Dottorato di Ricerca in
Computer Engineering and Science

Scuola di Dottorato in
Information and Communication Technologies

XXIV CICLO

Università degli studi di MODENA e REGGIO EMILIA
Sede Amministrativa di Modena

TESI PER IL CONSEGUIMENTO DEL TITOLO DI
DOTTORE DI RICERCA

Collaborative architectures for intrusion detection

Candidato:
Ing. Michele Messori

Relatore:
Prof. Michele Colajanni

Coordinatore del dottorato:
Prof. Andrea Prati

Direttore della scuola:
Prof. Giorgio Matteo Vitetta

February 2012

“We could be...
Sitting in the computer lab,
4 A.M. before the final paper is due,
Cursing the world ’cause I didn’t start sooner”

Avenue Q

Abstract

The widespread diffusion of new network security threats and the large scale of today attacks pose new challenges for the protection of networks and systems. There is a strong need for computer security infrastructures that are able to identify network intrusions and new malware as soon as possible. In particular, we claim that distributed network monitoring approaches are best suitable to react to inherently distributed threats, such as self-replicating malware and intrusions leveraging widespread vulnerabilities.

In this work, we present collaborative architectures suitable to large networked systems and mobile environments. There are three main contributions in this thesis.

- We introduce a new architecture suitable to large organization networks which leverages hierarchical and peer-to-peer cooperation schemes among distributed Network Intrusion Detection Systems (NIDSs). Its main objective is to facilitate cooperation among different networks by reducing the amount of alerts and by guaranteeing privacy. These features allow a system administrator to focus on the few alerts that represent a real threat for the controlled infrastructure and to be informed about the most dangerous intrusion(s) even before his department is attacked. Moreover, they make it possible to restrict information sharing concerning sensible data through a configuration of the privacy policy.

- We propose a novel distributed architecture for collaborative detection of cyber attacks and network intrusions based on Distributed Hash Tables (DHT) and Complex Event Processing (CEP). The main functional blocks of this architecture are the event sources, geographically distributed among different organizations, the collaboration layer, realized through a DHT, and the engine for CEP. We implement a prototype validated in a realistic scenario of Man-in-the-Middle attacks. The design of the architecture supports different CEP engines and is not tied to a specific event processing logic.

- We consider the mobility contexts which are facing a rapid diffusion of emerging threats. We found a new type of attack, called mobility-based

NIDS evasion technique, which allows a malicious user to exploit node mobility to perform stealth network attacks. They are undetectable even by the state-of-the-art NIDSs having stateful capabilities. The attack targets roaming events of Mobile Nodes and affects all protocols supporting seamless mobility, such as Wi-FI, Mobile IPv4 and Mobile IPv6. We introduce a new collaborative defense strategy based on the exchange of state information among NIDSs deployed in different networks. We designed and implemented a complete framework supporting the most popular protocols for node mobility that is based on Snort. The modular design guarantees great flexibility in terms of deployment and of future extension to possible new mobile protocols. The performance evaluation of the framework carried out in realistic scenarios demonstrate the efficacy and efficiency of the proposed solution that represent a clear advancement of the state of the art.

Acknowledgments

If today I am here writing these words, it is thanks to many people that supported me during my studies. Firstly, I would like to thank several professors that I will remember with pleasure: Fabrizia, she always taught math in a very funny way, Claudina, an extraordinary person always ready to help others, Vanna, if more professors were like her there would be better engineers, and last but not least prof. Michele Colajanni. He has been my adviser since my Laurea degree (BSc). He introduced me to a fascinating world and allowed me to join a great research group. I want to thank Claudia, Mauro, and Riccardo, that were working in their office, but (almost) always ready for suggestions, annoying bureaucracy issues, and coffee breaks.

I would like to thank all those who shared with me the daily lab life. Turoldo (also known as Mirco) has been a guide for me in the world of computer science and, in particular, of network security. He taught me a lot of interesting things and also how to count coffee cups! Sara, who listened to a lot of my vents and encouraged me all the times, in few words: wide shoulders to rely on! Framazz, who shared the desk with me, including spaghetti cables and candies. She has also been my personal debian updates tester and her spell checking capabilities have been extremely useful. Elton, probably the best representative a student can have. We attended so many courses and seminars together that I cannot remember them all. Stefy, a great and promising PhD student, Marcello, that inherited some of my duties, and Luca, who stayed with us for almost an year and now I met him again.

