT6D host.

The CN sends a "honeypot-requested" message with the address information to the honeypot-host database. The honeypot-commissioning-module retrieves the "honeypot-requested" message, acknowledges the CN with a "honeypot-deployed" message, picks the IP address, and looks up the container-queue for an available honeypot container. The honeypot-commissioning-module identifies an available container (in this case "hpot-container 1") and assigns the IPv6 address under attack, 2001:468:c80:c111:8c23:9665:ae9a:c757, to the local network interface of the honeypot container, and starts the Dionaea service and DionaeaFR python web server on the container.

Figure 5 shows an ICMP ping (sending ICMP echo request probe every 50 ms) results as seen by the attacker. Figure 6 shows the Wireshark capture running on the CN to identify the point when CN stops responding to the ICMP echo request packets incoming on the IPv6-address-of-interest indicating the unbinding. Figure 7 presents a second-level MT6D_view visualization of incoming attack traffic, showing an MT6D host with MAC address 00:24:e8:42:c4:7a receiving ICMP traffic of type 0x80 and a count of 24, on a specific discarded address 8c23:9665:ae9a:c757 from an attacker at address 2601:5c0:c000:773e:1565:8df2:1408:2077. Figure 8 presents attacker-view visualization depicting an attacker at address 2601:5c0:c000:773e:1565:8df2:1408:2077 sending traffic to MT6D host with MAC address 00:24:e8:42:c4:7a on a particular spawned MT6D address, that is, 8c23:9665:ae9a:c757, and the traffic composition is ICMP Type 0x80 and Code 0x00 (ICMP echo request) with a count of 24. MT6D_view and attacker_view present the visualization of the same traffic from two perspectives; the former shows the MT6D host at the center whereas the attacker_view puts the attacker-address at the center of the visualization and plots MT6D hosts that are getting hit from that specific attacker-address.

6.1.2. Observations on Interruption Window. From Figures 5, 6, 7, and 8 we can infer that the first 24 ICMP echo

```
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=19 hlim=52 time=25.866 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=20 hlim=52 time=28.564 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=21 hlim=52 time=30.185 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=22 hlim=52 time=25.166 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=23 hlim=52 time=24.148 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=24 hlim=52 time=34.412 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=26 hlim=52 time=31.686 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=27 hlim=52 time=32.178 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=28 hlim=52 time=32.715 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=30 hlim=52 time=23.711 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=31 hlim=52 time=30.459 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=32 hlim=52 time=24.892 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=33 hlim=52 time=24.720 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=34 hlim=52 time=26.199 ms
```

Figure 5: ICMP pingflood traffic from simulated attacker on a remote Internet connection at 50 ms interval.

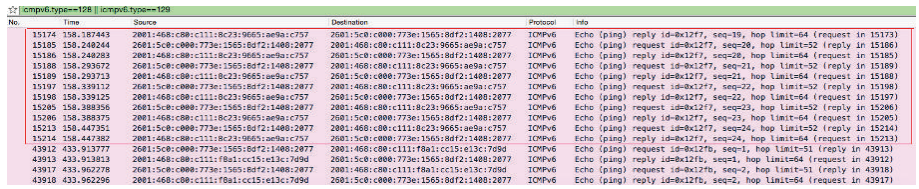


Figure 6: Wireshark capture showing packets hitting CN before the IP address migration.

request (ICMP type = 0x80 and code = 0) packets with sequence IDs till "24", hit the CN and one ICMP echo request packet with sequence ID "25" was lost, while packets (with sequence IDs from "26", "27",...) were routed to honeypot container resulting in subsequent ICMP echo replies in Figure 5 indicating successful IP address transition from CN to honeypot container.

To verify the address-transition and Dionaea service binding we used an nmap NSE auth-spoof (generally used for testing authentication servers for malware infection) script as shown in Figure 9 to generate test-traffic on various ports targeting address of interest, that is, 2001:468:c80:c111:8c23:9665:ae9a:c757. Figure 10 shows the DionaeaFR console displaying incoming connections for the address 2001:468:c80:c111:8c23:9665:ae9a:c757 on ports HTTPD, SMB, SIP, and so forth.

Figure 11 shows the hold-packet-counts and lost-packet-counts for the 10-honeypot deployment trial. We can also observe the IP address migration from the CN to the honeypot container on honeypot-host involves loss of one packet or none, from the perspective of an attacker probing a discarded MT6D address with an ICMP echo request packet every 50 ms. From the Wireshark packet captures, we attribute this packet-loss, that is, unresponsive ICMP echo requests, to the router on Virginia Tech production network either routing the packet to the CN even after the IP address is migrated to the honeypot container or the router forwarding the packet to honeypot container while the container is still in the process of binding the address to its network interface. In either case, even though the packet makes it to CN or the honeypot container depending on routing-table entry, neither of them can respond with ICMP echo reply as the address is not active on their respective network interfaces.

6.1.3. Observations on CPU and Memory Overheads. Figures 12, 13, and 14 show CPU and memory (RAM) loading characteristics for three independent trials of single honeypot container, five-honeypot container, and ten-honeypot container deployment. CPU time comprises Usr-CPU and Sys-CPU. Usr-CPU is CPU time spent in user-mode outside kernel code and Sys-CPU is the CPU time spent in the kernel within the process handling system calls and other kernel-space events. We observed that the trend of memory consumption with honeypot-service being deployed on each container is proportionally linear (~125 MB for each honeypot container launch) while CPU consumption peaks temporarily and then remains constant. The baseline free-memory consumption for the honeypot-host after a clean reboot was observed to be around 3025 MegaBytes. The baseline CPU consumption was observed to be nominal. From the observed memory and CPU consumption trend we substantiate the virtue of using the hot-spare LXC containers approach as the containers waiting as spares place nominal load on system resources and only request resources, Memory and CPU, on-demand when we invoke Dionaea and DionaeaFR management-server services on the container.

Figure 12 shows memory and CPU consumption for a single honeypot container experiment. From the origin until the point "A" on the free-memory curve represents the memory consumption on machine serving as baseline. At epoch time ~1443048570, the LXC container "hpot-container 1" is cloned from honeypot-LXC-template and the "hpot-container 1" is started and lies waiting as hot spare in the queue. We can observe a drop in free-memory at point "A". At epoch time ~1443048801, the honeypot-commissioning-module runs a script to bind the requested IPv6 address, start the Dionaea service, and start the DionaeaFR python web server on container "hpot-container 1". This step brings up Dionaea service bound to desired IP address on "hpot-container 1". We can observe a significant drop (~125 MB) in free-memory at point "B" on the curve in response to Dionaea and DionaeaFR service invocation. We can also observe

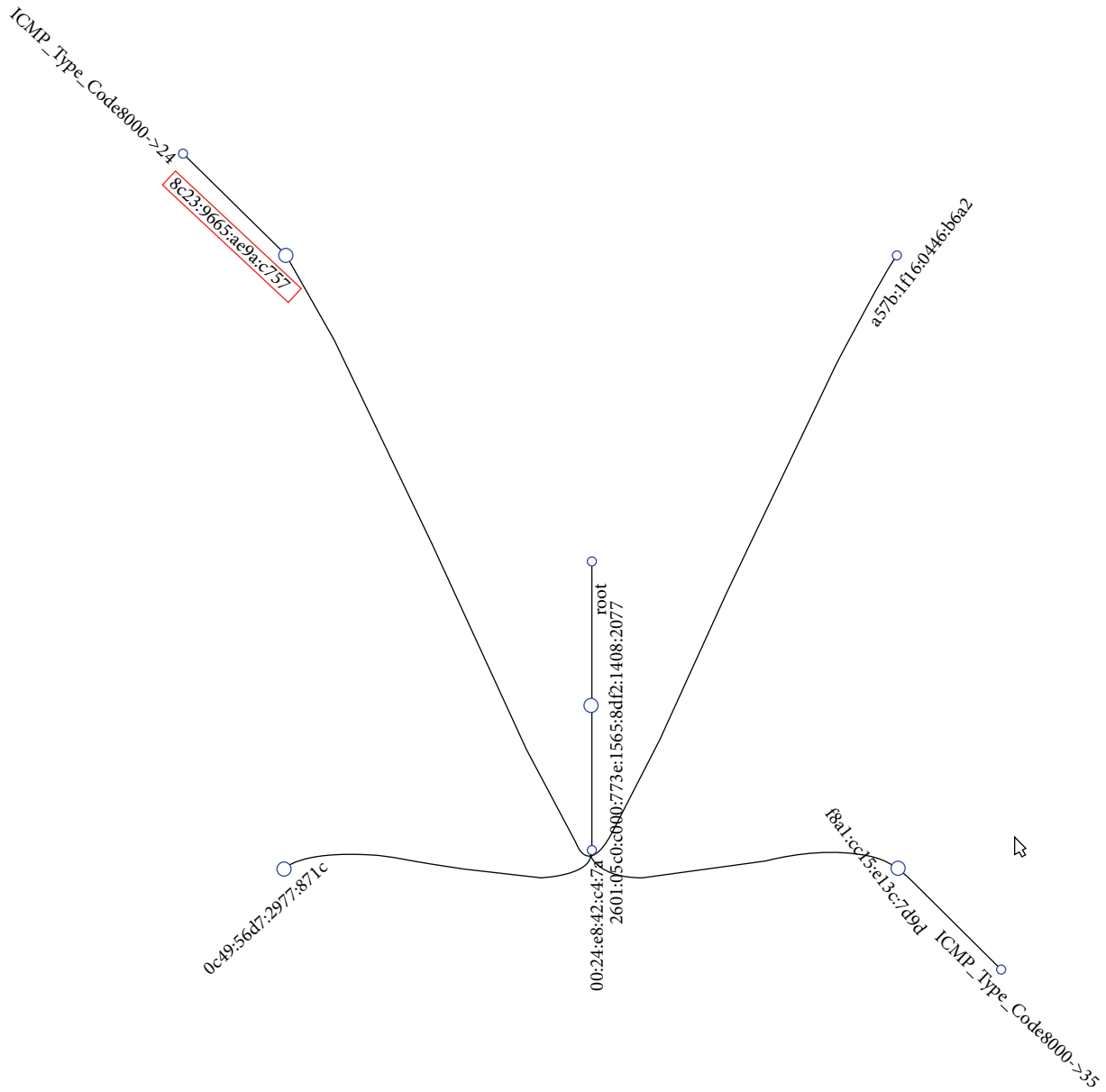


Figure 7: MT6D view of incoming traffic hitting the CN visualized.

the CPU (Usr and Sys CPU) consumption spike to a peak value at the time instant corresponding to point "B", that is, at the launch of honeypot services on the virtual container, and subsequently remain constant.

Figure 13 depicts the CPU and memory consumption trend for a five honeypot container trial. From origin until the point "A" on the free-memory curve represents the baseline memory consumption.

At epoch time ~1443023165 five containers ("hpot-container 1", "hpot-container 2",..., "hpotcontainer 5") are cloned from honeypot-LXC-template. These cloned containers are started and made to wait as hot-spares in queue. At epoch times corresponding to points "B", "C", "D", "E", and "F" on the free-memory curve, one can observe drops in available-memory corresponding to instants of binding requested IP address and invocation of the Dionaea service and DionaeaFR web-server on each LXC honeypot container. We can also observe the CPU consumption spike to peak value at the time instants corresponding to points "B", "C", "D", "E", and "F", that is, at the launch of honeypot services on the virtual-containers, and subsequently lie constant.

Figure 14 shows similar memory consumption behavior for a ten-honeypot-deployment trial, at epoch time corresponding to point "A", that is, ~1443029196; ten containers ("hpot-container 1", "hpot-container 2",..., "hpot-container 10") are cloned, started, and lie waiting in hot spares queue. Epoch times corresponding to points "B", "C", "D", "E", "F", "G", "H", "I", "J", and "K" on the free-memory curve correspond to IP address binding and invocation of Dionaea services. We can also observe the CPU consumption spike to

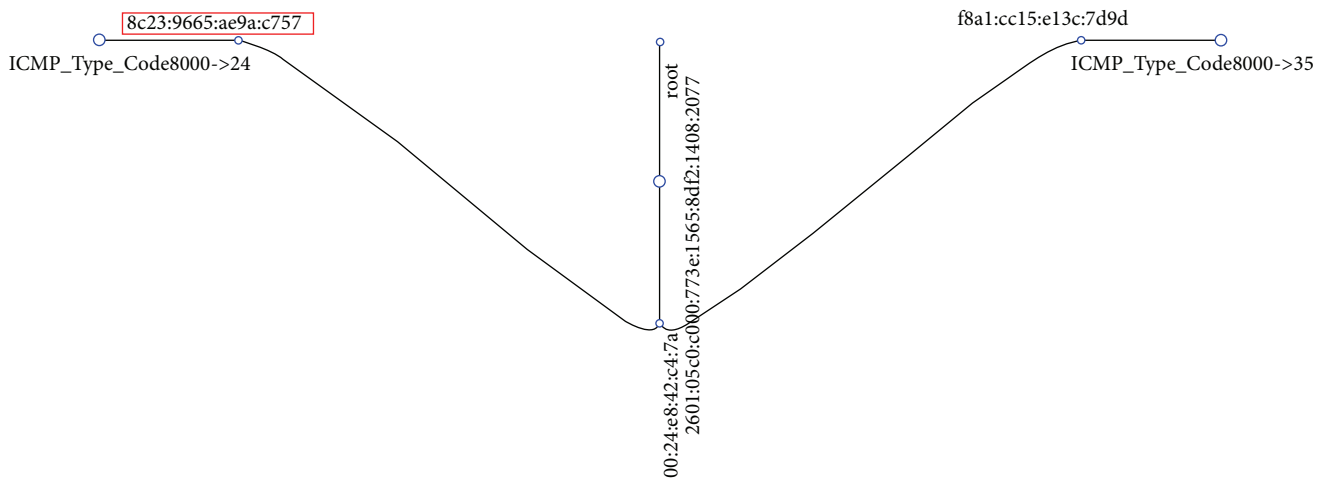


Figure 8: Attacker view of incoming traffic hitting the CN visualized.

```
pc1@pc1-OptiPlex-960:~$
pc1@pc1-OptiPlex-960:~$ sudo nmap -6 -sV --script=auth-spoof 2001:468:c80:c111:8c23:9665:ae9a:c757

Starting Nmap 6.40 ( http://nmap.org ) at 2015-10-02 16:51 EDT
```

Figure 9: Nmap script to send test traffic on various services to a deployed honeypot.

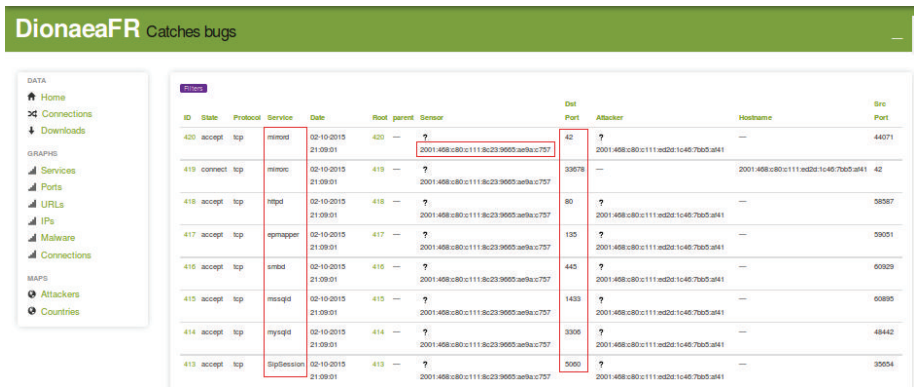


Figure 10: DionaeaFR console showing deployed honeypot connection statistics.

peak value at the time instants corresponding to points "B", "C", "D", "E", "F", "G", "H", "I", "J", and "K", that is, at the launch of honeypot services on the virtual-containers, and subsequently remain constant.

6.2. How Can a MT6D Node Leverage Our Solution. The MT6D nodes can consume the CN's intelligence on attacker-activity using the solution's Mongo database API. The JSON object responsible for Figure 8, the attacker-view visualization, can be queried from the CN's database by any MT6D host to detect a trailing attacker by identifying all the attacker-address nodes that have the MT6D host's MAC address as child node. In this case the MT6D host with the MAC address 00:24:e8:42:c4:7a can analyze the JSON object responsible for Figure 8 to understand its MAC address is a child node for the attacker address 2601:5c0:c000:773e:1565:8df2:1408:2077 and among its relinquished addresses particularly MT6D addresses ending with 8c23:9665:ae9a:c757 and f8a1:cc15:e13c:7d9d were hit with ICMP echo request traffic. Based on the attack traffic composition, the MT6D host can communicate with its MT6D partner and change its scheme parameters (e.g., stronger secret-key and faster hopping-interval) to evade any trailing attackers.

With this solution in place, nodes participating in MT6D can uncover any suspicious activity on their relinquished addresses and be able to use it in fine-tuning the scheme parameters. The solution's traffic enumeration and honeypot deployment capabilities offer insights into attacker methods. In this chapter, we evaluated our solution in terms of interruption window observable to an attacker, CPU, and memory overhead analyses in addition to discussing a mechanism to offer gathered intelligence on relinquished addresses to MT6D nodes through an JSON API. In the next section, we will conclude our findings and provide directions for future work.

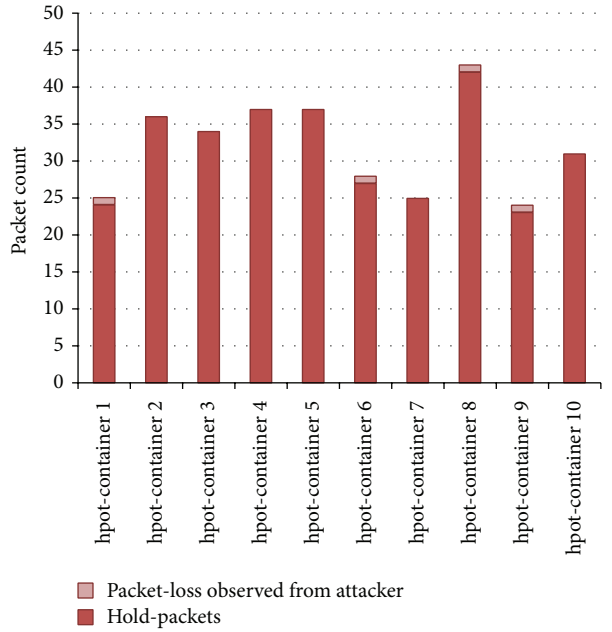


Figure 11: Chart giving out Hold-packet counts and packet-loss observed by an attacker during one of the 10-honeypot deployment trials involving sending an ICMP echo request every 50 ms, that is, rate of 20 packets/second.

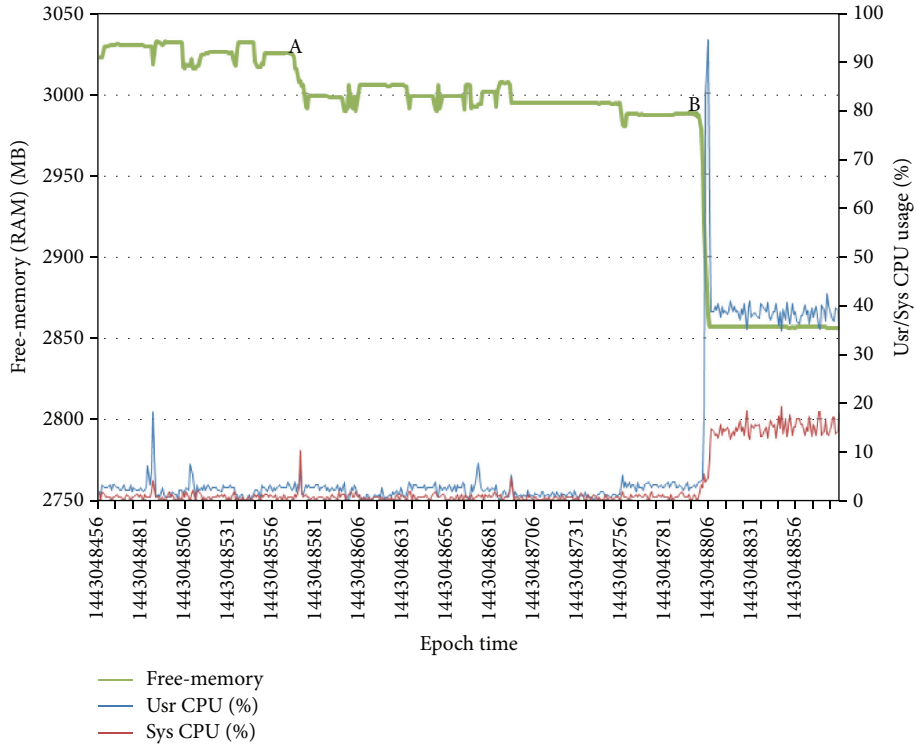


Figure 12: Memory and CPU loading characteristics for the scenario of 1-honeypot container hot spare and subsequent deployment.

7. Future Work

Future work should include CPU and memory overhead analyses of the solution under complex attack scenarios and evaluation of the solution's performance (IP-address migration times and packet-loss observable to attacker) with the honeypot-host sitting in a segregated environment like a DMZ. It is also important to perform overhead analysis on network devices, as such a solution grabbing discarded addresses on a complex MT6D network would place

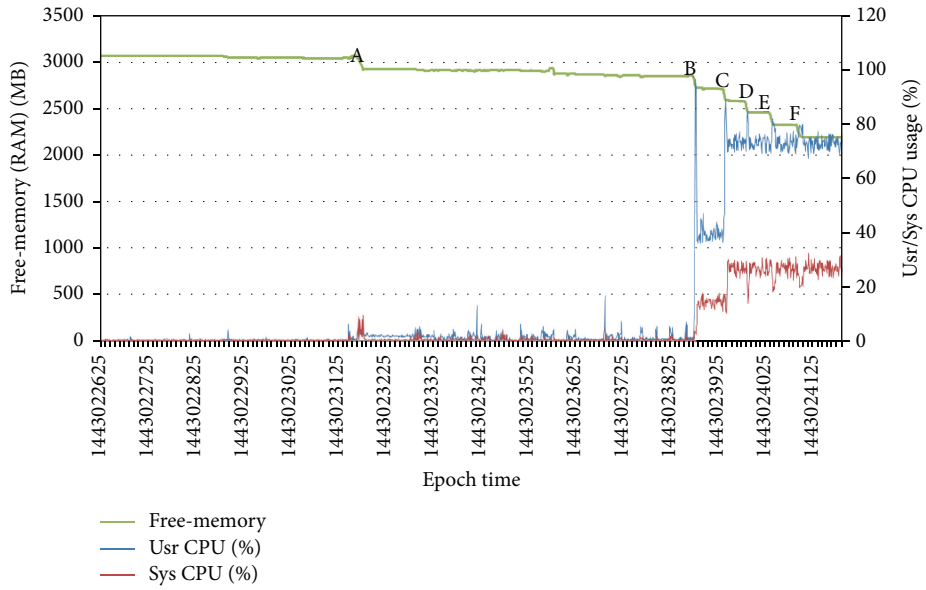


Figure 13: Memory and CPU loading characteristics for the scenario of 5-honeypot container hot spares and subsequent deployment.

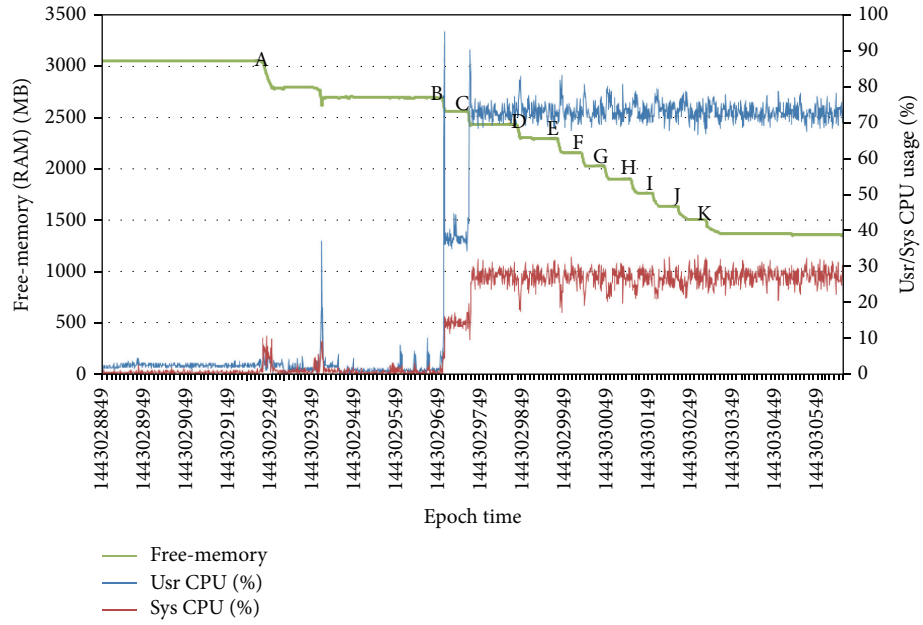


Figure 14: Memory and CPU loading characteristics for the scenario of 10-honeypot container hot spares and subsequent deployment.

overhead on the router in terms of persistent routing table entries. Future work can also include enhancing the solution with a new feature that would reclaim active honeypot containers based on no or low-priority incoming attacker traffic hitting a deployed honeypot container.

8. Conclusion

In this work, we proposed a solution to dynamically deploy a LXC container-based honeypot upon detecting suspicious activity on discarded MT6D addresses. We also evaluated the built solution in terms of interruption observable to an external attacker, CPU, and memory overhead analyses to demonstrate the feasibility and scalablility of such a solution. From our tests, we observed that the interruption observable to an attacker is one packet or none during the migration of the IPv6 address from the CN to the honeypot container. We also observed that the trend of the memory consumption with number of honeypot containers being deployed to be proportionally linear while CPU consumption peaks to a value and stays constant during the launch of the honeypot-service on each container. This supports the argument that

the presented solution in this effort is scalable provided that the memory and compute resources can be catered by the underlying honeypot-host hardware.

The implemented solution offers two visualization views, MT6D_view and attacker-view. MT6D_view displays nodes representing discarded addresses from all MT6D hosts in a local network and their corresponding incoming traffic. Attacker_view displays nodes corresponding to attackers addresses sending traffic and those specific MT6D hosts and discarded addresses that are receiving attack traffic. As we can observe from Figures 7 and 8 attacker_view simplifies the visualization over MT6D_view by the virtue of ignoring MT6D nodes with no interesting traffic. MT6D nodes with no incoming traffic no longer appear on the visualization, thus resulting in less-cluttered visualization of interesting activity. This paves way for visualizing suspicious activity on large MT6D networks without getting burdened by the periodic address-hopping activity.