I am extremely grateful to all of them and I am very happy to have found such great friends.

I wish to thank my family too. Dad, you passed your passion for computers and technological gadgets down on me. I will never forget the Saturday afternoons spent going to trade fair of electronics. Mom, my achievements would not have been possible without your support, including the time you spent helping me and my sisters doing our homeworks when we were children. Alice, it has been a real pleasure to be your older brother and, starting from elementary school to university, to travel together by scooter, by car, by bus, and by train. Giulia, you have always been my pretty younger sister and I really hope you will decide how

to be happy. Nonno, I always been proud to have a graduated grandfather. Mel, you joined my life before the beginning of the doctorate school, since then you supported and inspired me. I will never forget my first freedom kiss.

Finally, exactly one year ago, my uncle left us after a long battle with illness. This thesis is dedicated to Woody.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays Internet has definitely penetrated our life in depth enhancing our communication capabilities. Every day billions of devices connect to the network of networks allowing billions of people to work, to contact friends and to perform a myriad of other activities. However, information shared through these networks and the data stored by users on their machines represent a valuable asset that needs to be protected from malicious users, criminal organizations and hostile governments, which may target these digital properties in order to make profit and control, or weaken, their adversary, respectively.

Several proposals have been introduced to protect information systems and computer networks. In particular, an important and widespread defensive solution is represented by Intrusion Detection Systems, that aim to detect different kind of malicious activities, such as intrusions, attacks and propagating malwares. Since their early introduction [13, 42] IDSs have evolved enhancing their monitoring capabilities and improving their detection rate in order to thwart new attack strategies and to adapt to heterogeneous and dynamically changing environments. In particular, to better react to threats whose nature is inherently distributed (such as self-replicating malware, or intrusions leveraging widespread vulnerabilities), collaborative approaches are best suited due to their ability to gather information from a wide network space, thus allowing for early detection of emerging threats. This approach allows administrators to deploy timely countermeasures because all the network segments hosting at least one sensor can be alerted about new threats as soon as they are detected in any part of the collaborative system. However, several problems affect existing collaborative solutions. Indeed, while centralized solutions can leverage distributed sensors [60, 99, 110], the single aggregator may represent a bottleneck and actually is a single point of failure. Hierarchical architectures [41, 83, 102, 106, 116] improved scalability introducing different aggregation layers that separate distributed NIDS from the top of the chain. This solution partially distributes the analysis workload among the multiple aggrega-

tors, but it still presents a single point of failure and suffers limited scalability. The latter issues are avoided by peer-to-peer solutions [45, 56, 68, 70, 118] that guarantee fault tolerance and great scalability, but most of them focus only on specific threats.

In this thesis we consider collaborative architectures for intrusion detection that leverage distributed components. In particular, we focus on three different scenarios characterized by particular requirements and constraints:

- Large networked systems
- Cooperation among different organizations
- Mobile environments

In order to monitor large networked systems, centralized NIDSs are not effective, thus it is necessary to deploy a distributed architecture consisting of multiple cooperative NIDSs. However, the deployment of multiple NIDSs implies the generation of a huge amount of alerts, where the few relevant alerts related to real security threats are overwhelmed by irrelevant ones. System administrators have to analyze each event looking for those related to real attacks, but this represents a time consuming task that reduces the efforts spent on deploying effective countermeasures. Our proposal consists in a hybrid architecture suitable for multi-department, which integrate an alert ranking system able to highlight critical alerts [35]. This novel architecture leverage hierarchical communication scheme for intra-department communication and a peer-to-peer overlay for inter-department information sharing. This hybrid approach aims to facilitate cooperation among different departments, while the alert ranking system reduces the number of alerts requiring manual intervention of system administrators. Furthermore, the integration of these systems allows administrators to be notified about especially critical attacks that targeted other departments. We also provide some privacy preserving features that filter alerts containing particular data in order to avoid dissemination of sensitive information.

Cooperation among different organizations poses new constraints that require a specific design of the collaborative architecture. In particular computational power and data filtering represent critical issues. The former has to meet the levels required to analyze a very large number of events, which increase with the number of collaborating organizations, while the latter is influenced by the possibility that two or more organizations are competitors. In this case, preprocessing of the events generated by the organizations is two folding: filtering of sensitive data (i.e. dropping events before submission or anonymizing their content) in order to avoid information leakage, and normalization of submitted events, which

allows heterogeneous IT infrastructures protected by different IDS solutions to share their events. Considering these issues, we propose a distributed collaborative architecture where several Event Sources are deployed within each cooperating organization, while a Processing Container analyzes a huge amount of data leveraging multiple Processing Elements connected by a Distributed Hash Table overlay [49]. The former component is able to perform filtering, anonymization an normalization of heterogeneous events generated by the host organization. The latter elements analyze received events performing Complex Event Processing and generate alerts related to the actual threats. Furthermore, they share events and other information through a Distributed Hash Table that provides fault tolerance to the whole Processing Container and equally distributes the events among the active Processing Elements. The communication between Event Sources and Processing Container are managed by a Communication Middleware.

A new scenario which is facing a rapid diffusion of emerging threats is represented by mobile environments. Indeed, the number of mobile devices connected to Internet is constantly increasing and the network administrators have to face different challenges to manage and protect mobile nodes themselves and the rest of the network from the mobile nodes. In this thesis we introduce a new attack strategy, called mobility-based NIDS evasion technique [31], which allows malicious users to exploit packet fragmentation and node mobility to perform "stealth" network attacks, undetectable even by the state-of-the-art NIDSs having stateful capabilities. Despite it leverages fragmentation, this evasion technique is not caused by flaws in any NIDS implementation [78, 85, 96]. On the contrary, it targets roaming events and the only requirement is that at least one between attacker and victim roam from one network to another, while keeping alive the communication with the other node (i.e. without connection interruption). Thus, all protocols supporting seamless mobility [53] are subject to attacks exploiting this new evasion technique. In particular we describe the specific details of this attack for three different protocols: Wi-Fi [9], Mobile IP [81], and IPv6 [82]. Then, we propose a new defense strategy based on the exchange of state information, namely state migration, among NIDSs deployed in different networks (in particular those joined by the mobile node). The idea is to enable NIDS to gather all the data related to the Mobile Nodes network activities, including those happened within a different network. This approach provides the NIDS with all the information necessary to detect network-based attacks [36]. In order to validate our proposal, we implemented a complete framework which extends Snort [100], a widespread open source IDS. The implementation is based on a lightweight agent and a set of plugins, which are in charge of managing different protocols. This modular design guarantees great flexibility in terms of deployment and even of future extension in the case of new mobile protocols. The performance evaluation of the framework conducted in realistic scenarios demonstrates the suitability and efficacy of

the proposed solution, that represents an important step ahead with respect to the state of the art.

Finally, it is worth to be highlighted that all the proposals of this thesis have been experimentally validated in realistic scenarios. The hybrid architecture and the alert ranking system proposed in the first scenario, the geographically distributed architecture for cooperation among different organizations, and the framework for intrusion detection in mobile environments, have been designed and implemented from scratch or, when possible, through Open Source software, such as Snort [100], Prelude [84], Past [44], Scribe [91], Esper [48], Dynamics [46], radvd [93] and mip6d [16].

The rest of this thesis is organized in 5 chapters.
In Chapter 2 we introduce a brief taxonomy of Intrusion Detection Systems followed by an overview of the state-of-the-art.
In Chapters 3 and 4 we introduce our collaborative proposals for large organizations and multiple organizations, respectively. We present the requirements of these scenarios and then propose two novel collaborative architectures. Both the proposals have been implemented and validated running different experiments.
In Chapter 5 we present our research on mobile environments. We firstly introduce a new NIDS evasion strategy discovered by us and the necessary countermeasures. We also describe the details with respect to three protocols supporting mobility: Wi-Fi, Mobile IPv4 and Mobile IPv6. We then propose a complete and modular framework able to thwart the new NIDS evasion technique. The framework has been validated in a controlled environment and a performance analysis is provided.
Conclusions are finally drawn in Chapter 6.