The solution with database-integration also allows any MT6D node to query for traffic activity on its discarded MT6D addresses as a JSON object. This can be integrated as a feedback mechanism into work that our fellow researchers Morrell et al. are doing in the area of MT6D Client-Server stack in terms of an MT6D Server looking up suspicious activity on discarded addresses and communicating new MT6D scheme parameters such as longer secret-key or faster address-hopping interval to its MT6D client [19].

Previous work done by Morrell et al. [20] showed that a server can successfully bind up to 60000 randomly generated addresses, with each bound address communicating with an individual client. Based on these observations, the CN should be able to bind and listen on a large number of discarded MT6D addresses. We would also like to point out the solution's honeypot-deployment ability is unaffected by the number of MT6D nodes on network, as the incoming traffic enumeration and analysis are delegated to the CN, and the honeypot-host resources are engaged only when the CN matches incoming traffic with a configured attack-traffic signature.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

The authors would like to thank Chris Morrell, Mike Cantrell, and Dr. David Raymond, Virginia Tech, for their many helpful suggestions and comments during the course of research.

References

[1] D. Basam, R. Marchany, and J. G. Tront, "Attention: moving target defense networks, how well are you moving?" in *Proceedings of the 12th ACM International Conference on Computing Frontiers (CF '15)*, article 54, ACM, 2015.

[2] M. Dunlop, S. Groat, W. Urbanski, R. Marchany, and J. Tront, "MT6D: a moving target IPv6 defense," in *Proceedings of the IEEE Military Communications Conference (MILCOM '11)*, pp. 1321–1326, IEEE, Baltimore, Md, USA, November 2011.

[3] M. Dunlop, S. Groat, W. Urbanski, R. Marchany, and J. Tront, "The blind man's bluff approach to security using IPv6," *IEEE Security & Privacy*, vol. 10, no. 4, pp. 35–43, 2012.

[4] Google, "Dionaea, the honeynetproject's 2009," 2009, http://dionaea.carnivore.it/.

[5] R. Espadas, *Dionaeafr*, 2014, https://github.com/rubenespadas/DionaeaFR.

[6] Canonical, "Lxc," 2015, https://linuxcontainers.org/lxc.

[7] P. Biondi, "Scapy," 2011, http://www.secdev.org/projects/scapy.

[8] G. Combs, *Wireshark*, 2007, http://www.wireshark.org.

[9] M. Bostock, "D3.js," Data Driven Documents, 2012.

[10] D. Wieers, "Dstat," 2013, http://dag.wiee.rs/home-made/dstat/.

[11] L. Spitzner, "Honeypots: catching the insider threat," in *Proceedings of the 19th Annual Computer Security Applications Conference*, pp. 170–179, IEEE, December 2003.

[12] I. Kuwatly, M. Sraj, Z. Al Masri, and H. Artail, "A dynamic honeypot design for intrusion detection," in *Proceedings of the IEEE/ACS International Conference on Pervasive Services (ICPS '04)*, pp. 95–104, IEEE, Beirut, Lebanon, July 2004.

[13] C. Hecker, K. L. Nance, and B. Hay, "Dynamic honeypot construction," in *Proceedings of the 10th Colloquium for Information Systems Security Education*, University of Maryland, Adelphi, Md, USA, June 2006.

[14] K. Kishimoto, K. Ohira, Y. Yamaguchi, H. Yamaki, and H. Takakura, "An adaptive honeypot system to capture IPv6 address scans," in *Proceedings of the IEEE International Conference on Cyber Security (CyberSecurity '12)*, pp. 165–172, Washington, DC, USA, December 2012.

[15] J. Hieb and J. H. Graham, "Anomaly-based intrusion detection for network monitoring using a dynamic honey pot," Tech. Rep. TR-ISRL-04-03, Intelligent Systems Research Laboratory, University of Louisville, Louisville, KY, USA, 2004.

[16] A. W. Brzeczko, *Scalable framework for turn-key honeynet deployment [Ph.D. dissertation]*, Georgia Institute of Technology, Atlanta, Ga, USA, 2014.

[17] N. Memari, S. J. Hashim, and K. Samsudin, "Design of a virtual hybrid honeynet based on lxc virtualisation for enhanced network security," *Defence S&T Technical Bulletin*, vol. 7, no. 2, p. 120, 2014.

[18] P. Sokol and P. Pisarcik, "Digital evidence in virtual honeynets based on operating system level virtualization," in *Proceedings of the Security and Protection of Information*, pp. 22–24, Brno, Czech Republic, May 2013.

[19] C. Morrell, R. Moore, R. Marchany, and J. G. Tront, "DHT blind rendezvous for session establishment in network layer moving target defenses," in *Proceedings of the 2nd ACM Workshop on Moving Target Defense (MTD '15)*, pp. 77–84, ACM, Denver, Colo, USA, October 2015.

[20] C. Morrell, J. S. Ransbottom, R. Marchany, and J. G. Tront, "Scaling IPv6 address bindings in support of a moving target defense," in *Proceedings of the 9th International Conference for Internet Technology and Secured Transactions (ICITST '14)*, pp. 440–445, London, UK, December 2014.

Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2016, Article ID 1075648, 8 pages
http://dx.doi.org/10.1155/2016/1075648

Research Article

Hybrid Intrusion Detection System for DDoS Attacks

Özge Cepheli,[1] Saliha Büyükçorak,[1,2] and Güneş Karabulut Kurt[1]

[1]*Department of Electronics and Communication Engineering, Istanbul Technical University, 34469 Istanbul, Turkey*
[2]*Gebze Technical University, 41400 Kocaeli, Turkey*

Correspondence should be addressed to Özge Cepheli; irmakoz@itu.edu.tr

Received 6 November 2015; Revised 27 January 2016; Accepted 29 February 2016

Academic Editor: Andrea Ceccarelli

Distributed denial-of-service (DDoS) attacks are one of the major threats and possibly the hardest security problem for today's Internet. In this paper we propose a hybrid detection system, referred to as hybrid intrusion detection system (H-IDS), for detection of DDoS attacks. Our proposed detection system makes use of both anomaly-based and signature-based detection methods separately but in an integrated fashion and combines the outcomes of both detectors to enhance the overall detection accuracy. We apply two distinct datasets to our proposed system in order to test the detection performance of H-IDS and conclude that the proposed hybrid system gives better results than the systems based on nonhybrid detection.

1. Introduction

Distributed denial-of-service (DDoS) attacks stand as a crucial threat to Internet services. A DDoS attack is launched by producing an extremely large amount of traffic to exhaust resources of target systems. As shown in Figure 1, the attack is generally initiated by a single attacker, exploiting and taking control of several devices referred to as zombies. Frequently zombie devices are not aware of the fact that they are being used to perform an attack. The attacker usually makes a sweep operation to determine the devices that are eligible for being used as a zombie, for example, a device with an open port. After this stage, the attack is initiated by the attacker using zombie devices. As the number of zombies can be around hundreds or thousands (and theoretically it is possible to have even more) the detection of the attacker becomes a very hard task.

A number of methods have been proposed to prevent DDoS attacks in the literature, though there is still lack of a methodology addressing all requirements. Therefore, DDoS attacks are still a huge threat to network security. In this paper we propose a novel framework named as hybrid intrusion detection system (H-IDS) to detect DDoS attacks. In this system, in order to achieve more accurate detection we use both anomaly-based and signature-based detection techniques. Anomaly-based detector part of the proposed H-IDS is designed by using multidimensional Gaussian mixture models (GMMs) from a training dataset, while signature-based detector is formed by using SNORT [1]. In addition to this, we design a node referred to as hybrid detection engine (HDE) in order to control and evaluate outputs of these detectors. The proposed H-IDS enhanced the overall performance of DDoS attack detection and shortened the detection delay through using two detectors separately but in an integrated fashion. The proposed H-IDS can be implemented as a module in any IDS solution, as well as being used as a separate DDoS detection system. For the detection performance evaluation of the proposed hybrid detector, we utilize the widely used DARPA 2000 dataset and a dataset provided by a commercial bank in Turkey during a penetration test. With the H-IDS, true positive rate (TPR) is obtained as 92.1% for DARPA and 99.9% for the commercial bank dataset. The TPR is increased by 27.4% with the proposed H-IDS when compared to the signature-based detector for the dataset DARPA.

The remainder of this paper is organized as follows. In Section 2, a detailed overview of the related literature is given. In Section 3, the proposed H-IDS and its components are detailed along with working principles of this hybrid detector. In Section 4, we evaluate experiments by using two distinct datasets to validate our detection model. We conclude this paper in Section 5.

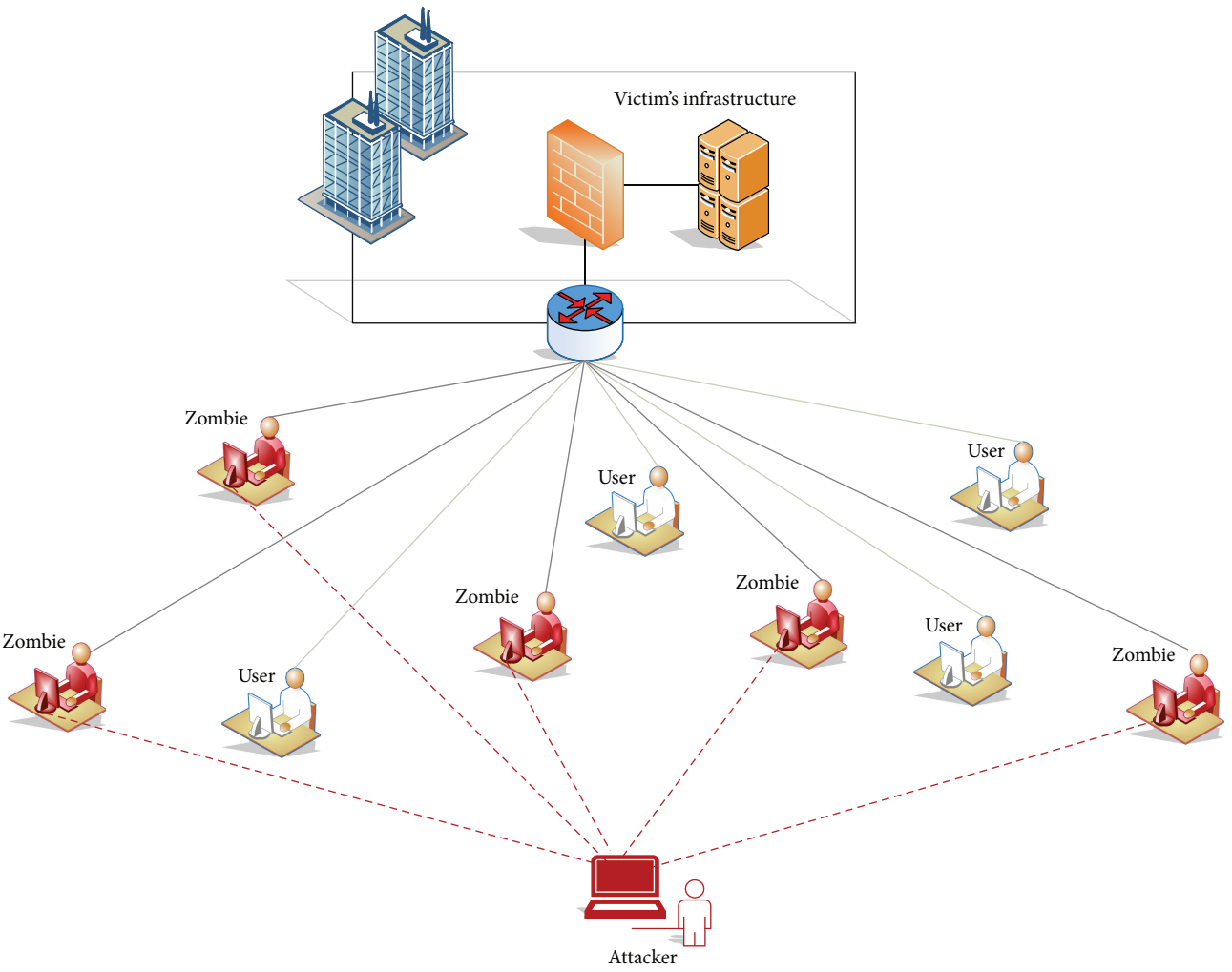


Figure 1: DDoS attack model.

2. Literature Overview

The entropy based DDoS countermeasure methods are independent of the specific attack features. In [2], Tao and Yu proposed a flow entropy based DDoS attack detector. The effectiveness of this method is shown thorough various experiments and simulations. The authors offered a mechanism for IP traceback against DDoS attacks based on entropy variations between normal and attack traffic. This is fundamentally different from commonly used packet marking [3, 4]. Xiang et al. proposed a novel low-rate DDoS attacks detector ground on new information metrics (i.e., the generalized entropy metric and the information distance metric). It is demonstrated that these metrics can expressly reduce the false positive rate by using actual DDoS datasets [5, 6].

DDoS attacks can be detected by examining of the network traffic changes. There are many proposed countermeasure methods based on self-similarity of the network. In [7], the authors introduced a real-time DDoS attack detector based on network self-similarity. It is shown that the attacks can be detected effectively and precisely using the rescaled range algorithm. In the study performed by Chonka et al. [8], by using the property of network self-similarity, a chaotic model is developed to find out DDoS flooding attack traffic. Chen et al. [9] proposed a DDoS intrusion detection algorithm ground on preprocessing network traffic and chaos theory that can detect an anomaly caused either by bursty legitimate traffic or by DDoS flooding attacks. The proposed algorithm's performance is improved by utilizing an exponential smoothing model as forecasting model [10].

Probabilistic methods are also frequently used to detect DDoS attacks. Joshi et al. [11] tested the efficiency of the cloud traceback (CTB) by using a back propagation neural network, named cloud protector, and came to the conclusion that the proposed CTB helps to find out the real sources of attacking packets. Thing et al. [12] proposed a new and high speed nonintrusive traceback technique based on the rationale that packets relating to a particular source-destination flow follow a relatively static path through routers. In [13], the authors introduced a novel anomaly detector ground on hidden semi-Markov model to detect the application layer based DDoS attacks. The effectiveness of this method is demonstrated

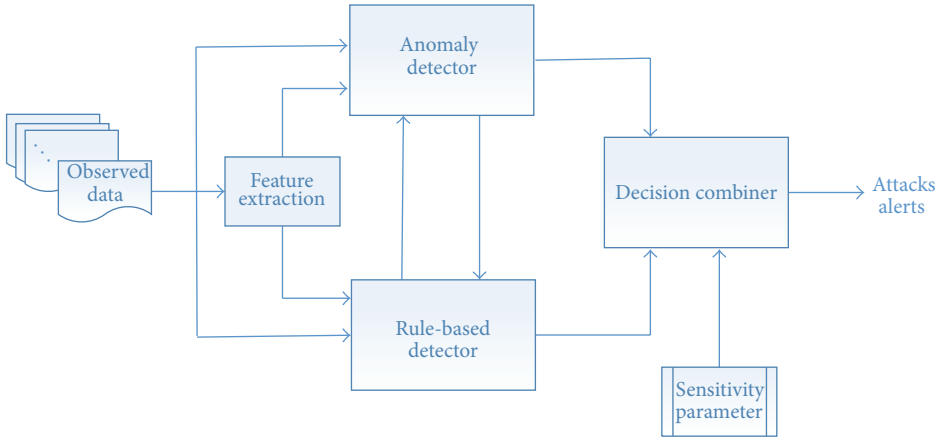


Figure 2: Proposed H-IDS model.

by conducting experiments using real web traffic data. The authors reached the conclusion that identifier/location separation can help to prevent DDoS attacks by investigating numerical results based on the real data [14].

In the study performed by Barati et al. [15], by using a machine learning technique composed of genetic algorithm and artificial neural network, it is shown that the accuracy of DDoS attack detection is improved. Yu et al. [16] guaranteed the quality of service for legitimate users by using a dynamic resource allocation strategy to confront DDoS attacks that target individual cloud customers. Thapngam et al. [17] investigated a detector based on the pattern behavior of traffic sources by observing packet arrivals. It is shown through experiments with several datasets that the proposed detector can discriminate DDoS attack traffic from flash crowd with a quick response.

There are also many hybrid detection algorithms proposed for DDoS attack detection. Hwang et al. [18] proposed a hybrid system that combines a signature-based IDS with an anomaly detection system in a cascade structure, achieving twice the detection accuracy of IDS only system. Gómez et al. [19] extended SNORT by adding an anomaly detection preprocessor. Afterwards, various hybrid systems are proposed following the same aim, to have the strengths of both signature- and anomaly-based detection.

In this paper, we propose a novel hybrid intrusion detection system (H-IDS) to accurately detect DDoS attacks. Our developed system makes use of both anomaly-based and signature-based detection methods in parallel. The decision combiner, which is the core processing unit of the system, combines outputs of the detectors and then generates an attack alarm with a tunable sensitivity parameter. Note that our proposed H-IDS differs from the existing studies in the literature, with its parallel detection methodology, due to its flexible nature with decision combiner and having tunable parameters.

3. Hybrid Intrusion Detection System (H-IDS)

The H-IDS designed within this paper is based on an original approach, where the outputs of an anomaly-based detector and a signature-based detector are collected. The parameters of the detectors are controlled by a centralized node. This node is referred to as hybrid detection engine (HDE). The design goal of this intrusion detection system is to enhance the overall performance of DDoS attack detection, by shortening the detection delay, while increasing the detection accuracy. The block diagram of the proposed H-IDS is shown in Figure 2. As can be seen from this figure, the observed data containing normal traffic and DDoS attacks is processed to extract some features; then processed data is linked to signature-based and anomaly-based detector blocks to detect attacks. Outputs of these detectors are examined by a decision combiner and an alarm gets produced according to sensitivity parameter. The components of the hybrid IDS are explained in detail in the following subsections.

3.1. Feature Extraction and Activity Model Calculation. The first step of the proposed detection process is to analyze the network traffic and to extract some features to build an activity model. In order to give an a priori idea of the detection problem, time analysis of DARPA 2000 dataset is given in Figure 3. From this figure, one can conclude that it is not an easy task to even distinguish between normal and attack periods by solely observing traffic density.

The model of normal network traffic can be achieved by using training data. The training data period can be as short as hours or as long as weeks, similar to the case in DARPA dataset. As the length of the training period affects the model accuracy greatly (but results in a delay), the time required for the training should be optimized in implementation.

In our study, the following features widely used in DDoS studies are selected: packet interarrival times, packet sizes, and protocol frequencies. Note that there are several features that can be used in order to achieve maximum performance.

3.2. Anomaly Detector. In this work, by using multidimensional Gaussian mixture models (GMMs), an anomaly-based detector is designed to distinguish normal and abnormal traffic in the data obtained from the feature extraction step. Expectation maximization (EM) algorithm is used

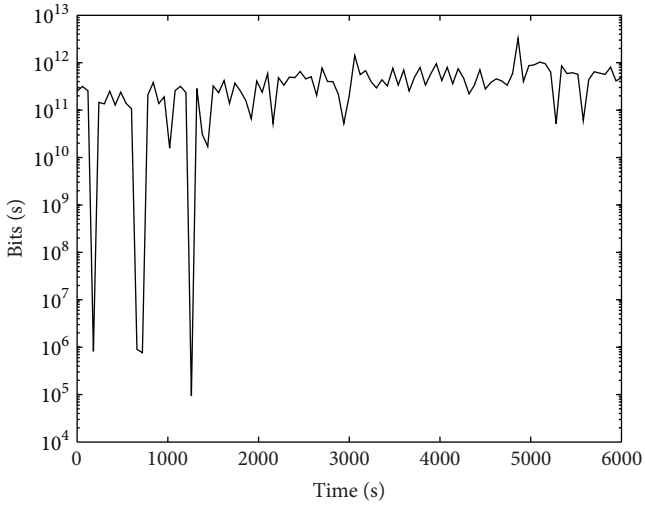


Figure 3: Time domain analysis of DARPA data showing traffic density in bits per second (bps) in logarithmic scale.

to estimate the parameters of the GMMs. Afterwards, the distance between the parameters is investigated and detection is made based upon comparison of this distance with defined thresholds, which constitutes the sensitivity parameter in H-IDS. The output of the anomaly detector is defined as isAlarm_a.

3.2.1. Expectation Maximization Algorithm. EM algorithm is commonly used for simplifying difficult maximum likelihood estimate (MLE) problems that are frequently encountered in mixture models and cannot be analytically solved [18, 20]. This algorithm is a practical parameter estimation technique and named as parametric methods, where the number of mixture components needs to be known a priori.

Let $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ be a given dataset, where $\mathbf{x}_i$ is an M-dimensional vector measurement. In a mixture model, the probability density function (pdf), $p(\mathbf{x})$, can be defined as in the following with a finite K component [20–22]:

$$p(\mathbf{x} \mid \Theta) = \sum_k \omega_k p_k(\mathbf{x} \mid \theta_k), \tag{1}$$

where $p_k(\mathbf{x} \mid \theta_k)$ is defined with parameters θ_k over $p(\mathbf{x})$ and refers to each component of the mixture. $k = 1, 2, \ldots, K$ is the number of mixture components and ω_k is the proportion of kth mixing component in the mixture (which are positive and sum to one). $\Theta = (\omega_1, \omega_2, \ldots, \omega_K, \theta_1, \theta_2, \ldots, \theta_K)$ is the complete set of parameters in a mixture model with K components. The component density $p_k(\cdot)$ is a multidimensional Gaussian distribution defined as

$$p_k(\mathbf{x} \mid \theta_k) = \frac{1}{(2\pi)^{M/2} \det(\Sigma)^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)\Sigma_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)\right], \tag{2}$$

where $\theta_k = (\boldsymbol{\mu}_k, \Sigma_k)$ represents parameters of each component in the mixture. $\boldsymbol{\mu}_k$ is the mean vector of length M and Σ_k is the covariance matrix of size $M \times M$. As defined in (1) and (2), Gaussian mixture models assume that all the data points are originated from a mixture of a finite number of Gaussian distributions with unknown parameters.

The EM algorithm begins with some initial estimated Θ values and proceeds by iteratively updating Θ until convergence. Each iteration consists of two steps: the expectation step (E-step) and the maximization step (M-step) [22, 23].

In the E-step, the membership coefficients of data point $\mathbf{x}_i$ in component k are calculated by using the current parameter values Θ as [22]

$$\gamma_{ik} = \frac{\omega_k p_k(\mathbf{x}_i \mid \theta_k)}{\sum_{k=1}^{K} \omega_k p_k(\mathbf{x}_i \mid \theta_k)}, \quad 1 \le k \le K,\ 1 \le i \le N, \tag{3}$$

where $\mathbf{x}_i$ refers to the data in the kth mixture and $\sum_{k=1}^{K} \gamma_{ik} = 1$.

In the M-step, parameter values are updated as mean, covariance, and mixing proportion belonging to each component in the mixture model, by using the membership coefficients obtained in the E-step and the dataset. The new mixture weights are calculated as

$$\widehat{\omega}_k = \frac{1}{N}\sum_{i=1}^{N} \gamma_{ik}, \quad 1 \le k \le K. \tag{4}$$

The updated mean values are obtained as

$$\widehat{\boldsymbol{\mu}}_k = \frac{\sum_{i=1}^{N} \gamma_{ik}\mathbf{x}_i}{\sum_{i=1}^{N} \gamma_{ik}}, \quad 1 \le k \le K. \tag{5}$$

Note that this is a vector equation since $\widehat{\boldsymbol{\mu}}_k$ and $\mathbf{x}_i$ are both M-dimensional vectors. Lastly, the covariance matrices of each component are calculated as

$$\widehat{\Sigma}_k = \frac{\sum_{i=1}^{N} \gamma_{ik}(\mathbf{x}_i - \widehat{\boldsymbol{\mu}}_k)(\mathbf{x}_i - \widehat{\boldsymbol{\mu}}_k)^T}{\sum_{i=1}^{N} \gamma_{ik}}, \quad 1 \le k \le K. \tag{6}$$

The M-step is defined by calculating new whole parameters and then membership weights are recalculated by going back to the E-step. The algorithm iteratively calculates estimation values with maximum likelihood for parameters by applying the E- and M-steps, iteratively.

3.2.2. Information Distance Metrics. The information distance metrics can be described as methods to measure the norm of the similarity between two pdfs. In this work, these metrics are used to quantify the distance between the pdfs of normal and abnormal traffic, and the distance $\mathscr{D}$ is chosen as the output of the anomaly detector.

Let $\mathscr{P} = (p_1, p_2, \ldots, p_{\mathscr{T}})$ and $\mathscr{Q} = (q_1, q_2, \ldots, q_{\mathscr{T}})$ represent two discrete probability distributions. The Kullback-Leibler ($\mathscr{KL}$) distance can be described as [2]

$$\mathscr{D}_{\mathscr{KL}}(\mathscr{P}, \mathscr{Q}) = \sum_t p_t \log_2 \frac{p_t}{q_t}. \tag{7}$$

Here, we note that $\mathscr{KL}$ distance cannot be a perfect metric due to the asymmetry properties, which will result in potential problems. There are a few metrics (i.e., Jeffrey distance,

Sibson distance, and Hellinger distance) that can handle the asymmetric problem of the $\mathcal{KL}$ distance [2, 17].

The Sibson distance can be calculated based on the $\mathcal{KL}$ distance as

$$\mathcal{D}_{\mathcal{S}}(\mathcal{P}, \mathcal{Q}) = \frac{1}{2}\left[\mathcal{D}_{\mathcal{KL}}\left(\mathcal{P}, \frac{1}{2}(\mathcal{P} + \mathcal{Q})\right) + \mathcal{D}_{\mathcal{KL}}\left(\mathcal{Q}, \frac{1}{2}(\mathcal{P} + \mathcal{Q})\right)\right]. \tag{8}$$

In the literature, it is indicated that the Sibson distance is a suitable candidate for DDoS detection in terms of data sensitivity and statistical features [2, 17]. Accordingly, in our numerical experiments, we choose Sibson distance and get a constant value as threshold (α) from HDE and calculated isAlarm_a as

$$\text{isAlarm}_a = \begin{cases} 0, & \mathcal{D}_{\mathcal{S}}(\mathcal{P}, \mathcal{Q}) < \alpha \\ 1, & \mathcal{D}_{\mathcal{S}}(\mathcal{P}, \mathcal{Q}) \geq \alpha. \end{cases} \tag{9}$$

3.3. Signature-Based Detector. Signature-based detector is a type of attack detectors that uses predefined signature sets in order to detect an alarm. The main principle is to extract some features from the traffic data and compare the values of these features with the predefined rules. This process usually does not depend on the application specific cases; however it is usually easier to implement and manage.