Chapter 2

State-of-the-art

In this chapter, we firstly introduce a brief taxonomy of Intrusion Detection Systems, focusing on the detection models and the targets, then we present an overview of the state-of-the-art, considering three different aspects that are related to the proposals of this thesis.

2.1 Taxonomy of Intrusion Detection Systems

Intrusion Detection Systems (IDSs) are defensive solutions whose aim is to raise alerts when an attack, an intrusion or any kind of malicious activity, is detected. In particular, alerts generated by an IDS can be either real positives, which means the IDS really detected something which may compromise system security, or false positives, which are the equivalent of false alarms. Depending on their characteristics, Intrusion Detection Systems can be classified into different categories and several works have been published providing complete taxonomies [18,40,65,73]. However, a complete analysis of these taxonomies is out of the scope of this thesis, then we introduce only few concepts that can help the reader to better understand these systems. The four main characteristics used to classify the IDS are:

Target that represents the target of the analysis

Detection method that is the approach used for the analysis

Behavior on detection that describes the response of the IDS to a detected attack

Usage Frequency that consists of only two values: periodic analysis, also called batch mode, and continuous monitoring, also called real-time analysis

In the next subsections, we provide further details about the first two elements of the list: the target and the detection models.

2.1.1 Host-based, Network-based and Operational IDS

Depending on the target, the detector, which performs the analysis, requires different type of data, hence accessing different data sources. The IDS can be classified into three different categories: host-based, network-based or operational IDS.

Host-based IDS

A Host IDS monitors both the state and the behavior of a single machine. The former task is usually performed by analyzing system logs produced by the operating system and, possibly, by other applications, while the latter task requires dynamic analysis of resources and system calls. Host IDSs offer a detailed overview of the machine they are monitoring and they are able to detect and block malicious activities, such as modification of configuration, or system files. Thus, they may prevent installation of malware on the monitored host. However Host IDSs also affect the performance of the machine, in particular, due to the growing complexity of the modern operating systems, the system calls analysis heavily impacts on available resources. Interesting open source Host IDS are OSSEC [2] and samhain [6]

Network-based IDS

A Network IDS (NIDS) monitors networks or network segments, analyzing network traffic captured through one or more sensors. It analyzes a large amount of data looking for illicit or suspicious network activities and for malicious network packets. Network IDSs represent an effective defense solution against network-based attacks and are widely deployed to protect IT infrastructures. With respect to the Host IDSs, the main advantage of Network IDSs is the possibility to monitor several machines by deploying only one component without affecting either host or network performance. On the other hand, NIDSs suffer from different drawbacks:

- NIDSs are not able to analyze internal state or dynamic behavior of the monitored machines;
- encrypted traffic can be analyzed only providing the encryption keys to the NIDS, which is feasible only under certain conditions;
- monitoring of modern high speed networks (up to tens of Gb/s) requires significant resources and fine tuning;
- network traffic analyzed by the NIDS may be different from the traffic received by monitored hosts, due to the network architecture and the operating systems installed on the hosts;

In particular, the latter is related to the inner working of the NIDS. Indeed, in order to monitor a network, the system needs to track each distinct connection, and reassemble and analyze every network packet transmitted over the monitored network. This is a sensitive task and represents the main target of several NIDS evasion techniques [78, 85, 96]. The most famous open source Network IDSs are Bro [1], which provides interesting features, Suricata [7], designed to leverage parallelization of modern architectures, and Snort [100], the most widely deployed open source NIDS.

Honeypots

A honeypot is used to collect new malware samples, and to trace and deflect the malicious activities of an attacker, performed to abusively access an information system. It consists of a closely monitored machine, which is not intended to offer useful services to normal users, hence any attempt of reaching it and logging into its services can be considered an attack. Thus, the deployment of a honeypot within a network has a twofold purpose: to drive the attackers to waste time on it instead of on production machines, and to study attackers activities in order to acquire useful information on new attack strategies suitable for designing effective countermeasures. Depending on the capabilities offered by the system, honeypots can be divided into three categories:

- Low interaction honeypots. They offer few and simple services, such as connection management and automatic download of payload from submitted urls. A skilled attacker can easily detect anomalies during the interaction with these elements, however they are still effective against several self replicating malware. Furthermore, this kind of honeypots are based on single processes, thus they require low resources, enabling the deployment of multiple honeypots on the same machine, and are less exposed to security breaches.