The first approach to detect network attacks is to use rule sets. This is the basis of all current IDS or intrusion prevention systems (IPSs) that are used in practice. Hence, there are many tools available, developed by various groups/companies. In addition to the proprietary solutions as the IPS feature of Palo Alto Next Generation Firewall and Juniper IDP, there are also open source signature-based solutions as SNORT [1] and Suricata [24]. In the scope of this study, we used SNORT as our signature-based detector. We specifically choose the rules that are commonly applied in the literature.

SNORT is a free and open source intrusion detection and prevention system (IDPS), created by Martin Roesch in 1998. After the acquisition by Cisco Systems on October 7, 2013, it continues to be developed as an open source solution. It is a widely used solution for network intrusion detection both for practical and for research implementation.

SNORT can be configured to run in three modes:

(i) Sniffer mode, which simply reads the packets off the network and displays them in a continuous stream on the console (screen).

(ii) Packet logger mode, which logs the packets to disk.

(iii) Network IDS mode, which performs detection and analysis on network traffic; this is the most complex and configurable mode.

The rule set of SNORT can be modified for special requirements. Note that different rule sets should be chosen for different performance results. However, in general, extensive optimization of all the rules in the rule set is not aimed at during the implementation of a signature-based solution. Instead one can use the periodically updated rule sets and further create additional rules for special requirements [24]. Granularity of the rule set can be changed on the run to control the security level of the detector. Hence, the amount of work that is necessary to configure the rule-based approach is less than that of the anomaly-based one. In our system, the granularity of the rule set is set by the HDE. The output of SNORT is denoted by isAlarm_r and calculated based on the value $\mathcal{A}(k)$, where k is the time frame index and $\mathcal{A}(k)$ is chosen as the number of generated alerts within the kth time frame. Using $\mathcal{A}(k)$, isAlarm_r is calculated as

$$\text{isAlarm}_r = \begin{cases} 0, & \mathcal{A}(k) = 0 \\ 1, & \mathcal{A}(k) \geq 0. \end{cases} \tag{10}$$

3.4. Hybrid Detection Engine. In this paper, we make use of anomaly- and signature-based detectors and combine their output in order to enhance the overall performance. Also, the hybrid detection engine controls the sensitivity levels of the anomaly- and signature-based detectors according to the calculated suspicion value. The functionalities of HDE can be listed as follows:

(i) Collecting the outputs of anomaly-based detector and signature-based detector.

(ii) Calculating the attack probability.

(iii) Controlling the security levels of the detectors.

(iv) Updating anomaly detector's normal network model.

(v) Updating the signature-based detectors rule set.

These functionalities are detailed below.

3.4.1. Collecting the Outputs of Detectors. The collection of outputs can be conducted in two different approaches: hard detection and soft detection. In hard detection, the outputs of the detectors are the isAlarm value, which is a binary number indicating if there is an attack or not. In soft detection, the outputs of the detectors are collected as a value referring to probability of an attack. As stated previously, hard detection is used within this study, for the results given in the next section. However, we propose the framework of HDE enabling the use of soft detection.

3.4.2. Calculation of the Attack Probability. The HDE calculates the final decision on the probability of an attack by using the collected outputs of the anomaly- and signature-based detectors. The calculation is performed according to a weighted correlation of the two detector inputs. For a hard decision we can define the process as a function as shown in Figure 4. The overall performance is highly related to the threshold selection (th_1 and th_2) of this function.

When using more than one detector, there is always a possibility that one of the detectors detects an intrusion while the other does not. In case of such an output (the blue fields in Figure 4), one option is to use "OR" relation, which means to decide on presence of an attack even if only one of the

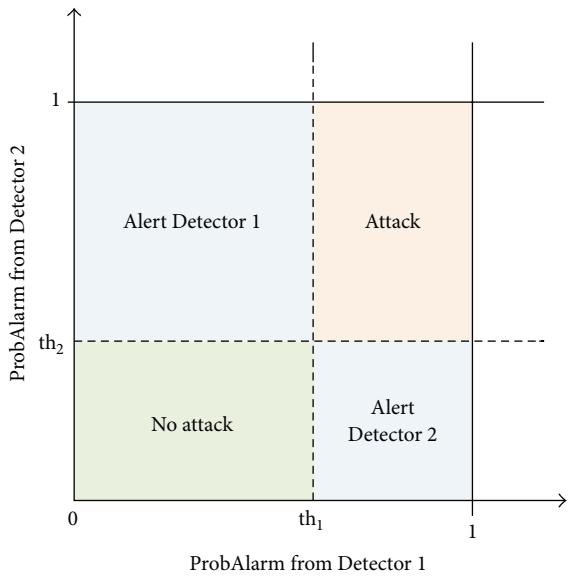


Figure 4: Detection function of HDE.

detectors detects an attack. The other option is to use "AND" relation, where the final decision is presence of the attack if and only if both of the detectors report an intrusion.

For the soft decision, one should provide a probability value for every point in the 2D plane in Figure 4.

3.4.3. Controlling the Security Levels of the Detectors. The security levels of the detectors are controlled by HDE, according to the suspicion level (attack probability). In lower security levels, H-IDS is on a light-working mode; the detector works with a less granular traffic model which has a lower processing power requirement. Hence, it is suitable for systems with high volume of data and lower processing abilities. The security levels can be configured in an adaptive manner according to the production requirements.

The security level of anomaly detector controls the detail level of the traffic modeling. The anomaly detector works with the most detailed model (more Gaussian mixture components) in the highest security level, while it uses a simple network activity model (one or two Gaussian mixture components) for other cases.

The security level of the signature-based detector controls the richness of the applied rule set. For lower security levels, a simple rule set is used while in higher security levels the content of the rule set is extended.

3.4.4. Updating Anomaly Detector's Normal Network Model. One of the most important properties of H-IDS is the feedback feature. If there is an attack that one of the detectors detects and the other does not, it usually means that one of the detectors has missed an attack or the other one gave a false alarm. If the signature-based method detects an alarm with a high probability and the anomaly detector has not detected any anomaly, then we should update the normal network activity model accordingly. This way, we can ensure that the normal network model does not involve an attack situation.

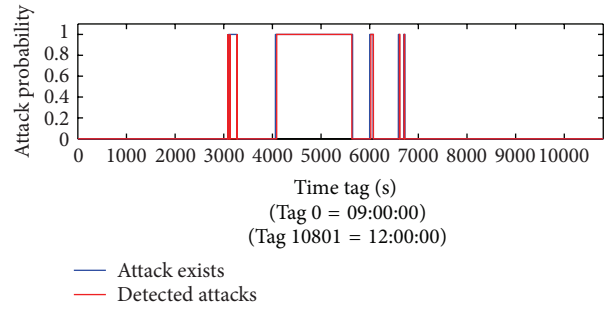


Figure 5: Hard detection results of DARPA dataset.

3.4.5. Updating the Signature-Based Detectors Rule Set. Similar to the update process of the anomaly-based detector's model, HDE also updates the rule set of the signature-based detectors, if it determines that the rule set has missed an attack. The updates of rule parameters are made directly, while rule additions may require a decision support system.

4. Numerical Results

In order to test the performance of our proposed system, the H-IDS is applied to DARPA 2000 dataset and a dataset acquired from a commercial bank in Turkey. The performance indicators are chosen as true positive rate (TPR) and false positive rate (FPR), which are calculated by

$$\begin{aligned} \text{TPR} &= \frac{N_{\text{TD}}}{N_A}, \\ \text{FPR} &= \frac{N_{\text{FD}}}{N_{\text{NA}}}, \end{aligned} \tag{11}$$

where N_{TD} and N_{FD} are the numbers of true detection instances and false detection instances, respectively. N_A represents the number of attack packets, whereas N_{NA} is the number of normal (nonattack) packets.

For the first step in our experiments, a low security level H-IDS using the OR rule is implemented with hard decision and the following results are achieved.

4.1. DARPA 2000 Dataset. We used the dataset DARPA 2000 Lincoln Laboratory Scenario (DDoS) 1.0 which is provided by MIT [25]. This dataset has been used in many studies to test performance of DDoS attack detection.

The attack scenario is carried out over multiple network and audit sessions. These sessions have been grouped into 5 attack phases over the course of which the attacker probes, breaks in, installs Trojan mstream DDoS software, and launches a DDoS attack against an off-site server.

The DARPA dataset is analyzed by using the H-IDS with a hard decision system and by using the OR rule. The obtained results for the first and second weeks of the available data are given in Figure 5. Here, we can see that we have detected the attack with 98.7% TPR and 0.73% FPR by utilizing the proposed H-IDS. Using the AND rule instead of the OR rule, we would have 61.6% TPR and 0.01% FPR.

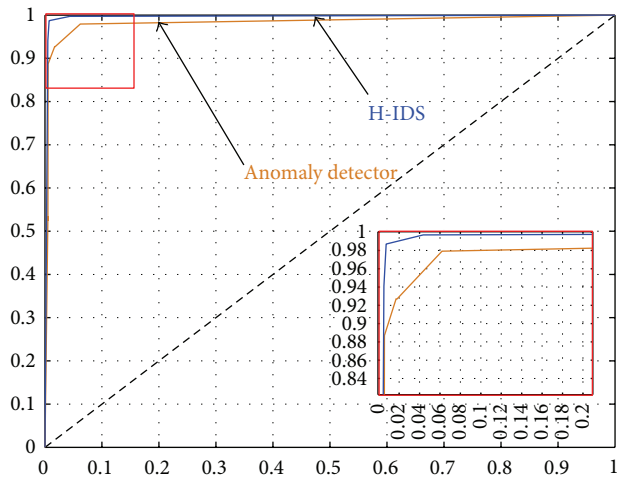


Figure 6: ROC curve for DARPA dataset.

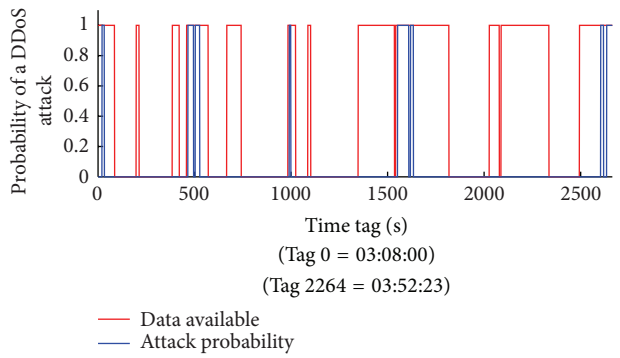


Figure 7: Hard detection results of a commercial bank penetration test dataset.

We also analyzed the sole detection rates of anomaly detector and signature-based detector. With anomaly detector we get a 92.1% TPR and 1.8% FPR, and with signature-based detector we get 64.7% TPR and 13.2% FPR. We can easily see that the attack detection with the H-IDS with OR rule outperforms both systems. This result is similar to the results of [18], as the H-IDS outperforms the single detector systems. Please note that as the authors in [18] have used a different dataset, combining the real-life traffic with the MIT/LL attack dataset, it is not possible to make an exact comparison. However, they reported a 47% detection rate for their system at 1% false alarms and 60% detection rate if the false alarms can be tolerated up to 30%. SNORT has almost a constant 30% detection rate with less than 0.1% false alarm rate.

In Figure 6, the receiver operating characteristic (ROC) curves for both anomaly detector and H-IDS are given for various thresholds (α). The curves show the trade-off between the detection rate and false positive rate under various attacks. Our detection scheme achieves closer to ideal detection performance than the sole use of anomaly detector. This result proves the effectiveness of our H-IDS detection mechanism.

4.2. Dataset of a Commercial Bank from a Penetration Test. The dataset provided by a commercial bank includes banking network data in production and a DDoS attack which is deliberately performed by 400 nodes (zombies) from Amazon.com servers to one web server in the bank's network. There were several ICMP echo attacks within a 45-minute period, each active for 3–7 minutes. The dataset contains 17239 unique IP addresses as destination IP where only one of them is the attack target. The dataset is analyzed and a hard decision system is made on available data. The results are given in Figure 7. Here, our system has a 99.9% TPR and 0.01% FPR, which is very successful. However please note that this particular DDoS attack was easy to detect scenario that is deployed in a penetration test, with very little background traffic and a traceable behavior. The detection performance would probably be lower if more stealthy attacks were performed, especially when attackers try to evade traces that are detectable by the signature-based detector. However even in this case the anomaly detector may detect changes in the network model.

5. Conclusion

In this paper we propose a novel hybrid detection system referred to as H-IDS, which is composed of anomaly-based and signature-based detection techniques for more accurate DDoS attack detection. The proposed detection system can be adopted to networks with varying traffic patterns due to the flexibility provided through the used decision combiner and the associated sensitivity parameter. We test the proposed H-IDS's performance against systems based on nonhybrid detection by using two distinct datasets (i.e., DARPA and a commercial bank penetration test). The results are satisfactory, which shows that the proposed hybrid system can be an efficient solution in the DDoS detection process. We also state that more sophisticated DDoS attacks may evade the signature-based detector rules, which are commonly known, and the system performance may decrease as the detection success solely depends on the anomaly detector. Also the training need of anomaly detector stands as a limitation on the overall system performance. The training data may not reflect the real network model in a practical system or even may be unavailable, which may result in decreased performance. Improvements to the present system, including the enhancement of aforementioned limitations, are left as future work.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work was supported in part by the ITEA2 Project ADAX and the TUBITAK under Grant no. 9130016.

References

[1] Snort, http://www.snort.org/.

[2] Y. Tao and S. Yu, "DDoS attack detection at local area networks using information theoretical metrics," in *Proceedings of the 12th IEEE International Conference on Trust, Security and Privacy in Computing and Comm*, pp. 233–240, Melbourne, Australia, July 2013.

[3] S. Yu, W. Zhou, R. Doss, and W. Jia, "Traceback of DDoS attacks using entropy variations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 3, pp. 412–425, 2012.

[4] S. Yu, W. Zhou, and R. Doss, "Information theory based detection against network behavior mimicking DDoS attacks," *IEEE Communications Letters*, vol. 12, no. 4, pp. 318–321, 2008.

[5] Y. Xiang, K. Li, and W. Zhou, "Low-rate DDoS attacks detection and traceback by using new information metrics," *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 2, pp. 426–437, 2011.

[6] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Information metrics for low-rate DDoS attack detection: a comparative evaluation," in *Proceedings of the 7th International Conference on Contemporary Computing (IC3 '14)*, pp. 80–84, IEEE, Noida, India, August 2014.

[7] Y. Xiang, Y. Lin, W. L. Lei, and S. J. Huang, "Detecting DDOS attack based on network self-similarity," *IEE Proceedings: Communications*, vol. 151, no. 3, pp. 292–295, 2004.

[8] A. Chonka, J. Singh, and W. Zhou, "Chaos theory based detection against network mimicking DDoS attacks," *IEEE Communications Letters*, vol. 13, no. 9, pp. 717–719, 2009.

[9] Y. Chen, X. Ma, and X. Wu, "DDoS detection algorithm based on preprocessing network traffic predicted method and chaos theory," *IEEE Communications Letters*, vol. 17, no. 5, pp. 1052–1054, 2013.

[10] X. Wu and Y. Chen, "Validation of chaos hypothesis in NADA and improved DDoS detection algorithm," *IEEE Communications Letters*, vol. 17, no. 12, pp. 2396–2399, 2013.

[11] B. Joshi, A. S. Vijayan, and B. K. Joshi, "Securing cloud computing environment against DDoS attacks," in *Proceedings of the International Conference on Computer Communication and Informatics (ICCCI '12)*, pp. 1–5, IEEE, Coimbatore, India, January 2012.

[12] V. L. Thing, M. Sloman, and N. Dulay, "Locating network domain entry and exit point/path for DDoS attack traffic," *IEEE Transactions on Network and Service Management*, vol. 6, no. 3, pp. 163–174, 2009.

[13] Y. Xie and S.-Z. Yu, "Monitoring the application-layer DDoS sttacks for popular websites," *IEEE/ACM Transactions on Networking*, vol. 17, no. 1, pp. 15–25, 2009.

[14] H. Luo, Y. Lin, H. Zhang, and M. Zukerman, "Preventing DDoS attacks by identifier/locator separation," *IEEE Network*, vol. 27, no. 6, pp. 60–65, 2013.

[15] M. Barati, A. Abdullah, N. I. Udzir, R. Mahmod, and N. Mustapha, "Distributed Denial of Service detection using hybrid machine learning technique," in *Proceedings of the 4th International Symposium on Biometrics and Security Technologies (ISBAST '14)*, pp. 268–273, Kuala Lumpur, Malaysia, August 2014.

[16] S. Yu, Y. Tian, S. Guo, and D. O. Wu, "Can we beat DDoS attacks in clouds?" *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 9, pp. 2245–2254, 2014.

[17] T. Thapngam, S. Yu, W. Zhou, and G. Beliakov, "Discriminating DDoS attack traffic from flash crowd through packet arrival patterns," in *Proceedings of the IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS '11)*, pp. 952–957, Shanghai, China, April 2011.

[18] K. Hwang, M. Cai, Y. Chen, and M. Qin, "Hybrid intrusion detection with weighted signature generation over anomalous internet episodes," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 1, pp. 41–55, 2007.

[19] J. Gómez, C. Gil, N. Padilla, R. Baños, and C. Jiménez, "Design of a snort-based hybrid intrusion detection system," in *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, pp. 515–522, Springer, Berlin, Germany, 2009.

[20] R. A. Redner and H. F. Walker, "Mixture densities, maximum likelihood and the EM algorithm," *SIAM Review*, vol. 26, no. 2, pp. 195–239, 1984.

[21] G. J. McLachlan and T. Krishnanl, *The EM Algorithm and Extensions*, Wiley Series in Probability and Statistics, 2nd edition, 2008.

[22] J. Bilmes, "A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models," Tech. Rep. TR-97-021, International Computer Science Institute (ICSI), 1997.

[23] J. Rennie, "A short tutorial on using expectation-maximization with mixture models," 2004, http://people.csail.mit.edu/jrennie/writing/mixtureEM.pdf.

[24] E. Albin and N. C. Rowe, "A realistic experimental comparison of the Suricata and Snort intrusion-detection systems," in *Proceedings of the 26th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA '12)*, pp. 122–127, Fukuoka, Japan, March 2012.

[25] MIT Lincoln Laboratory, Information Systems Technology, 2015, http://www.ll.mit.edu/ideval/data/2000data.html.

Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2016, Article ID 3095971, 6 pages
http://dx.doi.org/10.1155/2016/3095971

Research Article

SVM Intrusion Detection Model Based on Compressed Sampling

Shanxiong Chen,[1] Maoling Peng,[2] Hailing Xiong,[1] and Xianping Yu[1]

[1]*College of Computer and Information Science, Southwest University, Chongqing 400715, China*
[2]*Chongqing City Management Vocational College, Chongqing 400055, China*

Correspondence should be addressed to Shanxiong Chen; csxpml@163.com

Received 2 October 2015; Accepted 20 January 2016

Academic Editor: Michele Vadursi

Intrusion detection needs to deal with a large amount of data; particularly, the technology of network intrusion detection has to detect all of network data. Massive data processing is the bottleneck of network software and hardware equipment in intrusion detection. If we can reduce the data dimension in the stage of data sampling and directly obtain the feature information of network data, efficiency of detection can be improved greatly. In the paper, we present a SVM intrusion detection model based on compressive sampling. We use compressed sampling method in the compressed sensing theory to implement feature compression for network data flow so that we can gain refined sparse representation. After that SVM is used to classify the compression results. This method can realize detection of network anomaly behavior quickly without reducing the classification accuracy.

1. Introduction

With the rapid development of network technology, various Internet-based technologies are widely applied in various industries, leading to great improvement of productive forces. People are enjoying convenience and efficiency brought about by network, and a variety of potential threats are jeopardizing the security of network communication at the same time. At the beginning of network design, people paid more attention to data transmission efficiency and communication convenience and paid less attention to the security of network protocol [1]. Many network protocols are lacking secure communication mechanism; thus, there are naturally a lot of security vulnerabilities in Internet based on these network protocols [2, 3]. With the development of e-commerce, e-government affairs, and other businesses having high demand for security, a variety of network-based security communication protocols appeared, but these protocols are based on TCP/IP architecture, which is a kind of unsafe open system from the basic communication layer [4]. The existing attack techniques and technologies have unceasingly developed with the enhancement of security technology, so in the case of all kinds of inevitable network threats, a current research hotspot on network security is to timely and correctly detect security threats and to take appropriate treatment, so as to reduce the loss caused by network attacks [5–7].

Compressed sensing is a new data processing theory; there are many important applications in medical image [8] and signal processing [9], communications [10], harmonic detection [11], and so forth. Data acquisition and processing method of compressed sensing theory give rise to great performance improvement of intrusion detection technology [12, 13]. Currently, massive data processing is the performance bottleneck of network software and hardware equipment. In the phase of data acquisition, if the dimension of data can be reduced and characteristic information of network data can be directly obtained, the efficiency of the detection will be greatly improved [14, 15]. SVM intrusion detection technology based on compressed sensing uses the compressed sampling technology of compressed sensing theory to get a small amount of data concerning network behavior characteristics and then uses the support vector machine (SVM) to establish an intrusion detection model, so as to realize rapid judgment of intrusion behavior.

2. Compressed Sensing Theory

If there are only K nonzero elements in a discrete signal, the signal is considered to be K sparse. In view of a nonsparse

discrete signal u, the signal can obtain the sparse or nearly sparse representation in the condition of a proper sparse base $\Psi \in R^{N\times L}$:

$$u = \Psi x. \tag{1}$$

x is the sparse or nearly sparse representation of signal u. According to the CS (compressed sensing) theory, the sampling process of discrete signal is described as below: The signal u with a length of N is projected M times on the sensing matrix Φ $\{\Phi_i, i = 1, 2, \ldots, M\}$, and then the compressed form of the signal can be obtained [16]. Its expression is $y_i = \Phi_i^T u, i = 1, 2, \ldots, M$. In order to improve the efficiency of sampling, the frequency of sampling should be reduced as much as possible; usually, $M < N$. It can be seen that the length of y is less than that of u, so it is called compressed sensing. It is different from traditional data acquisition method that includes acquisition, compression, transmission, and decompression; the compressed sensing theory merely collects the information that best represents data characteristic rather than obtaining a complete signal and high resolution images. Compressed sampling method saves storage space and reduces transmission cost to a great extent. The biggest difference between compressed sensing and traditional data sampling mode is that compressed sensing has realized the compression in the process of data acquisition and reconstruction in the later phase; the traditional mode is to collect complete data information first and then to compress data for storage and transmission. Therefore, the CS theory provides an undersampling mode for data acquisition and can get information in the slower rate compared to Nyquist. The mathematical model of compressed sensing is expressed as below.

For signal $u \in R^{N\times 1}$, find a linear measurement matrix $\Phi \in R^{M\times N}$ $(M < N)$ for projection algorithm

$$y = \Phi u, \tag{2}$$

where

$$\Phi = \begin{bmatrix} \Phi_1^T \\ \Phi_2^T \\ \cdots \\ \Phi_M^T \end{bmatrix},$$

$$x = \begin{bmatrix} u_1 \\ u_2 \\ \cdots \\ u_N \end{bmatrix}, \tag{3}$$

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_M \end{bmatrix}.$$

y represents the collected signals. The crux of the problem is to recover signal u from signal y, and Φ is not a square matrix $(M < N)$, so it gets involved in a problem of solving an underdetermined equation. And u to be solved can have a solution set. Furthermore, the compressed sensing theory shows that, under the specific conditions, u is the uniqueness solution, and this solution is obtained through reconstructing y that is acquired by compressed sampling [17, 18].

Equation (2) shows the signal sampling mode, and the CS theory suggests that the solution of (2) must ensure that x is sparse, so as to solve the equation through $L0$ norm minimization problem. In reality, most of the signals are not sparse. The existing theory shows that when a signal is projected on the orthogonal transformation matrix, the absolute value of most transform coefficients is small [19], and the obtained transform vector is sparse or approximately sparse, which is considered as a concise expression of original signal, a prior condition of compressed sensing; namely, the signal must have a sparse representation under some type of transformation. Therefore, sparse transformation base Ψ is established, and the sparse representation of nonsparse signals is completed according to (1). Combined with (1) and (2), compression sampling of the signal u can be described as below: equation (2) is used for compressed sampling of the signal u to obtain y, and then (4) is used for x sparse solution; ultimately, x is used for sparse inverse transformation, so as to reconstruct the signal u. Consider

$$y = \Phi\Psi x = \Theta x, \tag{4}$$

where $\Theta = \Phi\Psi$, which is still an underdetermined equation; however, under certain constraints, y is used to solve x. Of course, if the signal is sparse, there is no need for sparse transformation; at this point $\Theta = \Phi$. In compressed sensing, the signal needs to meet the conditions; one constraint condition is sparse representation, and the other important one is to satisfy the RIP (Restricted Isometry Property) [20]; namely, there is a restricted isometry constant δ_s for the matrix Θ.

δ_s is defined as the minimum value to make the equation true. Consider

$$(1 - \delta_s)\|v_s\|_2^2 \le \|\Theta v_s\|_2^2 \le (1 + \delta_s)\|v_s\|_2^2. \tag{5}$$

Herein v_s represents s-order sparse vector.

3. SVM Intrusion Detection Model Based on Compressed Sensing

The SVM intrusion detection method based on compressed sensing is to carry out compressed sampling of the tagged training dataset, so as to obtain compressed characteristic data and then to input it into SVM classifier for training, so as to obtain the classification model. In the detection phase, carry out compressed sampling of the untagged dataset, and then reuse the built SVM classification model to classify data, to obtain normal or abnormal access behaviors, and then reconstruct the detected data of normal behaviors, to obtain the complete normal network data flow.