- High interaction honeypots. They emulate the activities of real systems offering to the attackers the same services and almost the same interaction level. These solutions are often implemented as virtual machines, hence enabling quick recovery in case of compromised machine, or generic failures, and allowing the deployment of a variable number of honeypots on a single machine, limited only by the underlying hardware. However, with respect to low interaction honeypots, this type of honeypot requires more resources and a careful configuration. Moreover, if compromised, they provide several instruments that the attacker can leverage to perform malicious activities, from within the network.

- Pure honeypots. They are full featured systems, which directly interact with the attackers, possibly storing limited, but interesting, information, in order to fake perfectly functioning production machines and to make harder for an attacker to understand it is an honeypot. These honeypots require high resources, both in terms of hardware and maintenance, and, if compromised, may represent a bridgehead to launch further attacks.

2.1.2 Signature-based or Anomaly-based Detection

Depending on the detection model adopted by the IDS, it is possible to classify these systems in two different categories: anomaly-based systems and signature-based systems, also known as misuse-detection systems.

Signature-based Systems

In order to detect malicious content, a signature-based IDS requires a description of the attack in the form of a signature usable for pattern-matching, then it analyzes retrieved content looking for portions of data which match one of those signatures. If a match exists, then the IDS raises an alert, which contains the signature that successfully matched the analyzed content and, possibly, a description of the attack in human-readable format. The two main advantages of signature-based IDS are the easy deployment of these solutions in any scenario, eventually requiring some tuning to improve performance, and the low false positive rates. Under certain conditions, they are also able to detect previously unseen attacks, but only if these attacks contains data equal to already known patterns. Indeed, they are inherently unable to detect polymorphic attacks, where the same attack comes with different payloads, and truly novel attacks, also known as zero-day attacks. For this reason signature-based IDSs require constant updates consisting of new and updated signatures, which are automatically generated by specific tools [61, 63, 76] or manually written by skilled human operators.

Anomaly-based Systems

Anomaly-based Intrusion Detection Systems analyze data looking for abnormal observations. The assumptions are that normal behavior of a system can be modeled and that any malicious activity differs from the model. The main advantage of anomaly detection is its ability to detect also truly novel attacks. On the other hand, it suffers from different drawbacks. Before deployment in production environment a certain period is required to model the normal behavior using a series of representative dataset. This operation can be manually performed by human operators or, in case of self-learning solutions, the IDS can autonomously create

the model. Furthermore, the model requires updates in order to track the natural changes of the normal activities. Another important drawback is the high rate of false positives and false negatives (malicious activities not detected). In particular, the former is caused by abnormal activities related to non malicious activities, while the latter is related to attacks that the IDS is not able to distinguish from normal activities, due to their similarity. In order to solve these problems, several works have been proposed [27, 50], however anomaly detection is still a very active research field.

2.2 Alerts management

For the sake of simplicity, in this section we refer to Network IDS, which are the most deployed IDS solutions, avoiding to distinguish from Host IDS. However, several of the following statements concerning alerts management are valid for both NIDS and HIDS. When a Network IDS detects a malicious payload transmitted over the monitored network, an alert is immediately generated and sent to the system or network administrator. Each alert contains several useful information, ranging from the IP address of the machine that has been targeted by the attack, to the vulnerability ID, related to the signature that matched the malicious payload (only in case of signature-based NIDSs).

The two main problems faced by system and network administrators, are the large amount of alerts generated by the deployed IDS and the, possibly high, false positive rate. While the latter problem mainly depends on the type and the quality of the detection method, as described in the previous Subsection 2.1.2, the former depends on several factors, such as the dimension and the characteristics of the monitored IT infrastructure. In both cases the effect is the same: the difficulty for the administrator to focus on important alerts, wasting time on analysis of useless alerts. Moreover several techniques exist which allow skilled attackers to cover up the attack traces by forcing a NIDS to generate storms of irrelevant alerts [75, 77]. Also if the NIDS correctly detects the attacks and generates related alerts, critical alerts are hidden among a huge amount of irrelevant and misleading alerts, as a needle in a haystack.

In order to help administrators to face this problem, several works have been proposed describing different techniques such as *alert verification* and *alert prioritization*. The former refers to all mechanisms that can help to determine whether an attack was successful or not, thus possibly reducing the number of alerts showed to the administrator. The latter aims to classify alerts based on their severity, in order to let the administrator focus on the most important categories. An interesting work has been proposed by Kruegel et al. [64, 104], which provides both prioritization and active verification based on a single vulnerability database. However,

the dependency on a single information source is identified as a drawback in [86], because of possible incompleteness, lack of timely updates, and lack of trust in the single information provider. Furthermore, despite the active verification allows to use updated information on current state of the machines, it may have detrimental effects on scanned machines and the monitored network.

2.3 Distributed IDS

With the increase in complexity of IT infrastructures, new solutions have been proposed to help administrators monitoring these environments. In particular, distributed and collaborative systems are emerging as the most valid solutions for large organization networks, which have to face modern threats coming from multiple sources. The first approaches proposed for distributed IDS were centralized solutions [60, 99, 110]. The main idea is to deploy several distributed sensors that collect data from different sources and then send all the events to a single aggregator that analyzes and possibly correlates all the available information and generates high level alerts. However, with the growing size of the monitored environments, the computational capacity of one centralized alert aggregator becomes a bottleneck. Hierarchical architectures improved the performance distributing the sensors and introducing different aggregation layers reducing the load on the top of the chain. Emerald [83] is one of the first proposal based on a hierarchical architecture that provides service analysis at domain level and enterprise level (i.e multi domain). GrIDS [102] is a Graph-based IDS that collects network data and monitors computers activity, then it aggregates this information into activity graphs in order to reveal the causal structure of network activity. NetSTAT [106] uses the state transition analysis approach and, by modeling both the network and the attacks, is able to focus on the relevant events and to generate a minimal amount of network data. Debar et al. [41] proposed a solution able to detect duplicates and to aggregate alerts using a series of "situations" that group events having common specific characteristics. HIDE [116] is an anomaly-based NIDS, with hierarchical architecture, which uses statistical models and neural network classification.

To avoid the single point of failure typical of both centralized and hierarchical architectures, several peer-to-peer solutions have been proposed, such as [45, 56, 68, 70, 118]. INTCTD [45] is based on a neural network and is able to dynamically modify the network resource access policies on the base of network traffic anomalies. Worminator [68] is focused on detection of worms. It shares information related to IP addresses and ports using Bloom Filters, which guarantees an high level of privacy. The same target but a different approach is proposed by [70]. It leverages a DHT overlay to exchange snapshots of the internal be-

havior of the peers, consisting in traces of the executed system calls, in order to detect suspicious sequences which can be caused by the propagation of a worm. A combination of peer-to-peer and hierarchical components represents the overlay network of DOMINO [112] that is able to detect attacks based on spoofed IP addresses and to distribute alerts information among different domains. Also the scheme proposed in [118] allows peers to exchange information related to IP addresses that each peer considers suspicious. Then, integrating several lists of suspicious addresses the system is able to pinpoint sources that are malicious with high probability and use them to build a collaborative blacklist. A different communication scheme is proposed by Indra [56], which uses trusted peers to guard the network as a whole, and shares information, gathered by custom sensors, using a publish/subscribe scheme.

Most of these solutions use high level information, which are shared after different preprocessing steps, such as filtering and aggregation, applied to reduce the amount of transferred data and increase the intrinsic value. However, other proposals exist that share raw data, such as portions of the internal state information of the NIDSs. A brief overview of these solutions is provided in the next Subsection 2.3.1.