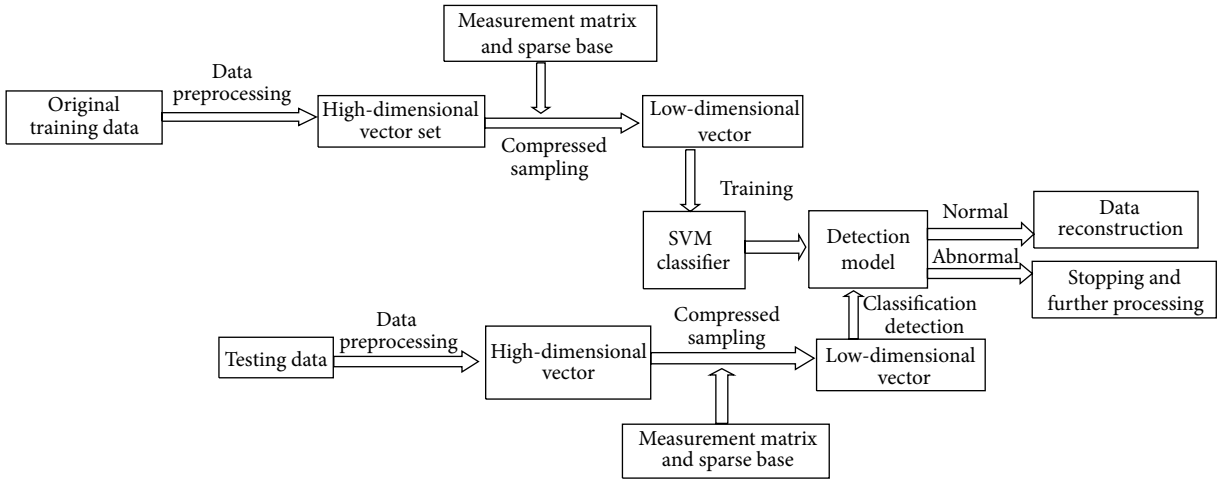


Figure 1: Intrusion detection process based on compressed sensing.

As shown in Figure 1, the steps for intrusion detection based on compressed sensing include the following:

(1) Pretreatment of dataset: the compressed sensing theory is to directly sample vector data, so training data and testing data should be expressed in the form of vector.

(2) Selection of proper measurement matrix and sparse matrix: measurement matrix and sparse base should meet the conditions of RIP, and data resulting from their compressed sampling must effectively express the original data at the same time.

(3) Construction of the SVM classifier: the SVM classifier can use compressed sampling to obtain low-dimensional data, so as to complete classification training, and testing dataset has high detection precision.

(4) After performing detection, if network access is normal, the reconstruction algorithm is used to restore detection data to full form before sampling.

4. Experiments and Analysis

The experiments used KDD CUP99 dataset. The dataset was collected in a network environment which was established in MIT Lincoln Laboratory, simulating local area network (LAN) of the US Air Force. It includes 9-week TCP dump network connections and system audit data, simulating various types of users, network traffic, and attack technique.

The compressed sensing theory request data must be expressed in the form of vector; therefore, each nonnumerical attribute must be converted into a numerical value, and herein the numerical value can simply replace category attributes. Furthermore, in order to eliminate the influence of characteristic dimension on the experimental results, continuous data need to be standardized. The following equation is used for standardization. $S = \{s_{ij} \mid i = 1, \ldots, N,\ j = 1, \ldots, D\}$ is input data, N represents the number of sample datasets, and D represents the characteristic digit of sample data, μ represents mean value, and σ is standard deviation of the sample. Therefore, the normalization expression of sample data is as below:

$$S'_{ij} = \frac{S_{ij} - \mu}{\sigma}, \tag{6}$$

where

$$\mu = \frac{1}{N}\sum_{i=1}^{N} S_{ij},$$
$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left(X_{ij} - \mu\right)^2}. \tag{7}$$

In order to clearly observe the experimental results, we introduced the following indicators for detection performance.

Detection rate refers to the ratio between the number of correct attack datasets detected in the testing set and the number of total actual attack datasets; that is, the equation is as below:

$$\mathrm{DR} = \frac{\text{the number of attacks detected}}{\text{the number of attacks}}\%. \tag{8}$$

False positive rate refers to the ratio between the number of attack datasets being identified by mistake after the test set is detected through the algorithm and the number of total attack datasets detected; that is, the equation is as below:

$$\mathrm{FPR} = \frac{\text{the number of false positive}}{\text{false positive + true positive}}\%. \tag{9}$$

The experiments adopted 10% training subsets of the KDD CUP99 dataset as training data of the classifier and the test subset tagged correct as test data. We first considered the classifier learning and detection without compressed sampling and then carried out compressed sampling of

Table 1: Detection result of the support vector machine (SVM) as the classifier.

Detection type		Normal	Probe	Dos	U2R	R2L
Noncompressed sampling	Detection rate (%)	98.73	97.34	99.27	94.21	99.35
	False positive rate (%)	0.87	1.03	0.92	1.08	0.92
Compressed sampling						
Gaussian random matrix	Detection rate (%)	98.23	96.42	97.08	90.37	97.64
	False positive rate (%)	0.82	1.19	0.96	1.26	0.98
Random Bernoulli matrix	Detection rate (%)	97.31	97.07	99.14	88.39	98.71
	False positive rate (%)	1.13	1.21	1.07	1.86	0.92
Partial Hadamard measurement matrix	Detection rate (%)	98.12	96.94	98.71	90.84	97.75
	False positive rate (%)	0.93	1.05	0.92	1.49	1.04
Toeplitz matrix measurement	Detection rate (%)	97.86	96.93	98.49	89.15	98.73
	False positive rate (%)	0.88	1.06	0.91	2.12	0.94
Structure random matrix	Detection rate (%)	96.57	96.72	97.87	87.35	98.76
	False positive rate (%)	1.15	1.23	0.97	2.34	0.96
Chirp measurement matrix	Detection rate (%)	97.33	96.24	98.39	90.08	99.01
	False positive rate (%)	1.08	1.33	1.15	1.13	0.94

(i) Probe: surveillance or probe, (ii) DoS: Denial of Service, (iii) U2R: User to Root, and (iv) R2L: Remote to Local.

training data and test data and input it into the SVM classifier for learning and detection. The experimental procedure is as follows:

(1) We extract test data from 10% training subsets of the KDD CUP99 dataset. Compressed sampling is to deal with numeric data; it is required to convert nonnumeric data into numeric data in the procedure, except data with attack attribute; namely, attack-type data cannot be converted to numeric data; otherwise, attack-type data cannot be recognized. So the corresponding relation between attribute value of attack-type data and each record should be kept.

(2) The KDD CUP99 dataset refers to normal and attack data collected for a long time; from the perspective of the entire data sample, there is a small number of attack datasets, so the standardized dataset is a sparse set, the formed matrix is a sparse matrix, and there is no need for sparsification.

(3) In the experiment, multiple measurement matrices are directly used for compressed sampling of training set: Gaussian random matrix, random Bernoulli matrix, partial Hadamard measurement matrix, Toeplitz measurement matrix, structure random matrix, and Chirp measurement matrix.

(4) The compressed data obtained are, respectively, input into the SVM-constructed classifier for training, thus forming a training model.

(5) The corrected subsets of KDD CUP99 dataset are selected as test set. Herein carry out normalized conversion of test data; namely, convert nonnumeric data into numerical data for compressing sampling, and then input compressed data into the training model for detection.

Table 1 shows the results of using the SVM-constructed classifier for intrusion detection of different sampling matrixes through 30 times of sampling.

Table 1 showed the results of using the compressed sampling method for intrusion detection and of using the noncompressed sampling method to directly input it into the classifier for training and detection. It can be seen from Table 1 that the results obtained through two methods were similar. For the traditional method of compressed sampling, the detection rate of five types, that is, Normal, Probe, Dos, U2R and R2L, was more than 98% except that of U2R. Under the condition of compressed sensing, the detection rate obtained by using different sampling matrixes was around 98%. Only the detection rate obtained by using the compressed sampling method for U2R attack-type data was lower, and the false positive rate was higher. After further analysis, it was found that the traditional method for U2R attack-type data had a low detection rate, which was related to dataset itself, less U2R type data, and deviation of the training model. In reality, the use of compressed sensing method cannot greatly improve the detection rate but can increase the efficiency of training and detection by reducing dimension data. Figure 2 showed training and detection time under different sampling matrixes after 30 times of sampling.

Figure 2 showed that, after the method of compressed sampling was used for information processing of KDD CUP99 dataset, time used for training and inspection was reduced. In particular, it can be found that for data obtained through using the Gaussian random matrix for compressed sampling its training and detection time is decreased greatly for four classifiers of detection.

Besides attack-type attribute, the KDD CUP99 dataset has 41D characteristics, and matrixes are used for compressed sampling of 41D characteristics. However, compressed sampling also affects detection precision. In theory, the higher the degree of compression is, the shorter the model's training and detection time is, affecting detection precision. In further

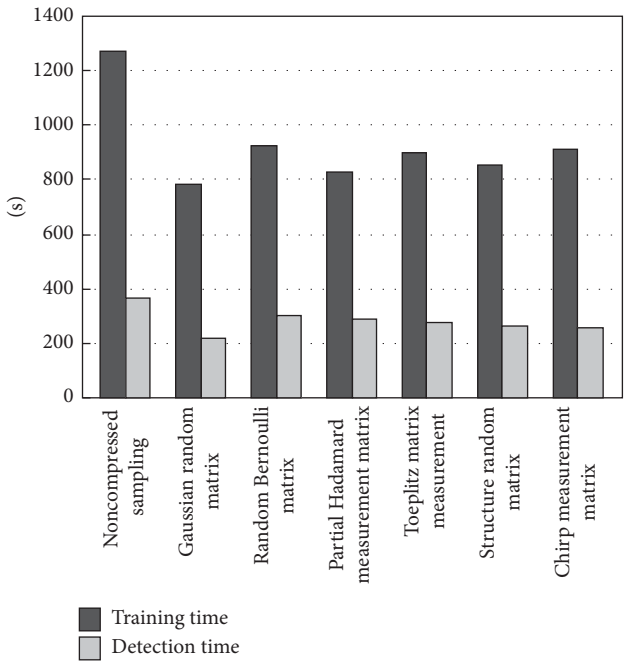


Figure 2: Training and detection time.

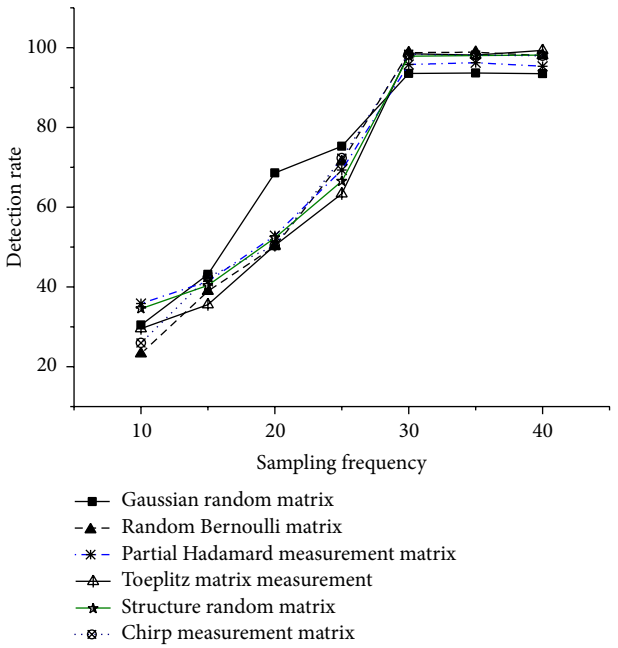


Figure 3: Detection rate of the K-nearest neighbor algorithm as a classifier for DoS attack.

experiments, we analyzed the relationship between compression degree and detection precision. For convenience of representation, the detection rate for DoS attack was selected for analysis. In the experiments, there are six sampling matrixes, respectively, under different sampling frequencies, so SVM method was used to obtain the detection rate of the corrected dataset according to sampling matrixes. Sampling frequencies were $10, 15, 20, \ldots, 40$. The dimension of KDD CUP99 data was 41D, so there was basically no compressed sampling under the condition of sampling frequency of 40.

It can be seen from Figure 3 that, under the condition of low sampling frequency, the SVM-constructed classifier had lower DoS detection precision. Based on the compressed sensing theory, in order to perfectly express data, sampling frequency M must have particular relations; that is, $M \geq C \cdot k \cdot \log N$. In general, four times of data sparseness are selected as the sampling frequency. The figure showed that, under the low sampling frequency, the detection rate was lower; with the increase of sampling frequency, the detection rate was increased accordingly. When the sampling frequency was up to 30 or so, the detection rate tended to be stable. At this time, the low-dimensional data obtained through compressed sampling of KDD CUP99 dataset can effectively express the original high-dimensional data. Thus, DoS detection rate is approximate to the detection rate obtained by using the method of noncompressed sampling. However, with the further increase of sampling frequency, there was no significant change of detection rate.

5. Conclusion

Intrusion detection needs to deal with massive network data, leading to low detection efficiency. In the paper, the compressed sensing technology was applied to realize network data compression, and the SVM method was used to anomalously detect the compressed data. We have arrived at a conclusion that, relative to direct use of the classifier for learning and detection of training set and testing set, the intrusion detection model established through compressed sensing had no significant change of detection rate and false positive rate, but training and detection time was greatly reduced, which is the key to detect network data flow. A large number of network datasets need rapid and real-time detection, so it can be seen that intrusion detection based on compressed sensing has provided a real-time network security protection mechanism.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (61303227), the Fundamental Research Funds for the Central Universities (XDJK2014C039, XDJK2016C045), and Postdoctoral Fund of Southwestern University (swu1114033).

References

[1] A. P. Lauf, R. A. Peters, and W. H. Robinson, "A distributed intrusion detection system for resource-constrained devices in ad-hoc networks," *Ad Hoc Networks*, vol. 8, no. 3, pp. 253–266, 2010.

[2] S. Raza, L. Wallgren, and T. Voigt, "SVELTE: Real-time intrusion detection in the Internet of Things," *Ad Hoc Networks*, vol. 11, no. 8, pp. 2661–2674, 2013.

[3] T. Zhang, E. Lee, and J. K. Seo, "Anomaly depth detection in trans-admittance mammography: a formula independent of anomaly size or admittivity contrast," *Inverse Problems*, vol. 30, no. 4, Article ID 045003, 2014.

[4] D.-S. Pham, S. Venkatesh, M. Lazarescu, and S. Budhaditya, "Anomaly detection in large-scale data stream networks," *Data Mining and Knowledge Discovery*, vol. 28, no. 1, pp. 145–189, 2014.

[5] R. Puzis, M. Tubi, Y. Elovici, C. Glezer, and S. Dolev, "A decision support system for placement of intrusion detection and prevention devices in large-scale networks," *ACM Transactions on Modeling and Computer Simulation*, vol. 22, no. 1, 2011.

[6] S. Chen, S. Wu, Y. Cao, and D. Tang, "An intrusion detection model based on non-negative matrix factorization," *Applied Mechanics and Materials*, vol. 148-149, pp. 895–899, 2012.

[7] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: a survey," *ACM Computing Surveys*, vol. 41, no. 3, article 15, 2009.

[8] J. Haupt, W. U. Bajwa, G. Raz, and R. Nowak, "Toeplitz compressed sensing matrices with applications to sparse channel estimation," *IEEE Transactions on Information Theory*, vol. 56, no. 11, pp. 5862–5875, 2010.

[9] L. B. Montefusco, D. Lazzaro, S. Papi, and C. Guerrini, "A fast compressed sensing approach to 3D MR image reconstruction," *IEEE Transactions on Medical Imaging*, vol. 30, no. 5, pp. 1064–1075, 2011.

[10] Y. Wiaux, G. Puy, and P. Vandergheynst, "Compressed sensing reconstruction of a string signal from interferometric observations of the cosmic microwave background," *Monthly Notices of the Royal Astronomical Society*, vol. 402, no. 4, pp. 2626–2636, 2010.

[11] F. Bonavolontà, M. D'Apuzzo, A. Liccardo, and G. Miele, "Harmonic and interharmonic measurements through a compressed sampling approach," *Measurement*, vol. 77, pp. 1–15, 2016.

[12] D. L. Donoho, "Compressed sensing," *IEEE Transactions on Information Theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[13] D. L. Donoho, A. Maleki, and A. Montanari, "The noise-sensitivity phase transition in compressed sensing," *IEEE Transactions on Information Theory*, vol. 57, no. 10, pp. 6920–6941, 2011.

[14] R. Bhargavi and V. Vaidehi, "Semantic intrusion detection with multisensor data fusion using complex event processing," *Sadhana—Academy Proceedings in Engineering Sciences*, vol. 38, no. 2, pp. 169–185, 2013.

[15] W. J. An and M. G. Liang, "A new intrusion detection method based on SVM with minimum within-class scatter," *Security and Communication Networks*, vol. 6, no. 9, pp. 1064–1074, 2013.

[16] E. J. Candès, "The restricted isometry property and its implications for compressed sensing," *Comptes Rendus Mathematique*, vol. 346, no. 9-10, pp. 589–592, 2008.

[17] K. Takeda and Y. Kabashima, "Statistical mechanical assessment of a reconstruction limit of compressed sensing: toward theoretical analysis of correlated signals," *EPL*, vol. 95, no. 1, Article ID 18006, 2011.

[18] L. Zelnik-Manor, K. Rosenblum, and Y. C. Eldar, "Sensing matrix optimization for block-sparse decoding," *IEEE Transactions on Signal Processing*, vol. 59, no. 9, pp. 4300–4312, 2011.

[19] Y. Zhao, Y. H. Hu, and H. Wang, "Enhanced random equivalent sampling based on compressed sensing," *IEEE Transactions on Instrumentation and Measurement*, vol. 61, no. 3, pp. 579–586, 2012.

[20] E. J. Candes and Y. Plan, "A probabilistic and RIPless theory of compressed sensing," *IEEE Transactions on Information Theory*, vol. 57, no. 11, pp. 7235–7254, 2011.

Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2016, Article ID 8034967, 17 pages
http://dx.doi.org/10.1155/2016/8034967

Research Article

Detection and Visualization of Android Malware Behavior

Oscar Somarriba,[1,2] Urko Zurutuza,[1] Roberto Uribeetxeberria,[1] Laurent Delosières,[3] and Simin Nadjm-Tehrani[4]

[1] *Electronics and Computing Department, Mondragon University, 20500 Mondragon, Spain*
[2] *National University of Engineering (UNI), P.O. Box 5595, Managua, Nicaragua*
[3] *ExoClick SL, 08005 Barcelona, Spain*
[4] *Department of Computer and Information Science, Linköping University, 581 83 Linköping, Sweden*

Correspondence should be addressed to Oscar Somarriba; oscar.somarriba@gmail.com

Received 7 November 2015; Accepted 7 February 2016

Academic Editor: Aniket Mahanti

Malware analysts still need to manually inspect malware samples that are considered suspicious by heuristic rules. They dissect software pieces and look for malware evidence in the code. The increasing number of malicious applications targeting Android devices raises the demand for analyzing them to find where the malcode is triggered when user interacts with them. In this paper a framework to monitor and visualize Android applications' anomalous function calls is described. Our approach includes platform-independent application instrumentation, introducing hooks in order to trace restricted API functions used at runtime of the application. These function calls are collected at a central server where the application behavior filtering and a visualization take place. This can help Android malware analysts in visually inspecting what the application under study does, easily identifying such malicious functions.

1. Introduction

Collecting a large amount of data issued by applications for smartphones is essential for making statistics about the applications' usage or characterizing the applications. Characterizing applications might be useful for designing both an anomaly-detection system and/or a misuse detecting system, for instance.

Nowadays, smartphones running on an Android platform represent an overwhelming majority of smartphones [1]. However, Android platforms put restrictions on applications for security reasons. These restrictions prevent us from easily collecting traces without modifying the firmware or rooting the smartphone. Since modifying the firmware or rooting the smartphone may void the warranty of the smartphone, this method cannot be deployed on a large scale.

From the security point of view, the increase in the number of internet-connected mobile devices worldwide, along with a gradual adoption of LTE/4G, has drawn the attention of attackers seeking to exploit vulnerabilities and mobile infrastructures. Therefore, the malware targeting smartphones has grown exponentially. Android malware is one of the major security issues and fast growing threats facing the Internet in the mobile arena, today. Moreover, mobile users increasingly rely on unofficial repositories in order to freely install paid applications whose protection measures are at least dubious or unknown. Some of these Android applications have been uploaded to such repositories by malevolent communities that incorporate malicious code into them. This poses strong security and privacy issues both to users and operators. Thus, further work is needed to investigate threats that are expected due to further proliferation and connectivity of gadgets and applications for smart mobile devices.

This work focuses on monitoring Android applications' suspicious behavior at runtime and visualizing their malicious functions to understand the intention behind them. We propose a platform-independent behavior monitoring infrastructure composed of four elements: (i) an Android application that guides the user in selecting, instrumenting, and monitoring of the application to be examined, (ii) an embedded client that is inserted in each application to be

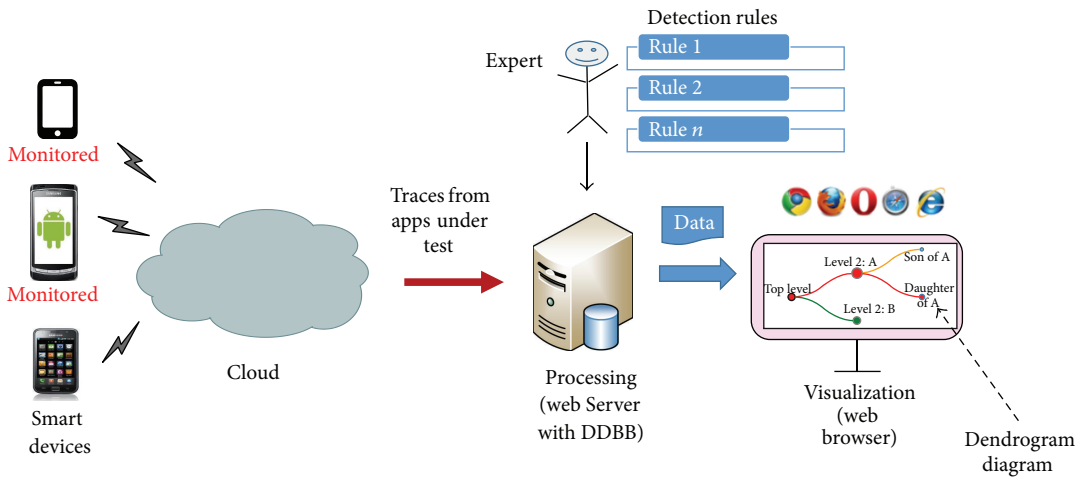


Figure 1: Overview of the monitoring system.

monitored, (iii) a cloud service that collects the application to be instrumented and also the traces related to the function calls, (iv) and finally a visualization component that generates behavior-related dendrograms out of the traces. A dendrogram [2] consists of many U-shaped nodes-lines that connect data of the Android application (e.g., the package name of the application, Java classes, and methods and functions invoked) in a hierarchical tree. As a matter of fact, we are interested in the functions and methods which are frequently seen in malicious code. Thus, malicious behavior could be highlighted in the dendrogram based on a predefined set of anomaly rules. An overview of the monitoring system is shown in Figure 1.

Monitoring an application at runtime is essential to understand how it interacts with the device, with key components such as the provided application programming interfaces (APIs). An API specifies how some software components (routines, protocols, and tools) should act when subject to invocations by other components. By tracing and analyzing these interactions, we are able to find out how the applications behave, handle sensitive data, and interact with the operating system. In short, Android offers a set of API functions for applications to access protected resources [3].

The remainder of the paper is organized as follows. Section 2 provides the notions behind the components used in the rest of the paper. Next, the related work is discussed in Section 3. Next we describe the monitoring and visualization architecture in Section 4, while we provide the details of the implementational issues of our system in Section 5. Later, in Section 6, we evaluate the proposed infrastructure and the obtained results by using 8 malware applications. Limitations and Conclusions are presented in Sections 7 and 8, respectively.

2. Background

Web Services extend the World Wide Web infrastructure to provide the means for software to connect to other software applications [4]. RESTFul Web Services are Web Services that use the principles of REpresentational State Transfer (REST) [5]. In other words, they expose resources to clients that can be accessed through the Hypertext Transfer Protocol (HTTP).

Regarding the Android operating system (OS), it is divided into four main layers: applications, application framework, middleware, and Linux kernel.

(i) Applications. The top layer of the architecture is where the applications are located. An Android application is composed of several components, amongst which we have Activities and Services. Activities provide a user interface (UI) of the application and are executed one at a time, while Services are used for background processing such as communication, for instance.

(ii) Application Framework. This layer is a suite of Services that provides the environment in which Android applications run and are managed. These programs provide higher-level Services to applications in the form of Java classes.

(iii) Middleware. This layer is composed of the Android runtime (RT) and C/C++ libraries. The Android RT is, at the same time, composed of the Dalvik Virtual Machine (DVM) (Android version 4.4. launches a new virtual machine called Android runtime (ART). ART has more advanced performance than DVM, among other things, by means of a number of new features such as the ahead-of-time (OTA) compilation, enhanced garbage collection, improved application debugging, and more accurate high-level profiling of the apps [6]) and a set of native (core) Android functions. The DVM is a key part of Android as it is the software where all applications run on Android devices. Each application that is executed on Android runs on a separate Linux process with an individual instance of the DVM, meaning that multiple instances of the DVM exist at the same time. This is managed by the Zygote process, which generates a fork of the parent DVM instance with the core libraries whenever it receives a request from the runtime process.

(iv) Linux Kernel. The bottom layer of the architecture is where the Linux kernel is located. This provides basic system functionality like process and memory management.

The kernel also handles a set of drivers for interfacing Android and interacting with the device hardware.

In standard Java environments, Java source code is compiled into Java bytecode, which is stored within .class format files. These files are later read by the Java Virtual Machine (JVM) at runtime. On Android, on the other hand, Java source code that has been compiled into .class files is converted to .dex files, frequently called Dalvik Executable, by the "dx" tool. In brief, the .dex file stores the Dalvik bytecode to be executed on the DVM.