2.3.1 Network IDS exploiting state migration

Network IDS exploiting state migration represents a little niche in the world of distributed IDS, and most of them are focused on parallel NIDS architectures. Different cooperation schemes [14, 32, 34, 101] have been proposed in order to allow the exchange of state information among locally distributed NIDSs, where each node analyzes a small fraction of traffic gathered from the network. [101] proposes a cooperation technique specific for Bro NIDS sensors. The aim is to synchronize one or more NIDS internal variables that are identified in the NIDS configuration, thus generating a pool of variable values that are shared among all the cooperative NIDS sensors distributed on different network links. Every change in the value of a synchronized variable causes a sensor to send an update message to all the cooperative sensors through a push-based model. The same mechanisms for coordinating lower-level analysis was also used to realize a NIDS cluster [105]. A different approach can be found in the framework proposed in [14, 32, 34]. It provides methods to export/import complete state information from/into a NIDS, and it is generally applicable to any NIDS. Although the latter solutions fit better to a large scale scenario with respect to the push-based model proposed in [101], all these proposals leverage NIDS internal state migration to reach the same goals: to balance workload among multiple NIDS and improve the monitoring performance.

2.4 IDS for mobile networks

A particular scenario for Intrusion Detection Systems is represented by mobile environments. Most of the solutions proposed for this context focus on wireless ad-hoc networks, such as MANETs, which present different characteristics with respect to the infrastructure network, thus they require particular approaches. Thamilarasu et al. [103] propose a Cross-layer Intrusion Detection System (CIDS) in order to mitigate DoS attacks in Ad-hoc networks, in particular their main targets are: Collisions, Misdirections and Packet Drops. The cross-layer design enables to detect intrusion at multiple levels of the protocol layers and to exploit the information from one layer, to detect intrusion in another layer. They also claim to improve accuracy of the IDS, thus reducing the false positive alerts. Zhang et al. [114] propose a distributed and cooperative IDS where every node participates with its resources. An IDS agent running on each node is responsible for monitoring local activities and communicating with neighbors IDS agents in order to provide both detection of local events and global intrusion detection. In order to provide a distributed and cooperative IDS, Albers et al. [12] propose to use mobile agents. A Local Intrusion Detection System (LIDS) is installed on every node to monitor local activities while a series of mobile agents report to the original node the information gathered on the other nodes. Then a Local Management Information Base is in charge to analyze received data and detect anomalies or match signatures of malicious activities. Mobile Agents are also at the base of the solution proposed by Kachirski and Guha [59]. The main idea is to separate the agents in three categories: Host Monitoring Agents, Network Monitoring Agents, Decision Agents and Action Agents. All the nodes are divided into clusters. Every node runs a Host Monitoring Agents which monitors local activities and an Action Agents able to react to suspicious events (i.e. by terminating related processes). Some nodes, which represent the cluster heads, also run an agent which monitors network activities and a Decision Agent which analyzes network packets and possibly other information received by the local monitors deployed on the other nodes. If the Decision Agent detects a malicious node, it asks the Action Agent running on that node to react.

All these proposals present interesting approaches to intrusion detection in mobile environments, but, as mentioned before, they are focused on wireless ad-hoc configuration. On the other hand, some proposals for infrastructure wireless networks exist, such as those focusing on detection of rogue access points [22, 69, 108]. However, for the best of our knowledge, no solutions have been proposed for monitoring mobile nodes within infrastructure networks.

Chapter 3

Selective and early threat detection in large networked systems

3.1 Introduction

In the last decade, the portion of network attacks carried out by self-replicating malware and automatic tools overtook the portion of those manually performed by skilled people. While the latter select a target at once and focus on it, the former typically scan whole LAN networks or random Internet addresses looking for vulnerable systems and attack them exploiting their weakness. Moreover, several attacker tools exist that are able to automatically trigger so called *alert storms* [75, 77], by reproducing fake attacks specifically build to match signature-based defensive solutions. In this scenario, effective management of security alerts generated by network intrusion detection systems represents a critical problem. NIDSs analyzing high speed network links, or networks connecting several nodes, produce a huge amount of irrelevant alerts among which the few relevant alerts related to real security threats are hidden, as needles in a hay stack. From the system administrator's perspective, these alarms are considered as false positives, but a NIDS alone is unable to assess whether they represent a real threat for the protected machines. It is the network administrator responsibility to examine each of the incidents reported by the NIDS, as well as to devise and deploy the proper countermeasures. Hence, administrators are forced to waste precious time for a manual analysis of irrelevant security alerts.