Android applications are presented on an Android application package file (APK) .apk, the container of the application binary that contains the compiled .dex files and the resource files of the app. In this way, every Android application is packed using zip algorithm. An unpacked app has the following structure (several files and folders) [7]:

(i) an *AndroidManifest.xml* file: it contains the settings of the application (meta-data) such as the permissions required to run the application, the name of the application, definition of one or more components such as Activities, Services, Broadcasting Receivers, or Content Providers. Upon installing, this file is read by the *PackageManager*, which takes care of setting up and deploying the application on the Android platform.

(ii) a *res* folder: it contains the resources used by the applications. By resources, we mean the app icon, its strings available in several languages, images, UI layouts, menus, and so forth.

(iii) an *assets* folder: it stores noncompiled resources. This is a folder containing applications assets, which can be retrieved by *AssetManager*.

(iv) a *classes.dex* file: it stores the classes compiled in the dex file format to be executed on the DVM.

(v) a *META-INF* folder: this directory includes *MANIFEST.MF* which contains a cryptographic signature of the application developer certificate to validate the distribution.

The resulting .apk file is signed with a keystore to establish the identity of the author of the application. Besides, to build Android applications, a software developer kit (SDK) is usually available allowing access to APIs of the OS [8]. Additionally, two more components are described in order to clarify the background of this work: the Android-apktool [9] and the Smali/Backsmali tools. The Android-apktool is generally used to unpack and disassemble Android applications. It is also used to assemble and pack them. It is a tool set for reverse engineering third party Android apps that simplifies the process of assembling and disassembling Android binary .apk files into Smali .smali files and the application resources to their original form. It includes the Smali/Baksmali tools, which can decode resources (i.e., .dex files) to nearly original form of the source code and rebuild them after making some modifications. This enables all these assembling/disassembling operations to be performed automatically in an easy yet reliable way.

However, it is worth noting that the repackaged Android binary .apk files can only possess the same digital signature if the original keystore is used. Otherwise, the new application will have a completely different digital signature.

3. Related Work

Previous works have addressed the problem of understanding the Android application behavior in several ways. An example of inspection mechanisms for identification of malware applications for Android OS is presented by Karami et al. [10] where they developed a transparent instrumentation system for automating the user interactions to study different functionalities of an app. Additionally, they introduced runtime behavior analysis of an application using input/output (I/O) system calls gathered by the monitored application within the Linux kernel. Bugiel et al. [11] propose a security framework named *XManDroid* that extends the monitoring mechanism of Android, in order to detect and prevent application-level privilege escalation attacks at runtime based on a given policy. The principal disadvantage of this approach is that the modified framework of Android has to be ported for each of the devices and Android versions in which it is intended to be implemented. Unlike [10, 11], we profile only at the user level and therefore we do not need to root or to change the framework of Android smartphones if we would like to monitor the network traffic, for example.

Other authors have proposed different security techniques regarding permissions in Android applications. For instance, Au et al. [12] present a tool to extract the permission specification from Android OS source code. Unlike the other methods, the modules named Dr. Android and Mr. Hide that are part of a proposed and implemented app by Jeon et al. [13] do not intend to monitor any smart phones. They aim at refining the Android permissions by embedding a module inside each Android application. In other words, they can control the permissions via their module. We also embed a module inside each Android application but it is used to monitor the Android application instead.

In the work by Zhang et al. [3], they have proposed a system called *VetDroid* which can be described as a systematic analysis technique using an app's permission use. By using real-world malware, they identify the callsites where the app requests sensitive resources and how the obtained permission resources are subsequently utilized by the app. To do that, *VetDroid* intercepts all the calls to the Android API and synchronously monitors permission check information from Android permission enforcement system. In this way, it manages to reconstruct the malicious (permission use) behaviors of the malicious code and to generate a more accurate permission mapping than PScout [12]. Briefly this system [3] applies dynamic taint analysis to identify malware. Different from *VetDroid*, we do not need to root or jailbreak the phone nor do we conduct the permission-use approach for monitoring the smartphone.

Malware detection (MD) techniques for smart devices can be classified according to how the code is analyzed, namely, static analysis and dynamic analysis. In the former case, there is an attempt to identify malicious code by

decompiling/disassembling the application and searching for suspicious strings or blocks of code; in the latter case the behavior of the application is analyzed using execution information. Examples of the two named categories are *Dendroid* [2] as an example of a static MD for Android OS devices and *Crowdroid* as a system that clusters system call frequency of applications to detect malware [14]. Also, hybrid approaches have been proposed in the literature for detection and mitigation of Android malware. For example, Patel and Buddhadev [15] combine Android applications analysis and machine learning (ML) to classify the applications using static and dynamic analysis techniques. Genetic algorithm based ML technique is used to generate a rules-based model of the system.

A thorough survey by Jiang and Zhou [16] charts the most common types of permission violations in a large data set of malware. Furthermore, in [17], a learning-based method is proposed for the detection of malware that analyzes applications automatically. This approach combines static analysis with an explicit feature map inspired by a linear-time graph kernel to represent Android applications based on their function call graphs. Also, Arp et al. [18] combine concepts from broad static analysis (gathering as many features of an application as possible) and machine learning. These features are embedded in a joint vector space, so typical patterns indicative of malware can be automatically identified in a lightweight app installed in the smart device. Shabtai et al. [19] presented a system for mobile malware detection that takes into account the analysis of deviations in application networks behavior (app's network traffic patterns). This approach tackles the challenge of the detection of an emerging type of malware with self-updating capabilities based on runtime malware detector (anomaly-detection system) and it is also standalone monitoring application for smart devices.

Considering that [17] and Arp et al. [18] utilize static methods, they suffer from the inherent limitations of static code analysis (e.g., obfuscation techniques, junk code to evade successful decompilation). In the first case, their malware detection is based upon the structural similarity of static call graphs that are processed over approximations, while our method relies upon real functions calls that can be filtered later on. In the case of Debrin, transformation attacks that are nondetectable by static analysis, as, for example, based on reflection and bytecode encryption, can hinder an accurate detection.

Although in [19] we have a detection system that continuously monitors app executions. There is a concern about efficiency of the detection algorithm used by this system. Unfortunately, in this case, they could not evaluate the Features Extractor and the aggregation processes' impact on the mobile phone resources, due to the fact that an extended list of features was taken into account. To further enhance the system's performance, it is necessary to retain only the most effective features in such a way that the runtime malware detector system yields relatively low overhead on the mobile phone resources.

Our proposed infrastructure is related to the approaches mentioned above and employs similar features for identifying malicious applications, such as permissions, network addresses, API calls, and function call graphs. However, it differs in three central aspects from previous work. First, we have a runtime malware detection (dynamic analysis) but abstain from crafting detection in protected environment as the dynamic inspections done by *VetDroid*. While this system provides detailed information about the behavior of applications, they are technically too involved to be deployed on smartphones and detect malicious software directly. Second, our visual analysis system is based on accurate API call graphs, which enables us to inspect directly the app in an easy-to-follow manner in the cloud. Third, we are able to monitor not just the network traffic, but most of the restricted and suspicious API calls in Android. Our platform is more dynamic and simpler than other approaches mentioned above.

General overview of the state of security in mobile devices and approaches to deal with malware can be found in [20], and in the work by Suarez-Tangil et al. in [21], as well as in recent surveys by Faruki et al. in [7] and Sufatrio et al. [6]. Malware in smart devices still poses many challenges and, in different occasions, a tool for monitoring applications at a large scale might be required. Given the different versions of Android OS, and with a rising number of device firmwares, modifying each of the devices might become a nontrivial task. This is the scenario in which the proposed infrastructure in this paper best fits. The core contribution of this work is the development of a monitoring and instrumentation system that allows a visual analysis of the behavior of Android applications for any device on which an instrumented application can run. In particular, our work results in a set of dendrograms that visually render existing API calls invoked by Android malware application, by using dynamic inspection during a given time interval, and visually highlighting the suspicious ones. Consequently, we aim to fill the void of visual security tools which are easy to follow for Android environments in the technical literature.

4. Platform Architecture

When Android applications are executed, they call a set of functions that are either defined by the developer of the application or are part of the Android API. Our approach is based on monitoring a desired subset of the functions (i.e., hooked functions) called by the application and then uploading information related to their usage to a remote server. The hooked function traces are then represented in a graph structure, and a set of rules are applied to color the graphs in order to visualize functions that match known malicious behavior.

For this, we use four components: the *Embedded client* and the *Sink* on the smartphone side, and the *Web Service* and the *Visualization component* on the remote server side.

A work flow depicting the main elements of the involved system is shown in Figure 2. In Stage 1, the application under study and a set of permissions are sent to the Web Service. Next, the main processing task of Stage 2, labeled as hooking process, is introduced. In this case, hooks or logging codes are inserted in the functions that require at least one of the permissions specified at the previous stage.

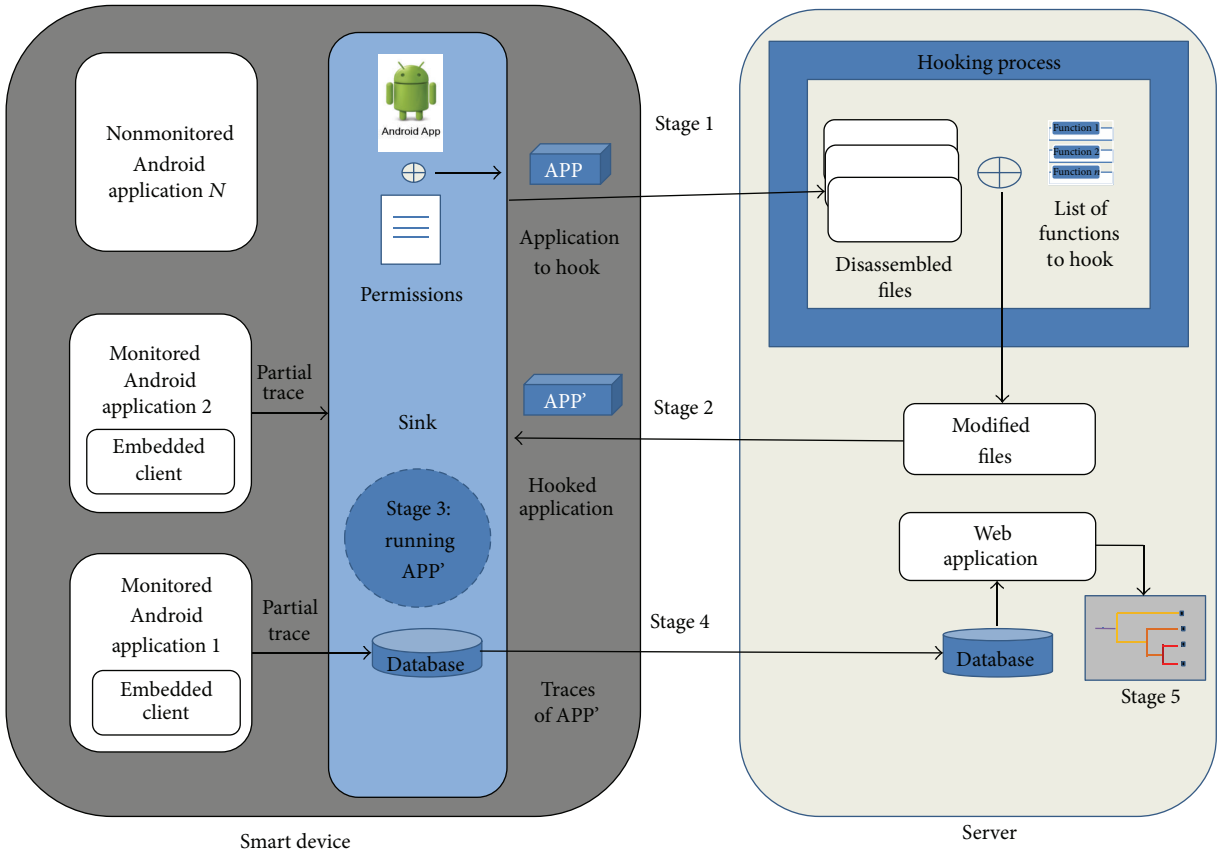


Figure 2: Schematics and logical stages of the system.

The new "augmented" application will be referred to as APP' from now on. Stages 3, 4, and 5 consist of running APP', saving the traces generated by APP' in the server's database, and showing the results as visualization graphs, respectively. The aforementioned infrastructure for platform-independent monitoring of Android applications is aimed to provide behavioral analysis without modifying the Android OS or root access to the smart device.

4.1. Embedded Client and Sink. The monitoring system consists of two elements: an embedded client that will be inserted into each application to be monitored and a Sink that will collect the hooked functions that have been called by the monitored applications. The embedded client simply consists of a communication module that uses the User Datagram Protocol (UDP) for forwarding the hooked functions to the Sink. Here, JavaScript Object Notation (JSON) is used when sending the data to the Sink, which allows sending dynamic data structures. In order to know the origin of a hooked function that has been received by the Sink, the corresponding monitored application adds its application hash, its package name, and its application name to the hooked function which we call a partial trace before sending it to the Sink.

The partial traces are built by the prologue functions (i.e., hook functions) that are placed just before their hooked functions and which modify the control flow of the monitored applications in order to build the partial traces corresponding with their hooked functions and passing the partial traces as parameter to the embedded client. Only the partial traces are built by the monitored application so that we add little extra overhead to the monitored application. The insertion of the embedded client and of the prologue functions in the Android application that is to be monitored is explained in Section 4.3.

The embedded client is written using the Smali syntax and is included on each of the monitored applications at the Web Service, at the same time that the functions hooks are inserted, before the application is packed back into an Android binary .apk file.

The Sink, on the other hand, is implemented as an Android application for portability both as a service and an activity whose service is started at the boot time. It is responsible for receiving the partial traces issued from all the monitored applications clients via a UDP socket, augmenting the partial traces to get a trace (i.e., adding a timestamp and the hash of the ID of the phone), storing them, and sending them over the network to the Web Service. As for the activity, it is responsible for managing the monitored applications via a UI, sending the applications to hook to the Web Service, and downloading the hooked applications from the Web Service. By hooked applications, we mean the applications in which

hooks have been inserted. Once an application has been hooked then we can monitor it.

Before storing the traces in a local database, the Sink first stores them in a circular buffer which can contain up to 500 traces. The traces are flushed to the local database when any of the following conditions are met: (i) when the buffer is half full, (ii) when the Sink service is shutting down, or (iii) upon an activated timeout expiring. This bulk flushing enables the Sink to store the traces more efficiently. Unfortunately, if the service is stopped by force, we lose the traces that are present in the circular buffer. Once the traces are persisted in the local database, the timeout is rescheduled. Every hour, the Sink application tries to send the traces that remain in the local database out to the Web Service. A trace is removed locally upon receiving an acknowledgment from the Web Service. An acknowledgment is issued when the Web Service has been able to record the trace in a SQL database with success. If the client cannot connect to the Web Service, it will try again at the next round.

When a user wants to monitor an application, a message with the package name as payload is sent to the Sink service which keeps track of all the applications to monitor in a list. When a user wants to stop monitoring a given application, a message is sent to the Sink service which removes it from its list of applications to monitor.

4.2. The Web Service. This server provides the following services to *Sink*: upload applications, download the modified applications, and send the traces. Now the key part of the whole system, where the logic of the method presented lies, is the tool that implements the application, a process known as "hooking." In the following, we explain it. The Web Service, implemented as a Servlet on a Tomcat web application server, is a RESTful Web Service which exposes services to clients (e.g., Android smartphone) via resources. The Web Service exposes three resources which are three code pages enabling the Sink to upload an application to hook, download a hooked application, and send traces. The hooking process is explained in more detail in Section 4.3.1.

The file upload service allows the Sink to send the target application to monitor and triggers the command to insert all the required hooks and the embedded client to the application. Also, it is in charge of storing the submitted Android binary .apk file on the server and receiving a list of permissions. This set of permissions will limit the amount of hooks to monitor, hooking only the API function calls linked to these permissions. Conversely, the file download service allows the Sink to download the previously sent application, which is now prepared to be monitored. A ticket system is utilized in order to keep tracking of the current application under monitoring. The trace upstream service allows the Sink to upload the traces stored on the device to the server database and remove the traces from the devices local SQLite database. Upon receiving traces, the Web Service records them in a SQL database and sends an acknowledgment back to the Sink. In case of failure in the server side or in the communication channel, the trace is kept locally in the SQLite database until the trace is stored in the server and an acknowledgment is received by the Sink. In both cases, it might occur that the trace has just been inserted in the SQL database and no answer is sent back. Then the Sink would send again the same trace and we would get a duplication of traces. However, the mechanism of primary key implemented in the SQL database prevents the duplication of traces. A primary key is composed of one or more data attributes whose combination of values must be unique for each data entry in the database. When two traces contain the same primary key, only one trace is inserted while the insertion of the other one throws an exception. When such an exception is thrown, the Web Service sends back an acknowledgment to the Sink so as to avoid the Sink resending the same trace (i.e., forcing the Sink to remove from its local database the trace that has already been received by the Web Service).

4.3. Instrumenting an Application. In this section, we first describe the process of inserting hooks into an Android application and then we show an example of a hook implementation. A tutorial on instrumentation of Android applications is presented by Arzt et al. in [22].

However, before proceeding with the insertion of instrumentation code to the decompiled APK below, we would like to clarify the effect of disassembling the uploaded applications, that is, the differences between the original code and code generated after instrumentation. Briefly, the disassembling of the uploaded application is performed by using the Smali/Baksmali tool which is assembler/disassembler, respectively, for the dex-format (https://source.android.com/devices/tech/dalvik/dex-format.html). This is the format used by Dalvik, one of the Android's JVM implementations. Thus, the disassembling is able to recover an assembler-like representation of the Java original code. This representation is not the original Java source code (Baksmali is a disassembler, not a decompiler after all). However, Baksmali creates both an exact replica of the original binary code behavior and high-level enough to be able to manipulate it in an easy way. This is why we can add additional instructions to instrument the original code for our purposes and then reassemble it back to a dex file that can be executed by Android's JVM. On the other hand, as discussed in [22], instrumentation of applications outperforms static analysis approaches, as instrumentation code runs as part of the target app, having full access to the runtime state. So, this explains the rationale behind introducing hooks in order to trace core sensitive or restricted API functions used at runtime of the apps. In other words, the Smali code reveals the main restricted APIs utilized by the apps under test, even in the presence of source code obfuscation. We can therefore resort to monitoring these restricted APIs and keep tracking of those Android suspicious programs' behavior.

4.3.1. Hooks Insertion. The hooking process is done in 6 steps: (i) receiving the application to hook from the smartphone, (ii) unpacking the application and disassembling its Dalvik byte code via the Android-apktool, (iii) modifying the application files, (iv) assembling Dalvik byte code and packing the hooked application via the Android-apktool, (v) signing the

```
(1) .class public Lcom/mainactivity/MainActivity;
(2) ...
(3) invoke-static/range {v2 ··· v6}, log_sendTextMessage(···)
(4) invoke-virtual/range {v1 ··· v6}, sendTextMessage(···)
(5) ...
```

Listing 1: Main activity class.

```
(1) .class public Lorg/test/MonitorLog;
(2) ...
(3) .method public static log_sendTextMessage(···)
(4) ...
(5) const-string v0, "packageName: com.testprivacy,..."
(6) invoke-static {v0}, sendLog(Ljava/lang/String;)
(7) return-void
(8) .end method
(9)
(10) .method public static sendLog(Ljava/lang/String;)
(11) .locals 3
(12) .parameter payload
(13) move-object v0, p0
(14) ...
(15) new-instance v1, Ljava/lang/Thread;
(16) new-instance v2, Lorg/test/EmbeddedClient;
(17) invoke-direct {v2, v0}, init(Ljava/lang/String;)
(18) invoke-direct {v1, v2}, init(Ljava/lang/Runnable;)
(19) invoke-virtual {v1}, start()
(20) return-void
(21) .end method
```

Listing 2: Monitor log class.

hooked application, and (vi) sending the hooked application upon request of the smartphone.

Step (iii) can be subdivided into several substeps:

(1) adding the Internet permission in the *AndroidManifest* to enable the embedded client inserted in the application to hook to communicate with the Sink via UDP sockets,

(2) parsing the code files and adding invocation instructions to the prologue functions before their corresponding hooked functions: when the monitored application is running, before calling the hooked function, its corresponding prologue function will be called and will build its corresponding partial trace. The list of desired functions to hook is provided by the administrator of the Web Service. For instance, if the administrator is interested in knowing the applications usage, it will hook the functions that are called by the application when starting and when closing,

(3) adding a class that defines the prologue functions: it is worth noting that there will be as many prologue functions as functions to hook. Each prologue function builds its partial trace. Since we do not log the arguments of the hooked functions, the partial traces that are issued by the same monitored application will only differ by the name of the hooked function. It is also worth noting that the prologue functions are generated automatically.

Since every Android application must be signed by a certificate for being installed on the Android platform, we use the same certificate to check if the hooked application comes from our Web Service. For this, the certificate used in the Web Service has been embedded in the Sink application. This prevents attackers from injecting malicious applications by using a man-in-the-middle attack between the smartphone and the Web Service.

4.3.2. Hook Example. Consider a case where the function *sendTextMessage*, used to send short messages (SMS) on the Android platform, is to be logged in a monitored application. This function is called in the main activity class of the application corresponding to the code Listing 1. As for the class shown in Listing 2, it defines the prologue functions and the function responsible for passing the partial traces, built

by the prologue functions, to the embedded client. For space reasons, we will not show the embedded client.

In the main activity class corresponding to the class shown in Listing 1, the function *sendTextMessage* is called at line (4) with its prologue function *log_sendTextMessage* which has been placed just before at line (3). Since the hooked function may modify common registers used for storing the parameters of the hooked function and for returning objects, we have preferred placing the prologue functions before their hooked functions. The register *v1* is the object of the class *SmSManager* needed to call the hooked function. As for the registers *v2* to *v6*, they are used for storing the parameters of the hooked function. Since our prologue functions are declared as static, we can call them without instantiating their class 2, and therefore we do not need to use the register *v1*.

An example of the monitor log class is shown in Listing 2. The name of the class is declared at line (1). At lines (3) and (10), two functions are defined, namely, *log_sendTextMessage* and *sendLog*. The former function, prologue function of the hooked function *sendTextMessage*, defines a constant string object containing the partial trace at line (5) and puts it into the register *v0*. Then the function *sendLog* is called at line (6) with the partial trace as parameter. The latter function saves the partial trace contained in the parameter *p0* into the register *v0* at line (13). At lines (15) and (16), two new instances are created, respectively: a new thread and new instance of the class *EmbeddedClient*. Their instances are initialized, respectively, at lines (17) and (18). Finally, the thread is started at line (19) and the partial trace is sent to the Sink. It is worth noting that, in these two examples, we have omitted some elements of the code which are replaced by dots to facilitate the reading of the code.

4.4. Visualization. The visualization of anomalous behavior is the last component of the proposed architecture. In order to perform a visual analysis of the applications' behavior in a simplified way, a D3.js (or just D3 for Data-Driven Documents (JavaScript library available at http://d3.org/)) graph was used. D3 is an interactive and a browser-based data visualizations library to build from simple bar charts to complex infographics. In this case, it stores and deploys graph oriented data on a tree-like structure named dendrograms using conventional database tables. Generally speaking, a graph visualization is a representation of a set of nodes and the relationships between them shown by links (*vertices* and *edges*, resp.).

This way, we are able to represent each of the analyzed application's behaviors with a simple yet illustrative representation. In general, the graphs are drawn according to the schema depicted in Figure 3. The first left-hand (root) node, "Application," contains the package name of the application, which is unique to each of the existing applications. The second middle node (parent), "Class," represents the name of the Android component that has called the API call. The third node, "Function" (the right-hand or child node), represents the names of functions and methods invoked by the application. It is worth noting that each application can include several classes and each class can call various functions or methods.

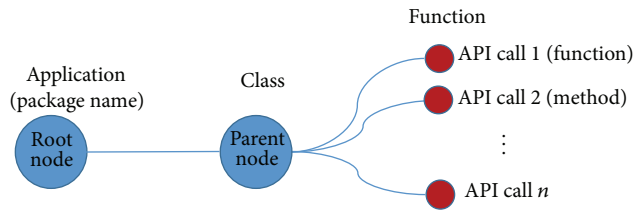


Figure 3: Schema used for the dendrograms.

In other words, function calls are located in the right-hand side of the dendrogram. For each node at this depth we are looking for known suspicious functions derived from a set of predefined rules as described below.

4.4.1. Rules "Generation". The rules aim to highlight restricted API calls, which allow access to sensitive data or resources of the smartphone and are frequently found in malware samples. These could be derived from the static analysis where the classes.dex file is converted to Smali format, as mentioned before, to get information considering functions and methods invoked by the application under test. On the other hand, it is well know that many types of malicious behaviors can be observed during runtime only. For this reason we utilize dynamic analysis; that is, Android applications are executed on the proposed infrastructure (see Figure 2) and interact with them. As a matter of fact, we are only interested in observing the Java based calls, which are mainly for runtime activities of the applications. This includes data accessed by the application, location of the user, data written to the files, phone calls, sending SMS/MMS, and data sent and received to or from the networks.

For the case that an application requires user interactions, we resort to do that manually so far. Alternatively, for this purpose one can use MonkeyRunner toolkit, which is available in Android SDK.

In [18, 23], authors list API functions calls that grant access to restricted data or sensible resources of the smartphone, which are very often seen in malicious code. We base our detection rules in those suspicious APIS calls. In particular, we use the following types of suspicious APIs:

(i) API calls for accessing sensitive data, for example, IMEI and USIMnumbeleakage, such as *getDeviceId()*, *getSimSerialNumber()*, *getImei()*, and *getSubscriberId()*,

(ii) API calls for communicating over the network, for example, *setWifiEnabled()* and *execHttpRequest()*,

(iii) API calls for sending and receiving SMS/MMS messages, such as *sendTextMessage()*, *SendBroadcast()*, and *sendDataMessage()*,

(iv) API calls for location leakage, such as *getLastKnownLocation()* and *getLatitude()*, *getLongitude()*, and *requestLocationUpdates()*,

(v) API function calls for execution of external or particular commands like *Runtime.exec()*, and *Ljava/lang/Runtime; -> exec()*,

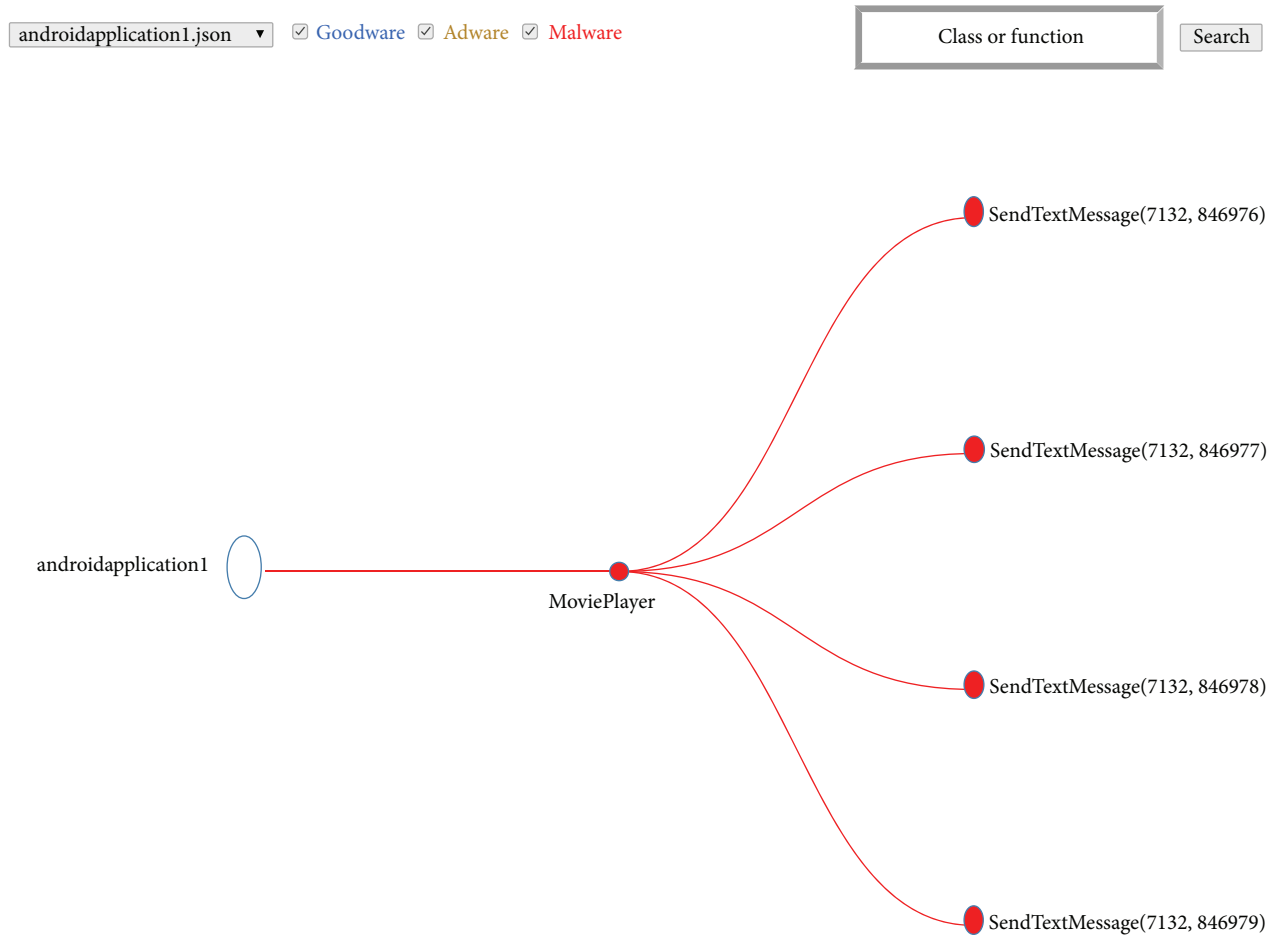


Figure 4: The simplified dendrogram of the malware *FlakePlayer* has been generated using the D3. Note that at the upper left corner of the figure there is a combobox to select the monitored malware (here, for simplicity, we use a shortened version of package name of the app, i.e., *androidapplication1*). Besides, lining up to the right of the combobox, there are three activated checkboxes, labeled as Goodware in blue, Adware in orange, and Malware in red. Also, at the upper right corner of the figure, there is a *search button* that allows us to look for classes or functions. The complete package name of the malware *FakePlayer* is *org.me.androidapplication1.MoviePlayer*.

(vi) API calls frequently used for obfuscation and loading of code, such as *DexClassLoader.Loadclass()* and *Cipher.getInstance()*.

Here the rule module uses the above-mentioned API calls to classify the functions and methods invoked on the runtime of the applications into three classes, that is, Benign, Adware, or Malware. So in this way, we can generate IF-THEN rules (cf. rules-based expert systems). Next we show example rules that describe suspicious behavior. Some of the rules generated by us are similar or resemble the ones in [24], namely,

(1) a rule that shows that the examined app is not allowed to get the location of the smart device user:

IF Not (*ACCESS_FINE_LOCATION*) AND *CALL_getLastKnownLocation* THEN Malware,

(2) another rule which might detect that the application is trying to access sensitive data of the smartphone without permission:

IF Not (*READ_PHONE_STATE*) AND *CALL_getImei* THEN Malware.

Our approach selects from the database those functions that have been executed that match the suspicious functions described in the rules. Package name and class name of such function are colored accordingly to the "semaphoric" labeling described in Section 6.1.

To illustrate the basic idea we choose a malware sample, known as *FakePlayer*, in order to draw its graph. Thus, by means of running the filtering and visualization operations we end up with the graph of the malware, shown in Figure 4.

The system allows adding new rules in order to select and color more families of suspicious functions.

5. Testbed and Experimentation

Before introducing the reader into the results of using the monitoring and visualization platform, we need to explain the testbed. We first describe the experiment setup; then we follow the steps of running the client-side Sink.

5.1. Experiment Set Up. All the experiments have been realized on a Samsung Nexus S with Android Ice Cream

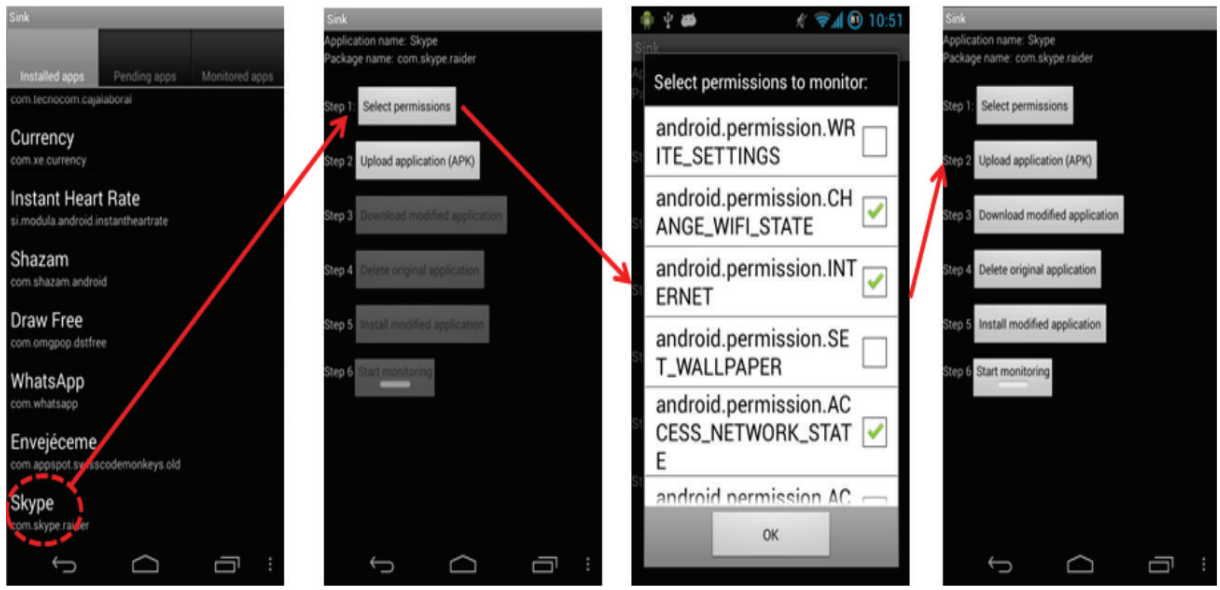


Figure 5: User interface of the Sink. (a) Choosing the application, (b) selecting the menu for permissions, (c) electing the permissions, and (d) steps of the monitoring process.

Sandwich (ICS). The Nexus S has a 1 GHz ARM Cortex A8 based CPU core with a PowerVR SGX 540 GPU, 512 MB of dedicated RAM, and 16 GB of NAND memory, partitioned as 1 GB internal storage and 15 GB USB storage.

We have explored different Android applications in order to evaluate the whole framework; some of these samples have been taken from the Android Malware Genome Project (The Android Malware Genome Project dataset is accessible at http://www.malgenomeproject.org/):

(i) *FakePlayer malware*,

(ii) *SMSReplicator malware*,

(iii) *iMatch malware*,

(iv) *DroidKungFu1 malware*,

(v) *DroidKungfu4 malware*,

(vi) *The spyware GoldDream in two flavors*,

(vii) *GGTracker malware*.

5.2. Client-Side Monitoring. The activities in Figure 5(a) display all the applications installed on the device that did not come preinstalled, from which the user selects a target application to monitor. Once an application is selected, the next step is to choose which permission or permissions the user wants to monitor. This can be observed in the third snapshot (white background) of Figure 5(c). Following the permissions clearance, the interface guides the user along several activities starting with the uploading of the selected application which is sent to the Web Service where the hooks are inserted. After this hooking process has finished, the modified application is downloaded from the Web Service. Afterwards, the original application is uninstalled and replaced by the modified application. Finally, a toggle allows starting and stoping monitoring of the application at any time by the user.

We focus on the functions of the Android API that require, at least, one permission. This allows the user to select from the Sink those permissions that are to be monitored at each application. This allows understanding how and when these applications use the restricted API functions. The PScout [12] tool was used to obtain the list of functions in the "API permission map." This way, the permission map obtained contains (Android 4.2 version API level 17) over thirty thousand unique function calls and around seventy-five different permissions. Besides, it is worth mentioning here that we refer to those associated with a sensitive API as well as sensitive data stored on device and privacy-sensitive built-in sensors (GPS, camera, etc.) as "restricted API functions." The first group is any function that might generate a "cost" for the user or the network. These APIs include [8], among others, Telephony, SMS/MMS, Network/Data, In-App Billing, and NFC (Near Field Communication) Access. Thus, by using the API map contained in the server's database, we are able to create a list of restricted ("suspicious") API functions.

The trace managing part is a service that runs in background with no interface and is in charge of collecting the traces sent from the individual embedded clients, located on each of the monitored applications. It adds a timestamp and the hash of the device ID and stores them on a common circular buffer. Finally, the traces are stored in bulk on a common local SQLite database and are periodically sent to the Web Service and deleted from the local database.

In summary, the required steps to successfully run an Android modified instrumented application are listed in Figure 5(d) and comprise the following.

Step 1 (select permissions). Set up and run the platform. Choose an application APP to be monitored on the device. Elect the permission list.

Step 2 (upload the application (APK)). Then, when this command is launched to upload the applications to the Web Service, the hooking process is triggered.

Step 3 (download modified application). This starts the downloading of the hooked application.

Step 4 (delete original application). This command starts the uninstallation process of the original application.

Step 5 (install modified application). This command starts the installation process of the modified application using Android's default application installation window.

Step 6 (start monitoring). Finally, a toggle is enabled and can be activated or disabled to start or stop monitoring that application as chosen by the user.

6. Results

To evaluate our framework, in this section we show the visualization results for several different applications to both benign and malicious. Then we proceed to evaluate the Sink application in terms of CPU utilization and ratio of partial traces received. Finally, we estimate the CPU utilization of a monitored application and its responsiveness.

6.1. Visual Analysis of the Traces. As mentioned before, a set of predefined rules allows us to identify the suspicious API functions and depending on its parameters (e.g., application attempts to send SMS to a short code that uses premium services) we assign colors to them. This enables us to quickly identify the functions and associate them with related items. On top of that, by applying the color classification of each node of the graph associated with a function in accordance with the color code (gray, orange, and red) explained below, it allows a "visual map" to be partially constructed. Furthermore, this graph is suitable to guide the analyst during the examination of a sample classified as dangerous because, for example, the red shading of nodes indicates malicious structures identified by the monitoring infrastructure.

In particular, to give a flavor to this analysis, the dendrogram of *FakePlayer* in Figure 4 provides the user with an indication of the security status of the malware. Different colors indicate the level of alarm associated with the currently analyzed application:

(i) Gray indicates that no malicious activity has been detected, as of yet.

(ii) Orange indicates that no malicious behavior has been detected in its graph, although some Adware may be presented.

(iii) Red indicates in its graph that a particular application has been diagnosed as anomalous, meaning that it contained one or more "dangerous functions" described in our blacklist. Moreover, it could imply the presence of suspicious API calls such as *sendTextMessage* with forbidden parameters, or the case of using restricted API calls for which the required permissions have not been requested (root exploit).

So, it is possible to conduct a visual analysis of the permissions and function calls invoked per application, where using some kind of "semaphoric labeling" allows us to identify easily the benign (in gray and orange colors) applications. For instance, in Figure 4 there is a presence of malware, and the nodes are painted in red.

The dendrogram shown for *FakePlayer* confirms its sneaky functionality by forwarding all the SMS sent to the device to the previously set phone number remaining unnoticed. For the sake of simplicity, we reduce the API function call *sendTextMessage*(phoneNo, null, SMS Content, null, null) to *sendTextMessage*(phoneNo, SMS Content). It uses the API functions to send four (see Figure 4) premium SMS messages with digit codes on it in a matter of milliseconds. Of course, sending a SMS message does not have to be malicious per se. However, for example, if this API utilizes numbers less that 9 digits in length, beginning with a "7" combined with SMS messages, this is considered a costly premium-rate service and a malware that sends SMS messages without the user's consent. The malware evaluated sends SMS messages that contain the following strings: 846976, 846977, 846978, and 846979. The message may be sent to a premium SMS short code number "7132," which may charge the user without his/her knowledge. This implies financial charges. Usually, when this malware is installed, malicious Broadcast Receiver is enrolled directly to broadcast messages from malicious server to the malware, so that user cannot understand whether specific messages are delivered or not. This is because the priority of malicious Broadcast Receiver is higher than SMS Broadcast Receiver. Once the malware is started, sending the function call *sendTextMessage* of SMS Manager API on the service layer, a message with premium number is sent which is shown in Figure 4.

6.2. Interactive Dendrograms. In general, it is needed to conduct the visual analysis from different perspectives. To do that we have developed an interactive graph visualization. So, we have four options or features in the D3 visualization of the application to monitor, namely, (a) selection of full features of the application (Goodware checkbox, Adware checkbox, and Malware checkbox), (b) the Goodware checkbox indicating that the app is assumed to be Goodware, (c) the Adware checkbox of the application, and (d) the Malware checkbox to look for malicious code. The analyst can choose to observe a particular Java class or function by typing the name of it inside the search box and clicking on the related search button.

Figures 6 and 7 illustrate a big picture of the whole behavioral performance of the malware *DroidKungFu1* whose package name is *com.nineiworks.wordsXGN*, and the malicious function calls are invoked. For the sake of simplicity, we shorten the package name of *DroidKungFu1* to *wordsXGN* in the dendrogram. As a matter of fact, we apply a similar

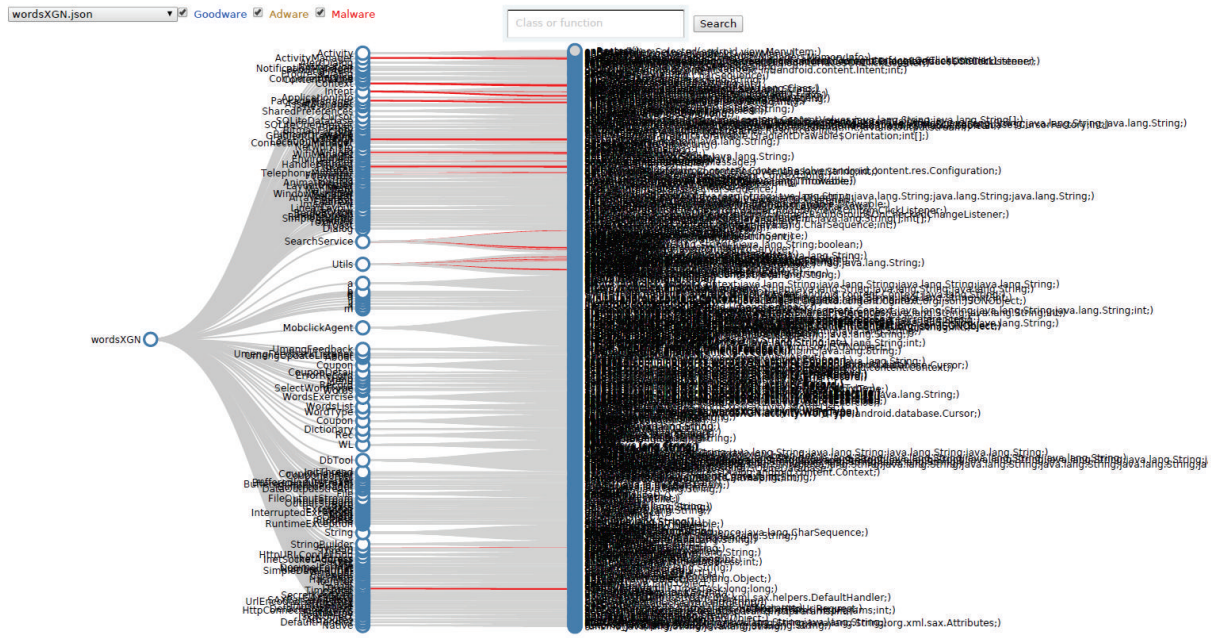


Figure 6: Visualization of the *DroidKungFu1* malware with full features chosen (i.e., all the checkboxes are activated).

Table 1: Malware family, detection rules, and suspicious functions.

Malware family	Detection rules	Suspicious functions
FakePlayer	IF (SEND_SMS) && (CALL_sendTextMessage() with preset numbers) THEN Malware	sendTextMessage(7132, null, 846976, null, null)
SMSReplicator	IF (SEND_SMS) && (CALL_sendTextMessage() with preset numbers) THEN Malware	sendTextMessage(1245, null, {From: 123456789 Hi how are you}, null, null)
iMatch	IF Not (ACCESS_FINE_LOCATION) && IF (SEND_SMS) THEN Malware	requestLocationUpdates(); sendTextMessage()
DroidKungFu1	[IF (INTERNET) && IF Not (ACCESS_FINE_LOCATION)] ‖ [IF (READ_PHONE_STATE) && IF (INTERNET)] THEN Malware	getLatitude(); getLongitude(); getDeviceid(); getLIne1Number(); getImei()
DroidKungFu4	IF (INTERNET) && IF (READ_PHONE_STATE) THEN Malware	getDeviceid(); getLIne1Number(); getSimSerial(); getImei();
GoldDream (Purman)	[IF (READ_PHONE_STATE) && IF Not (SEND_SMS)] ‖ [IF Not (READ_PHONE_STATE) && IF (INTERNET)] THEN Malware	getDeviceId(); getLIne1Number(); getSimSerial(); sendTextMessage(); getImei()
GoldDream (Dizz)	[IF (READ_PHONE_STATE) && IF Not (SEND_SMS)] ‖ [IF Not (ACCESS_FINE_LOCATION) && IF (INTERNET)] THEN Malware	getDeviceId(); getLIne1Number(); getSimSerial(); sendTextMessage(); requestLocationUpdates(); getImei()
GGTracker	[IF (READ_PHONE_STATE) && Not (SEND_SMS)] ‖ [IF Not (ACCESS_FINE_LOCATION) && IF (INTERNET)] THEN Malware	getDeviceId(); getLIne1Number(); getSimSerial(); sendTextMessage(); requestLocationUpdates(); getImei()

labeling policy to the other dendrograms. Moreover, we have the dendrograms for the *DroidKungFu4* in Figure 8. In particular, in the graph of Figure 8(a), we conduct the visual inspection by using full features (i.e., all the checkboxes active simultaneously) looking for red lines (presence of malware, if that is the case). Furthermore, in the graph of Figure 8(b), now we can focus our visual examination in the malicious functions carried out by the application. The visual analysis of the *DroidKungFu1* and *DroidKungFu4* includes encrypted root exploits, Command & Control (C & C) servers which in the case of *DroidKungFu1* are in plain text in a Java class file, and shadow payload (embedded app). In Table 1, we have some of the suspicious function calls utilized by the malware which pop up from the dendrograms. Regarding the IF-THEN rules, the allowed clauses or statements in our infrastructure are permissions and API functions calls. The fundamental operators are Conditional-AND which is denoted by &&, Conditional-OR which is denoted by ||,

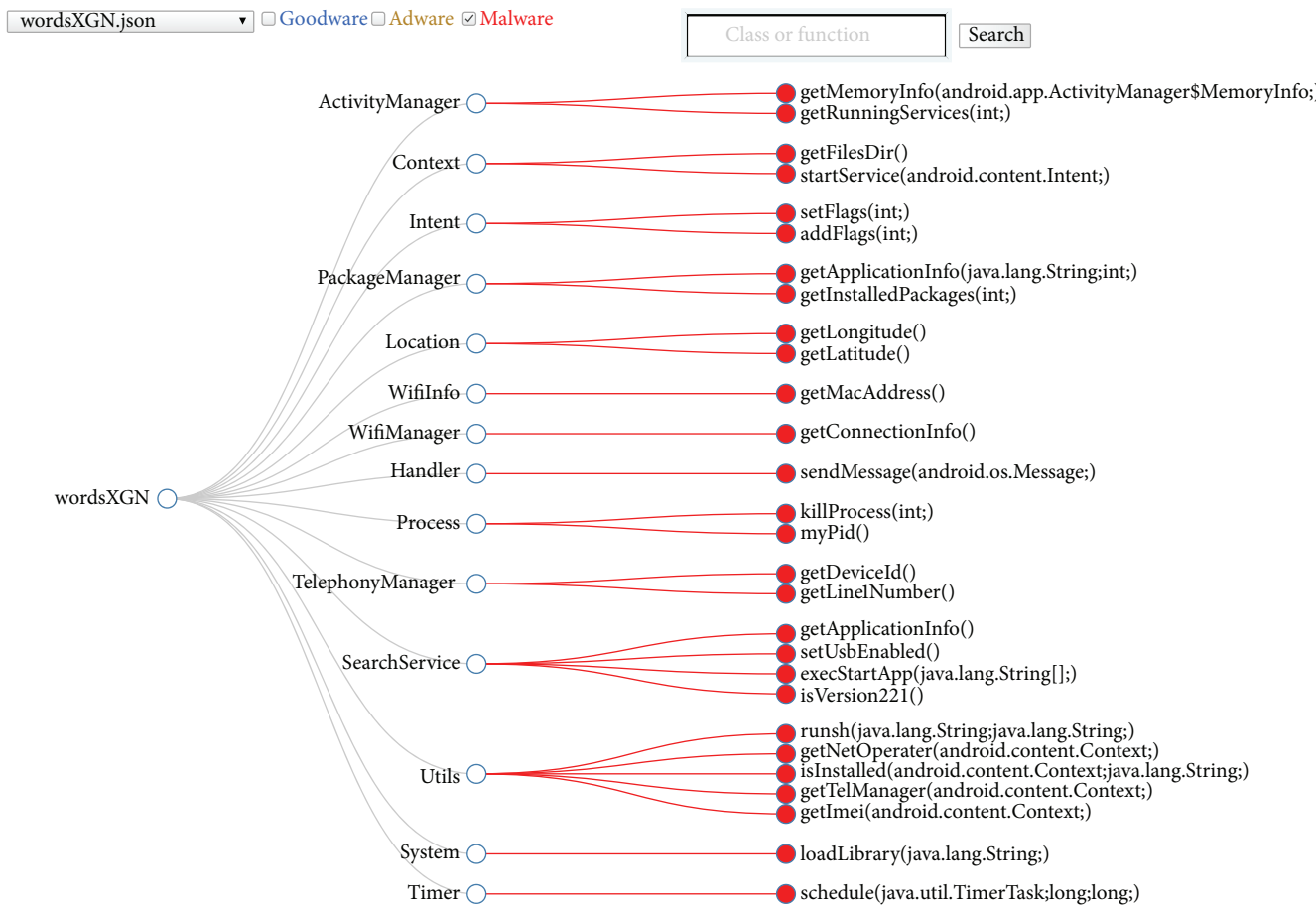


Figure 7: Visualization of the malicious API calls detected by our system for *DroidKungFu1*. Note the chosen options of the monitored malware in the dendrogram at the upper left side. First, we shorten version of the package name (*wordsXGN*) of the malware in the combobox. Next we have three checkboxes, namely, Goodware, Adware, and Malware. In this graph, only the red checkbox has been activated in order to conduct the visual analysis. The full package name of *DroidKungFu1* is *com.nineiworks.wordsXGN*.

and Not. For example, if the examined app does not have permission to send SMS messages in the *AndroidManifest* file and that app trys to send SMS messages with the location of the smartphone THEN that application may have malicious code. The rule generated for this case is shown below:

> IF Not (*SEND_SMS*) && (*ACCESS_FINE_LOCATION*) THEN Malware.

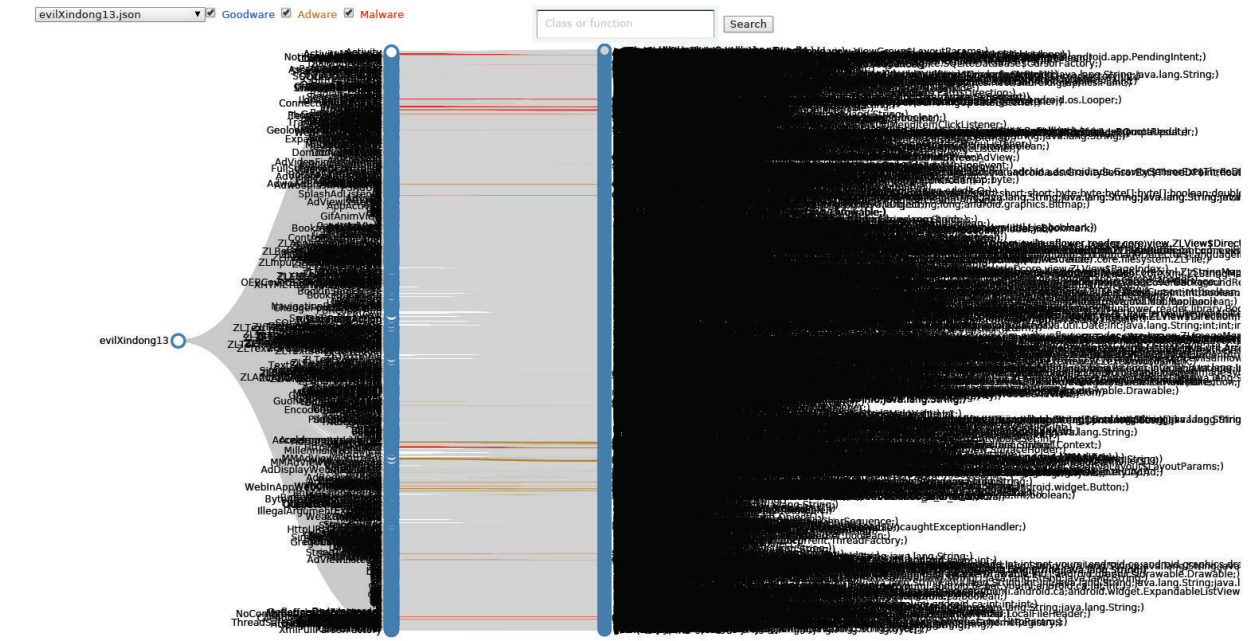
Here, malicious code and malware are interchangeable terms. The possible outcomes are Goodware or Malware. Nevertheless, the proposed infrastructure might be capable of evaluating a third option, Adware, in a few cases. In this paper we do not describe the IF-THEN rules for the third kind of outcome. In this work, we restrain the possible outcomes to the two mentioned options.

We have used 7 rules in our experimentation which are listed in Table 1 (note that rules 1 and 2 are the same). We have listed in Table 1 the most frequently used rules. They mainly cover cases of user information leakage.

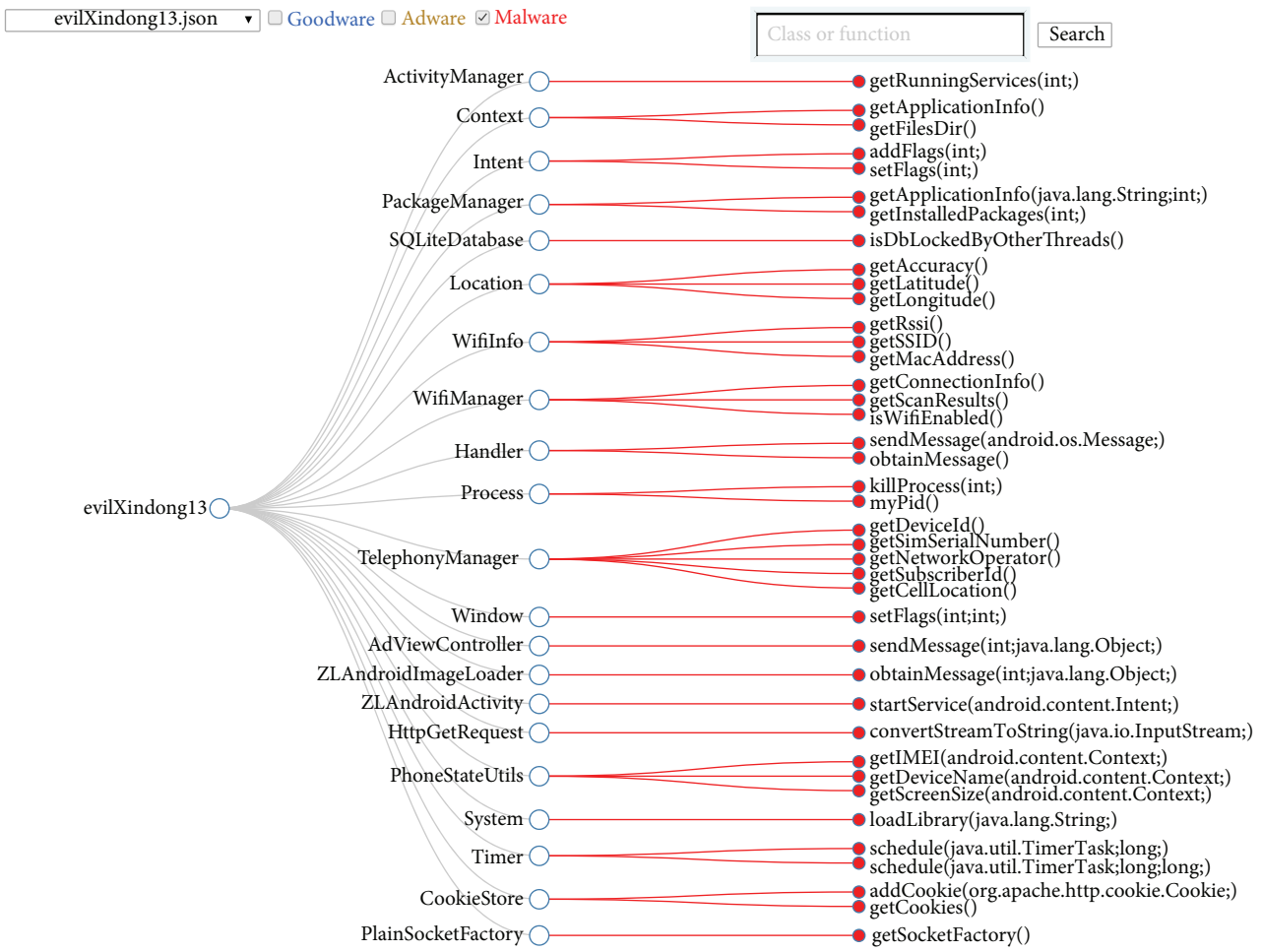
The most frequently used detection rules that we have utilized in our experimentation are listed in Table 1 (second column).

6.3. Client-Side CPU Use Analysis. We define the CPU utilization of a given application as the ratio between the time when the processor was in busy mode only for this given application at both the user and kernel levels and the time when the processor was either in busy or idle mode. The CPU times have been taken from the Linux kernel through the files "/proc/stat" and "/proc/pid/stat" where pid is the process id of the given application. We have chosen to sample the CPU utilization every second.

The CPU utilization of the Sink application has been measured in order to evaluate the cost of receiving the partial traces from the diverse monitored applications, processing them, and recording them in the SQLite database varying the time interval between two consecutive partial traces sent. We expect to see that the CPU utilization of the Sink increases as the time interval between two consecutive partial traces sent decreases. Indeed, since the Sink must process more partial traces, it needs more CPU resource. This is confirmed by the curve in Figure 9. The CPU utilization has a tendency towards 30% when the time interval between two consecutive partial traces received tends to 10 ms because the synthetic application takes almost 30% of the CPU for building and



(a) Dendrogram in full features (Goodware, Adware, and Malware) for *DroidKungFu4*



(b) Graph visualization of the detected malware in the case of *DroidKungFu4*

FIGURE 8: Dendrograms of the tested application. (a) Graph of the *DroidKungFu4* in full features, and (b) graph of the malicious functions invoked by *DroidKungFu4*. The full package name of DroidKungFu4 is com.evilsunflower.reader.evilXindong13.

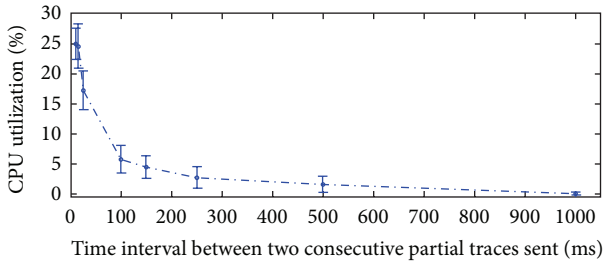


Figure 9: CPU utilization of the Sink.

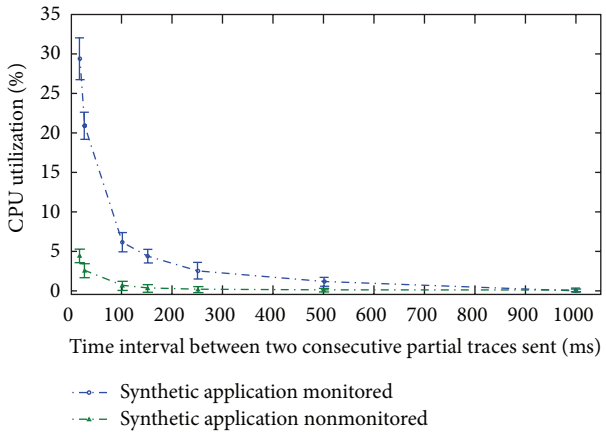


Figure 10: Difference of CPU utilization between an application monitored and nonmonitored.

sending partial traces, and the rest of applications utilize the rest of the CPU resource. When no monitored applications send partial traces to the Sink and the Sink is running in the background (i.e., its activity is not displayed on the screen), it consumes about 0% of the CPU.

The CPU utilization of a synthetic application has also been measured in order to evaluate the cost of building and sending the partial traces to the Sink while the time interval between two consecutive partial traces sent was varied. We expect to see a higher CPU utilization when the application is monitored. Indeed, since the synthetic application must build and send more partial traces, it needs more CPU resources. This is confirmed by Figure 10. We note that the increase of CPU utilization of the application can be up to 28% when it is monitored. The chart shows an increase in application CPU utilization to a level up to 38% which is justified when the monitoring is fine-grained at 10 ms. However, this high frequency is not likely to be needed in real applications.

6.4. Responsiveness. We define an application as responsive if its response time to an event is short enough. An event can be a button pushed by a user. In other words, the application is responsive if the user does not notice any latency while the application is running. In order to quantify the responsiveness and see the impact of the monitoring on the responsiveness of monitored applications, we have measured the time spent for executing the prologue function of the synthetic application. We have evaluated the responsiveness of the monitored application when the Sink was saturated by partial traces requests, that is, in its worst case. The measured response time was on average less than 1 ms. so, the user does not notice any differences when the application is monitored or not, even though the Sink application is saturated by partial traces. This is explained by the fact that UDP is connectionless and therefore sends the partial traces directly to the UDP socket of the Sink without waiting for any acknowledgments.

7. Limitations

So far we illustrated the possibilities of our visual analysis framework by analyzing 8 existing malicious applications. We successfully identified different types of malware accordingly to the malicious payload (e.g., privilege escalation, financial charges, and personal information stealing) of the app while using only dynamic inspection in order to obtain the outcomes. Even though the results are promising, they only represent a few of the massive malware attacking today's smart devices. Of course, the aim of this system is not to replace existing automated Android malware classification systems because the final decision is done by a security analyst.

Although, here, we propose a malware detector system based on runtime behavior, this does not have detection capabilities to monitor an application's execution in real time, so this platform cannot detect intrusions while running. It only enable detecting past attacks.

Also, one can figure out that malware authors could try avoiding detection, since they can gain knowledge whether their app has been tampered with or no. As a result, the actual attack might not be deployed, which may be considered a preventive technique. Moreover, it is possible for a malicious application to evade detection by dynamically loading and executing Dalvik bytecode at runtime.

One of the drawbacks of this work could be the manual interactions with the monitored application during runtime (over some time interval). Also, the classification needs a more general procedure to get the rule-based expert system. The natural next step is to automate these parts of the process. For example, in the literature there are several approaches that can be implemented in order to automatically generate more IF-THEN rules [15] or to resort to the MonkeyRunner kit available in Android SDK to simulate the user interactions. Of course, the outcomes of the 8-sample malware presented here are limited to longest time interval used in the study, which was 10 minutes. Extending this "playing" time with the app using tools for the automation of user's interactions could provide a more realistic graph and better pinpoint the attacks of the mobile malware.

Another limitation of this work is that it can only intercept Java level calls and not low level functions that can be stored as libraries in the applications. Thus, a malicious app can invoke native code through Java Native Interface (JNI), to deploy attacks to the Android ecosystem. Since our approach builds on monitoring devices that are not rooted, this approach is out of the scope of our research.

It is worth mentioning that our API hooking process does not consider the Intents. The current version of the

infrastructure presented in this paper is not capable of monitoring the Intents sent by the application, as sending Intents does not require any kind of permission. Not being able to monitor Intents means that the infrastructure is not able to track if the monitored application starts another app for a short period of time to perform a given task, for instance, opening a web browser to display the end-user license agreement (EULA). Also, adding this feature would allow knowing how the target application communicates with the rest of the third party and system applications installed on the device.

Ultimately, this framework could be useful for final users interested in what apps are doing in their devices.

8. Conclusions

We provide a monitoring architecture aiming at identifying harmful Android applications without modifying the Android firmware. It provides a visualization graph named dendrograms where function calls corresponding to predefined malware behaviors are highlighted. Composed of four components, namely, the embedded client, the Sink, the Web Service, and the visualization, any Android application can be monitored without rooting the phone or changing its firmware.

The developed infrastructure is capable of monitoring simultaneously several applications on various devices and collecting all the traces in the same place. The tests performed in this work show that applications can be prepared to be monitored in a matter of minutes and that the modified applications behave as they were originally intended to, with minimal interference with the permissions used for. Furthermore, we have shown that the infrastructure can be used to detect malicious behaviors by applications, such as the monitored *FakePlayer*, *DroidKungFu1* and *DroidKungFu4*, and the *SMSReplicator* and many others taken from the dataset of the Android Malware Genome Project.

Evaluations of the Sink have revealed that our monitoring system is quite reactive, does not lose any partial traces, and has a very small impact on the performance of the monitored applications.

A major benefit of the approach is that the system is designed as platform-independent so that smart devices with different versions of Android OS can use it. Further improvements on the visualization quality and the user interface are possible, but the proof of concept implementation is demonstrated to be promising. For future work, we plan to extend the current work in order to develop a real-time malware detection infrastructure based on network traffic and on a large number of apps.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This research has been partially funded by Basque Governments Elkartek Program under Grant no. KK-2015/0000080. Also, the last author would like to acknowledge the support by the RICS Centre for Resilient Information and Control Systems (http://www.rics.se/) financed by the Swedish civil contingencies agency.

References

[1] Gartner, "Gartner says worldwide traditional PC, tablet, ultramobile and mobile phone shipments on pace to grow 7.6 percent in 2014," October 2015, http://www.gartner.com/newsroom/id/2645115.

[2] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and J. Blasco, "Dendroid: a text mining approach to analyzing and classifying code structures in Android malware families," *Expert Systems with Applications*, vol. 41, no. 4, pp. 1104–1117, 2014.

[3] Y. Zhang, M. Yang, B. Xu et al., "Vetting undesirable behaviors in android apps with permission use analysis," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS '13)*, pp. 611–622, ACM, Berlin, Germany, November 2013.

[4] Microsoft, "Web services," 2014, http://msdn.microsoft.com/en-us/library/ms950421.aspx.

[5] R. T. Fielding, *Architectural styles and the design of network-based software architectures [Ph.D. thesis]*, University of California, Irvine, Calif, USA, 2000.

[6] Sufatrio, D. J. J. Tan, T.-W. Chua, and V. L. L. Thing, "Securing android: a survey, taxonomy, and challenges," *ACM Computing Surveys*, vol. 47, no. 4, article 58, 2015.

[7] P. Faruki, A. Bharmal, V. Laxmi et al., "Android security: a survey of issues, malware penetration, and defenses," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 2, pp. 998–1022, 2015.

[8] Android.com, "Android system permissions," 2014, http://developer.android.com/guide/topics/security/permissions.html.

[9] R. Winsniewski, "Android-apktool: a tool for reverse engineering android apk files," 2012.

[10] M. Karami, M. Elsabagh, P. Najafiborazjani, and A. Stavrou, "Behavioral analysis of android applications using automated instrumentation," in *Proceedings of the 7th International Conference on Software Security and Reliability (SERE-C '13)*, pp. 182–187, Gaithersburg, Md, USA, June 2013.

[11] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, and A.-R. Sadeghi, "Xmandroid: a new android evolution to mitigate privilege escalation attacks," Tech. Rep. TR-2011-04, Technische Universität Darmstadt, Darmstadt, Germany, 2011.

[12] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "PScout: analyzing the Android permission specification," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS '12)*, pp. 217–228, ACM, Raleigh, NC, USA, October 2012.

[13] J. Jeon, K. K. Micinski, J. A. Vaughan et al., "Dr. android and Mr. hide: fine-grained permissions in android applications," in *Proceedings of the 2nd ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM '12)*, pp. 3–14, ACM, Raleigh, NC, USA, October 2012.

[14] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behaviorbased 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.

[15] K. Patel and B. Buddhadev, "Predictive rule discovery for network intrusion detection," in *Intelligent Distributed Computing*,

vol. 321 of *Advances in Intelligent Systems and Computing*, pp. 287–298, Springer, Basel, Switzerland, 2015.

[16] X. Jiang and Y. Zhou, *Android Malware*, Springer, New York, NY, USA, 2013.

[17] H. Gascon, F. Yamaguchi, D. Arp, and K. Rieck, "Structural detection of android malware using embedded call graphs," in *Proceedings of the ACM Workshop on Artificial Intelligence and Security*, pp. 45–54, ACM, 2013.

[18] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: effective and explainable detection of android malware in your pocket," in *Proceedings of the Annual Symposium on Network and Distributed System Security (NDSS '14)*, San Diego, Calif, USA, February 2014.

[19] A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, and Y. Elovici, "Mobile malware detection through analysis of deviations in application network behavior," *Computers & Security*, vol. 43, pp. 1–18, 2014.

[20] M. La Polla, F. Martinelli, and D. Sgandurra, "A survey on security for mobile devices," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 1, pp. 446–471, 2013.

[21] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and A. Ribagorda, "Evolution, detection and analysis of malware for smart devices," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 961–987, 2014.

[22] S. Arzt, S. Rasthofer, and E. Bodden, "Instrumenting android and java applications as easy as abc," in *Runtime Verification*, pp. 364–381, Springer, Berlin, Germany, 2013.

[23] S.-H. Seo, A. Gupta, A. M. Sallam, E. Bertino, and K. Yim, "Detecting mobile malware threats to homeland security through static analysis," *Journal of Network and Computer Applications*, vol. 38, no. 1, pp. 43–53, 2014.

[24] K. Patel and B. Buddadev, "Detection and mitigation of android malware through hybrid approach," in *Security in Computing and Communications*, vol. 536 of *Communications in Computer and Information Science*, pp. 455–463, Springer, Basel, Switzerland, 2015.

Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2016, Article ID 7538108, 8 pages
http://dx.doi.org/10.1155/2016/7538108

Research Article

Performance Analysis of a DEKF for Available Bandwidth Measurement

Diego Santoro and Michele Vadursi

Department of Engineering, University of Naples "Parthenope", Centro Direzionale Isola C4, 80143 Naples, Italy

Correspondence should be addressed to Michele Vadursi; vadursi@uniparthenope.it

Received 4 December 2015; Revised 29 January 2016; Accepted 3 February 2016

Academic Editor: Muhammad Taher Abuelma'atti

The paper presents a characterisation analysis of a measurement algorithm based on a Discrete-time Extended Kalman Filter (DEKF), which has recently been proposed for the estimation and tracking of end-to-end available bandwidth. The analysis is carried out by means of simulations for different rates of variations of the available bandwidth and permits assessing the performance of the measurement algorithm for different values of the filter parameters, that is, the covariance matrixes of the measurement and process noise.

1. Introduction

The available bandwidth (AB) of a path, also known as unused bandwidth, is a key network metric [1]. Its estimate is used for route selection, quality of service (QoS) verification, and traffic engineering. More recently, AB estimation has increased its importance with respect to new demanding scenarios like cloud computing, voice-over-IP (VoIP), and multimedia networks, where support tools for traffic shaping and traffic policing as well as developing network management strategies are crucial [2–7]. Moreover, available bandwidth can directly affect the performance of distributed systems and clocks [8, 9].

AB of a link i is defined as the unused or spare capacity of i during the time interval T. It depends on underlying transmission technology and propagation medium as well as the traffic load that occurs in the considered time interval [1, 10]. Traffic load is a time-varying metric and is related to the network utilization. In detail, assuming that at a specific instant t the channel is busy or idle, the instantaneous network utilization is referred to as $u(t)$ and takes value 1 or 0. The average utilization of a link i in the period T is defined as follows:

$$u_i(t) = \frac{1}{T}\int_{t-T}^{t} u(x)\,dx. \tag{1}$$

The AB at a given instant t of the link i can be thus expressed as follows:

$$A_i = (1 - u_i)\,C_i, \tag{2}$$

where C_i is the capacity of i [1, 10]. Finally, AB of an end-to-end path P is given by

$$A = \min_{k=1,\ldots,N} A_i, \tag{3}$$

where N is the number of hops along P and A_i represents AB of the ith link along P. In detail, the link of P with the minimum AB is referred to as tight link, while the link with the minimum capacity is referred to as narrow link. However, the term bottleneck in general refers to the link of a path that is characterised by the minimum AB and capacity.

The literature presents several tools to estimate AB of an end-to-end path, which are classified according to the underlying models that can be classified in PGM and PRM-based models. The PGM-based model is generally preferred compared to PRM model for its less intrusiveness and, generally, ease of implementation, although, the latter is more accurate. However, the literature presents several pieces of work based on the PGM model that loosen some of the underlying hypothesis with the ultimate purpose of evaluating its performance in more demanding scenarios.

The paper presents an analysis of the performance of a DEKF as a support tool for tracking abrupt AB changes during the measurement time presented in [11]. Abrupt changes during the measurement time de facto represent a loosening of the underlying PGM-based model's hypothesis of low variation of the AB during the measurement time. The presented characterisation study is performed in terms of covariance matrix of the process noise and measurement noise for different speed of the AB variations, in order to highlight limits, performance, and applicability of the method. The analysis provides useful results for the estimation of the most appropriate values for both aforementioned covariance matrixes.

The paper is organised as follows. Section 2 describes the PRM and PGM models and introduces the fundamentals of the DEKF under analysis. Section 3 gives details on the simulations that have been carried out. Section 4 presents and discusses the results of the simulations. Finally, Section 5 gives the conclusions.

2. Related Work

2.1. Available Bandwidth Estimation. Several measurement methodologies and tools have been developed and are available in the literature and on the market [12] for the estimation of the end-to-end AB of a network path. They can be classified in PGM-based (Probe Gap Model) methods and PRM-based (Probe Rate Model) methods. PGM-based tools implement the so-called iterative probing approach, whereas PRM-based tools are based on direct probing.

An implementation of PRM exploits a self-congestion of end-to-end path. In detail, the system sends a train of packets whose probing packet rate is decreased as it reaches AB of the path, and then a queue starts building up at the bottleneck link. The bottleneck link is in general referred to as the link of the path with the minimum AB and capacity [1]. The queue increases the probing packet rate at the receiver node which gets lower than the probing packet rate at the sender node. Tuning the probing packet rate of the train of packets at the sender node with the probing packet rate at the receiver node makes it possible to calculate the AB of the path, which is by definition the AB of the bottleneck link. Examples of applications based on the PRM are Pathload [13], Pathchirp [14], and PTR/IGI [15]. An implementation based on the PGM sends a pair of packets (but it can also be a train of packets [1]) separated by an initial time interval Δ_{in} at the sender node; then the receiver node receives a pair of packets separated by an increased time interval Δ_{out}. Like the PRM, also the PGM requires that the routers along the end-to-end path implement a FIFO queue and the cross traffic to be constant (it is also called path persistent cross traffic) and that it slowly changes during the measurement time. The time interval Δ_{out} at the receiver node represents the time interval of the bottleneck link to transmit the pair of packets and the cross traffic that occurs between them. AB of the bottleneck is calculated according the following formula:

$$A = \left(1 - \frac{\Delta_{\text{out}} - \Delta_{\text{in}}}{\Delta_{\text{out}}}\right) C, \tag{4}$$

where C is the capacity of the bottleneck link. It is worth noting that the PGM requires that there is a single bottleneck along the end-to-end path. In other words, this means that the tight link (which is the link with the minimum AB) matches the narrow link (which is the link with the minimum capacity) [1]. The probing packets rate at the first hop is set equal to the capacity of the bottleneck. Examples of measurement tools based on PGM are Delphi [16] and Spruce [10].

PGM-based measurement methodologies are simple to implement and generally less intrusive than PRM-based ones. However, due to their underlying assumptions, such methodologies can suffer from reduced accuracy.

Both PGM and PRM models take the following hypothesis: (i) there are no L2 (Layer-2) devices, such as switches, of the OSI (Open System Interconnected) model along the end-to-end path, (ii) the routers implement FIFO (First-In-First-Out) queues, and (iii) average rate of the path persistent cross traffic changes slowly and it is constant during the measurement time. Moreover, the PGM model requires that there is one bottleneck along the path, which means that the narrow link matches with the tight link [10].

The work presented in [17] showed that PGM-based tools can underestimate the AB in case the condition of having a constant traffic along the whole path is not met. The paper demonstrates that the measurement model is strongly affected by the position of the bottleneck link in the path and the model uncertainty can lead to nonnegligible measurement error. In particular, in case of one-hop persistent traffic, which means that the cross traffic rate changes at each hop, the measurement equation turns into a different expression. The model modification depends not only on the position of the bottleneck in the path but also on the initial rate of probing packets. In case the bottleneck is not the first hop, the probing packet rate could happen to be increased at the hops preceding the bottleneck, thus resulting in a variation of the one-hop probing packet.

Recently, a measurement method based on DEKF to track the AB in presence of abrupt variations due to variations of path persistent cross traffic over time was presented [11]. The method is based on a PGM and specifically on the model presented in [10]. Although a few experiments are reported in [11], no performance analysis of the filter with respect to its parameters and/or the speed of AB variations is given.

2.2. Fundamentals of Kalman Filtering. The Kalman filter solves the problem of estimating the state $x \in \mathfrak{R}^n$ of a discrete-time controlled process that is governed by the linear stochastic difference equation [18, 19]:

$$\begin{aligned} x_k &= Ax_{k-1} + Bu_{k-1} + w_{k-1}, \\ z_k &= Hx_k + v_k, \end{aligned} \tag{5}$$

where $A \in \mathfrak{R}^{n\times n}$ is the state transition matrix of the process, $B \in \mathfrak{R}^{n\times l}$ ties the input vector with the state vector, $u_{k-1} \in \mathfrak{R}^l$ represents the input vector at the step $k-1$, $z_k \in \mathfrak{R}^m$ represents the measurements carried out on the system at the step k, and $H \in \mathfrak{R}^{m\times n}$ is the measurement matrix.

$w_{k-1} \in \Re^n$ and $v_k \in \Re^m$ represent, respectively, the process and the measurement noise, which are assumed to be statistically independent of each other and have normal probability distribution: $p(w) = \aleph(0, Q)$ and $p(v) = \aleph(0, R)$, where $Q \in \Re^n$ and $R \in \Re^m$ are, respectively, the covariance matrix of the process and measurement noise. In case, the system is described by nonlinear stochastic difference equations as the following:

$$\begin{aligned} x_k &= f\left(x_{k-1}, u_{k-1}, w_{k-1}, k-1\right), \\ z_k &= h\left(x_k, k\right) + v_k. \end{aligned} \tag{6}$$

Then, the above system can be described as a DEKF, where the functions f and h are approximated to their first-order Taylor's series expansions. The state vector estimation at the step k is given by first calculating the following predict equations:

$$\begin{aligned} \widehat{x}_k &= f\left(\widehat{x}_{k-1}, u_{k-1}, 0\right), \\ P_k^- &= A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T, \end{aligned} \tag{7}$$

where

$$\begin{aligned} A_k &= \frac{\partial f_i}{\partial x_j}\left(\widehat{x}_{k-1}, u_{k-1}, 0\right), \\ W_k &= \frac{\partial f_i}{\partial w_j}\left(\widehat{x}_{k-1}, u_{k-1}, 0\right), \end{aligned} \tag{8}$$

A_k being the Jacobian matrix of partial derivatives of f with respect to x and represents the state transition matrix and W_k being the Jacobian matrix of partial derivatives of f with respect to v. Q_k is the original covariance matrix of the process noise at the step k and it is assumed to be a constant and therefore the subscript is dropped in the rest of the paper.

The state vector estimation is, then, given by the following measurements update equations:

$$\begin{aligned} K_k &= P_k^- H_k^T \left(H_k P_k^- H_k^T + V_k R_k V_k^T\right)^{-1}, \\ \widehat{x}_k &= \widehat{x}_k^- + K_k\left(z_k - h\left(\widehat{x}_k^-, 0\right)\right), \\ P_k &= \left(I - K_k H_k\right) P_k^-, \end{aligned} \tag{9}$$

where

$$\begin{aligned} H_k &= \frac{\partial h_i}{\partial x_j}\left(\widehat{x}_k, u_{k-1}, 0\right), \\ V_k &= \frac{\partial f_i}{\partial v_j}\left(\widehat{x}_k, u_{k-1}, 0\right) \end{aligned} \tag{10}$$

and $P_k \in \Re^{n \times n}$ represents the error covariance matrix at the step k ($P_k^- \in \Re^{n \times n}$ is the a priori error covariance matrix).

The DEKF presented in [11] is intended to track AB variations during the measurement time in order to provide more accurate AB measurements, which according to (5) can be formalised as follows: (5) turn into the following:

$$\begin{aligned} x_k &= x_{k-1} + w_{k-1}, \\ z_k &= -\frac{\Delta_{\text{in}}}{C} A_k + 2\Delta_{\text{in}} + v_k, \end{aligned} \tag{11}$$

where x_k represents the state equation, z_k represents the measurement equation, Δ_{in} is the initial probing rate, and C the capacity of the bottleneck link.

3. Simulation Setup

The analysis has been carried out on a dataset of 60 items, each one containing several scenarios simulated by using MATLAB. Each scenario is made up of 1440 samples. In detail, the algorithm first creates the simulation environment, during which it randomly generates 8 hops with different link capacity and AB and the bottleneck link along with all its AB and capacity and then it calculates the AB variation for the bottleneck.

After the configuration environment is ready, the algorithm starts the DEFK. The same configuration is run for several values of slope variation and values of the covariance matrixes of the process noise and measurement noise.

Regarding the AB variation, the algorithm starts from the AB of the bottleneck and then decreases it till reaching a lower AB calculated during the configuration phase. The slope variation is simulated by using two slopes, that is, two different numbers of samples to represent the transition: 15 samples, which represent an abrupt variation, and 360 samples. Examples of outputs of the algorithm for 15 and 360 samples are, respectively, shown in Figures 1 and 2. In detail, the two figures show AB estimated by the DEKF against the actual AB value by using a value of the covariance matrix of the process noise equal to 10^{10} and a value of the covariance matrix of the measurement noise equal to 10^{-12}. The simulation is carried out for several values of the covariance matrix of the process noise $\{10^9, 10^{10}, 10^{11}, 10^{12}, 10^{13}, 10^{14}, 10^{15}$, and $10^{16}\}$ and measurement noise $\{10^{-14}, 10^{-13}, 10^{-12}, 10^{-11}, 10^{-10}$, and $10^{-9}\}$. The algorithm performance is evaluated in terms of percentage relative error, which is defined as

$$e_{r,i}\% = \left|\frac{\widehat{\text{AB}_i} - \text{AB}_i}{\text{AB}_i}\right| * 100. \tag{12}$$

The percentage relative error between normal values provided by the PGM (AB_i) and the AB estimates by the DEKF ($\widehat{\text{AB}_i}$) provides information about how close the estimate is to eventual outcomes and permits tracking the accuracy of the AB estimation.

4. Results

The results are presented in Figures 3 and 4. The former shows the 3D bars plots of the percentage of the relative errors falling below three given thresholds. In particular such thresholds are 1% (Figures 3(a) and 3(b)), 5% (Figures 3(c) and 3(d)), and 10% (Figures 3(e) and 3(f)) for abrupt AB variations of 15 samples and slower AB variations of 360 samples. Figure 4 shows the 3D bars plots of the percentage of the relative errors for the same aforementioned thresholds and slope variations, which are limited to the AB transitions; that is, only related errors observed during the AB transition are considered to calculate these percentages.

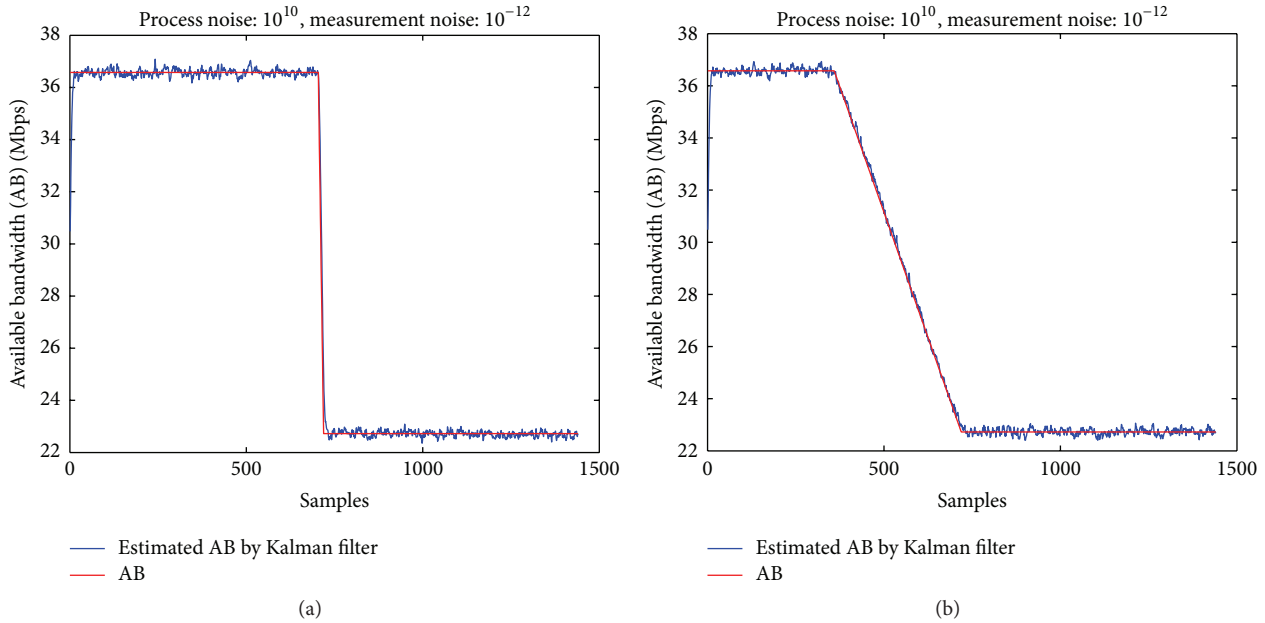


Figure 1: The AB estimated by DEKF against the actual AB value for (a) an abrupt AB variation and (b) a slower variation.

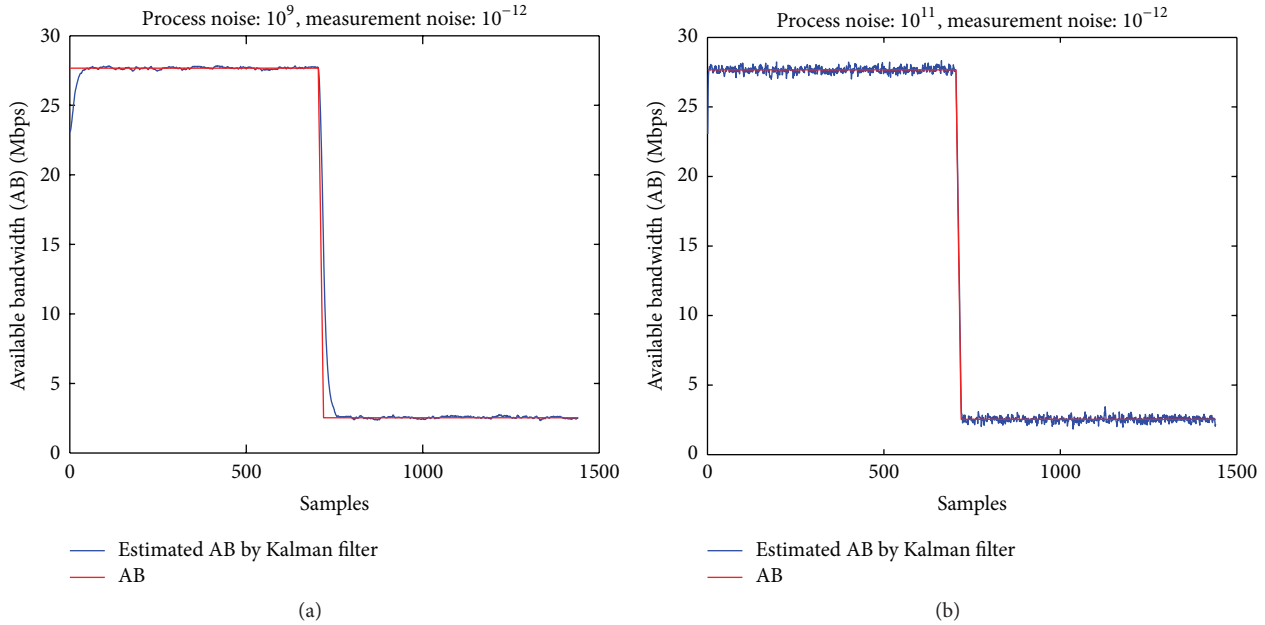


Figure 2: Comparison of the DEKF's performance for an abrupt AB variation for a value of the covariance matrix of the measurement noise equal to 10^{-12} and for two different values of the covariance matrix of the process noise: (a) 10^{9} and (b) 10^{11}.

The relative errors are plotted over a plan whose axes represent the covariance matrixes of the measurement and process noise, respectively. In detail, each bar represents the percentage of errors calculated according to (12) that fall below the considered threshold.

It is worth noting that all results depicted in Figures 3 and 4 represent average values over the 60 datasets considered for the simulations.

From Figure 3, we can observe that the percentage of the relative errors falling in the considered thresholds rises as the value of the covariance matrix of measurement noise decreases and the value of the covariance matrix of the process noise rises. In other words, the measurement algorithm exhibits better performance for high values of the covariance matrix of the process noise. The percentage of the relative error improves as the thresholds rises.

It is worth noting that the process noise of the DEKF proposed in [11] represents a degree of freedom and it can be chosen according to the covariance matrix of the measurement noise for which the DEKF algorithm shows the

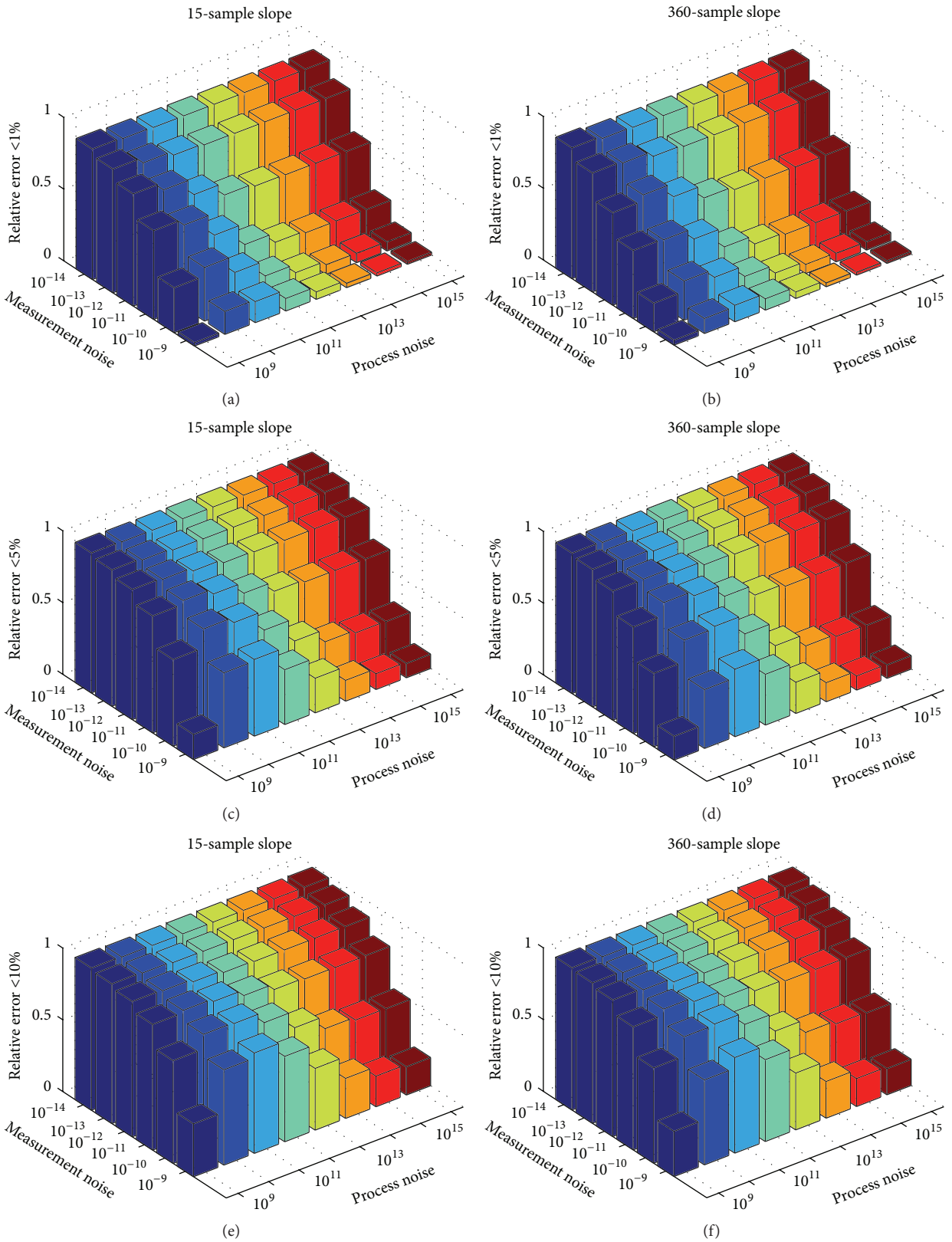


FIGURE 3: Percentage of relative errors that are below different thresholds θ for fast AB variations (a, c, and e) and slow AB variations (b, d, and f). $\theta = 1\%$ in (a) and (b), $\theta = 5\%$ in (c) and (d), and $\theta = 10\%$ in (e) and (f).

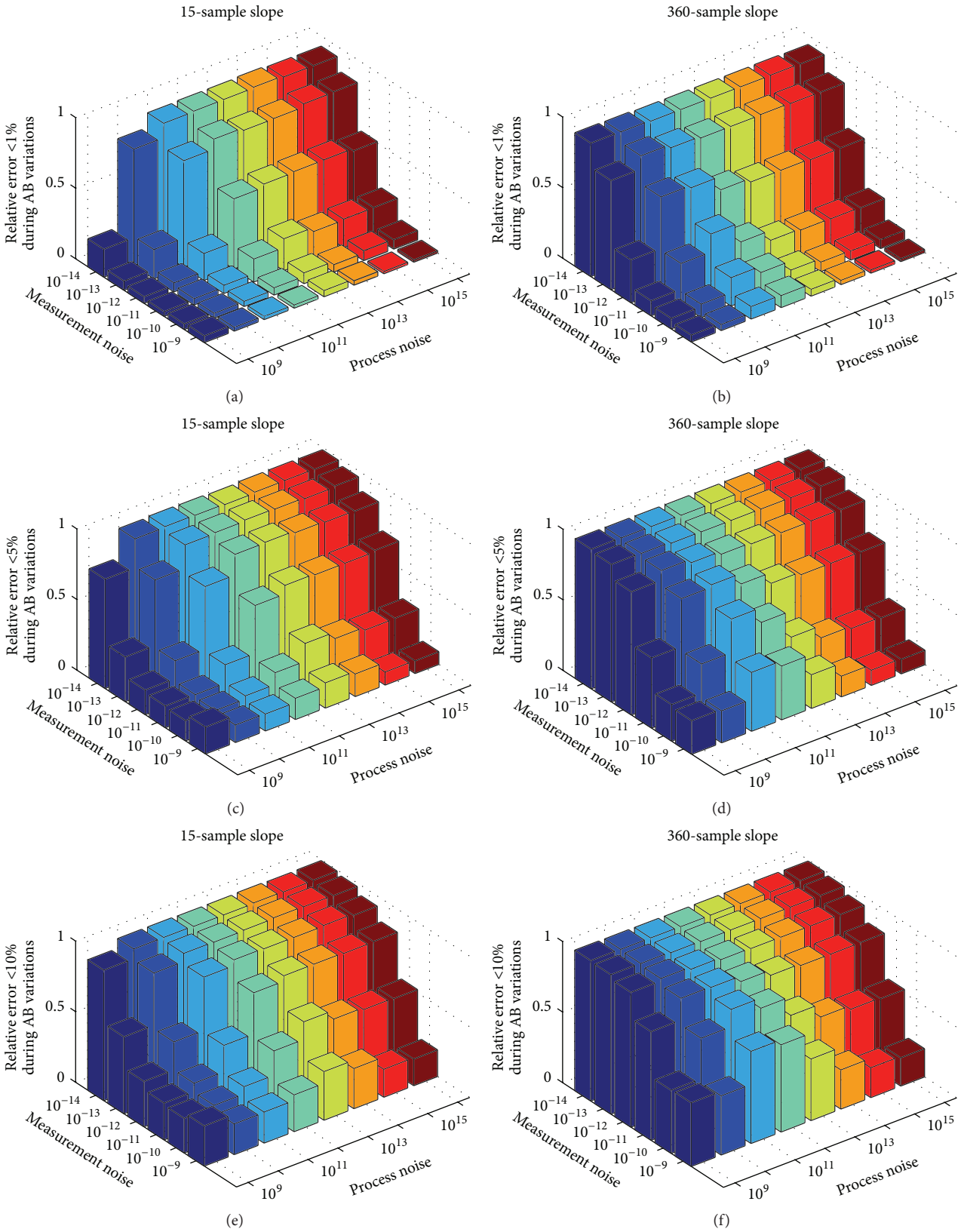


Figure 4: Percentage of relative errors that are below different thresholds θ for fast AB variations (a, c, and e) and slow AB variations (b, d, and f). $\theta = 1\%$ in (a) and (b), $\theta = 5\%$ in (c) and (d), and $\theta = 10\%$ in (e) and (f).

best performance. Therefore, once the value of the covariance matrix of the measurement noise is fixed, which depends on the network, the covariance matrix of the process noise can be chosen according to the 3D bar which shows the greatest percentage of the errors lower than the considered threshold. However, as shown in Figure 4, the performance of the algorithm gets worse for low values of the covariance matrix of the process noise during AB variations. Such a poor performance is due to the fact that DEKF is not able to follow the AB variation for low values of the covariance matrix of the process noise.

An example is given in Figure 2, which shows AB estimated by the DEKF against its nominal value when the covariance matrix of the measurement noise equals 10^{-12} and for two values of the covariance matrix, which are, respectively, 10^9 (Figure 2(a)) and 10^{11} (Figure 2(b)), during an abrupt AB variation. By comparing the two figures, it is possible to notice that the algorithm is not able to track the abrupt variation in case of a lower value of the covariance matrix of the process noise. Further confirmation is provided by comparing the figures in first column shown in Figure 4, which represent abrupt AB variations, with the figures in second column of the same figure, which represent a slower AB variation.

5. Conclusion

The paper has presented a characterisation analysis of the performance of a DEFK applied to a PGM-based measurement method for tracking AB variations in case of the fact that cross traffic is not path-persistent and exhibits both slow and abrupt variation over time.

The paper has highlighted that the covariance matrix of the process noise for the DEKF can be properly chosen in order to decrease the measurement noise and, finally, achieve a better AB estimation. Moreover, the paper has shown that, for fixed values of the covariance matrix of the measurement noise, the performance of the DEKF, in terms of relative error, improves as the value of the covariance matrix of the process noise decreases. However, the DEKF algorithm is not able to follow abrupt AB variations as the value of the covariance matrix of the process noise rises.

Further research activities will focus on the performance assessment of the measurement method on real networks and for different patterns of available bandwidth variations.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] R. S. Prasad, C. Dovrolis, M. Murray, and K. Claffy, "Bandwidth estimation: metrics, measurement techniques, and tools," *IEEE Network*, vol. 17, no. 6, pp. 27–35, 2003.

[2] D. M. Batista, L. J. Chaves, N. L. S. da Fonseca, and A. Ziviani, "Performance analysis of available bandwidth estimation tools for grid networks," in *Proceedings of the IEEE 14th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD '09)*, pp. 1–5, Pisa, Italy, June 2009.

[3] T. A. L. Genez, L. F. Bittencourt, N. L. S. Da Fonseca, and E. R. M. Madeira, "Refining the estimation of the available bandwidth in inter-cloud links for task scheduling," in *Proceedings of the IEEE Global Communications Conference (GLOBECOM '14)*, pp. 1127–1132, Austin, Tex, USA, December 2014.

[4] T. A. L. Genez, L. F. Bittencourt, N. L. S. Da Fonseca, and E. R. M. Madeira, "Estimation of the available bandwidth in inter-cloud links for task scheduling in hybrid clouds," *IEEE Transactions on Cloud Computing*, 2015.

[5] M. Imai, Y. Sugizaki, and K. Asatani, "A new estimation method using RTT for available bandwidth of a bottleneck link," in *Proceedings of the IEEE 27th International Conference on Information Networking (ICOIN '13)*, pp. 529–534, IEEE, Bangkok, Thailand, January 2013.

[6] A. Mahanti, C. Williamsonm, M. Arlittm, and A. Mahantit, "Comparing wired-side and wireless-side WLAN monitoring techniques: a case study," in *Proceedings of the 32nd IEEE Conference on Local Computer Networks (LCN '07)*, pp. 901–910, Dublin, Ireland, October 2007.

[7] N. Basher, A. Mahanti, A. Mahanti, C. Williamson, and M. Arlitt, "A comparative analysis of web and peer-to-peer traffic," in *Proceedings of the 17th International Conference on World Wide Web (WWW '08)*, pp. 287–296, Beijing, China, April 2008.

[8] A. Bondavalli, A. Ceccarelli, F. Brancati, D. Santoro, and M. Vadursi, "Differential analysis of Operating System indicators for anomaly detection in dependable systems: an experimental study," *Measurement*, vol. 80, pp. 229–240, 2016.

[9] A. Bondavalli, F. Brancati, A. Ceccarelli, L. Falai, and M. Vadursi, "Resilient estimation of synchronisation uncertainty through software clocks," *International Journal of Critical Computer-Based Systems*, vol. 4, no. 4, pp. 301–322, 2013.

[10] J. Strauss, D. Katabi, and F. Kaashoek, "A measurement study of available bandwidth estimation tools," in *Proceedings of the 3rd ACM SIGCOMM Internet Measurement Conference (IMC '03)*, pp. 39–44, Miami Beach, Fla, USA, October 2003.

[11] L. Angrisani, G. Miele, R. Schiano Lo Moriello, and M. Vadursi, "A kalman filtering based method for available bandwidth measurement," in *Proceedings of the IEEE International Instrumentation and Measurement Technology Conference (I2MTC '15)*, pp. 1215–1220, IEEE, Pisa, Italy, May 2015.

[12] December 2015, http://www.idt.mdh.se/~ajn12/index.phtml?choice=software.

[13] M. Jain and C. Dovrolis, "End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput," *IEEE/ACM Transactions on Networking*, vol. 11, no. 4, pp. 537–549, 2003.

[14] S. Suthaharan and S. Kumar, "Measuring available bandwidth: pathChirp's chirp train structure remodeled," in *Proceedings of the Australasian Telecommunication Networks and Applications Conference (ATNAC '08)*, pp. 379–384, Adelaide, Australia, December 2008.

[15] N. Hu and P. Steenkiste, "Evaluation and characterization of available bandwidth probing techniques," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 6, pp. 879–894, 2003.

[16] V. Ribeiro, M. Coates, R. Riedi, S. Sarvotham, B. Hendricks, and R. Baraniuk, "Multifractal cross-traffic estimation," in *Proceedings of the ITC Specialist Seminar on IP Trac Measurement,*

Modeling, and Management, Monterey, Calif, USA, September 2000.

[17] L. Lao, C. Dovrolis, and M. Y. Sanadidi, "The probe gap model can underestimate the available bandwidth of multihop paths," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 5, pp. 29–34, 2006.

[18] M. S. Grewal and A. P. Andrews, *Kalman Filtering. Theory and Practice*, Prentice Hall, Englewood Cliffs, NJ, USA, 1993.

[19] G. Welch and G. Bishop, *An Introduction to the Kalman Filter*, University of North Carolina at Chapel Hill Department of Computer Science, Chapel Hill, NC, USA, 2001.