Managing a high volume of alerts constantly generated by NIDS gets worse when monitoring large network-based information systems. Indeed, complete coverage of a realistic network environment can only be achieved by implementing a distributed architecture for intrusion detection, comprising multiple cooperative NIDSs deployed throughout the protected network. However, the multiplica-

tion of NIDSs implies a growth of the total number of alerts generated within the protected network, and that requires manual validation and analysis by the system administrator.

In this chapter we propose an innovative solution suitable for large organization networks [35]. We consider an infrastructure composed by several interconnected departments, each comprising a multitude of smaller network segments connecting hundreds of nodes. The proposed architecture enhances the cooperation of distributed NIDSs deployed in several network departments, by integrating three novel features:

- A novel distributed alert ranking scheme, that is able to analyze NIDS alerts in order to rank them on the base of whether they represent a real threat or a false positive.
- A hybrid architecture which leverages both hierarchical and peer-to-peer communication schemes for intra-department and inter-department cooperation, respectively.
- A privacy preserving alert sharing service which is in charge to selectively propagate early warnings among cooperating departments by disseminating important alerts related to critical components and filtering all those that contain sensitive data.

The proposed alert ranking scheme supports different external data sources in order to gather information related to known vulnerabilities, it is able to rank alerts at run-time and to share the computational workload among several distributed components. We also introduce an alert sharing service which analyzes already ranked alerts looking for critical ones which represent a real threat for vital components of the monitored infrastructures. Even though there are no machines within the department related to the raised alert, critical systems vulnerable to the detected attack may be deployed within other departments, then it could be a precious help to share this kind of information. On the other hand, alerts generated by NIDSs may leak, directly or indirectly, sensitive data. In this scenario, the proposed architecture is able to selectively issue early warnings to the administrators of the departments hosting the vulnerable critical systems, allowing them to deploy proactive security countermeasures and avoiding to disseminate sensitive information.

In order to support these features, we introduce a novel hybrid architecture which take advantage of both hierarchical and peer-to-peer communication schemes to enable cooperation among distributed components deployed in different network segments of large organizations. The advantages of this design are a higher scalability and fault tolerance with respect to purely hierarchical architectures [20,

21,60,98,99,107,110,115] and a better management of the information flow compared to purely peer-to-peer architectures [19, 28, 33, 41, 87, 102, 111, 116, 117]. This solution is suitable for large organizations, where the hierarchical architectures monitor different network segments within single departments, while the peer-to-peer cooperation scheme is used to connect all the departments. To the best of our knowledge, this is the first proposal of a hybrid architecture for monitoring complex networks belonging to a large organization.

An overview of the proposed architecture is presented in Section 3.2. The alert ranking system is described in Section 3.3 and the details of the cooperation scheme adopted by distributed NIDSs are explained in Section 3.4. A prototype implementation of the proposed architecture is described in Section 3.5. Concluding remarks are given in Section 3.6.

3.2 Architecture overview

In this chapter we propose an architecture suitable for network infrastructure of large organizations. These complex infrastructures are composed by geographically distributed departments, each of them consisting of multiple network segments.

In this scenario, only the deployment of distributed IDS and NIDS guarantees the complete monitoring of the whole infrastructure. Then, the alerts generated by these distributed sensors need to be aggregated and correlated in order to represent valuable information for system administrators. Different solutions have been proposed to perform these tasks. Centralized architectures are based on a single correlator, which aggregates and analyzes the alerts received by all the sensors. While these architectures are simples to deploy, they present two critical drawbacks: they are not scalable, indeed the centralized correlator could represent a bottleneck, and they provide low dependability, due to the single point of failure. Moreover, a large complex network divided into different departments is reasonably managed by different system administrators, at least one for department, responsible for all the networks belonging to their departments. Indeed, alerts containing sensitive information may be considered as classified, thus it is preferable to avoid a department administrator to directly access security warnings related to other departments.

Hierarchical architectures provide some enhancement introducing local aggregators, which improve scalability and may offer filtering features to avoid forwarding of sensitive data, but they still present a scarce dependability, due to the single point of failure represented by the top correlator.

We propose a hybrid architecture which combines hierarchical and peer-to-peer NIDS cooperation schemes, together with distributed alert ranking servers

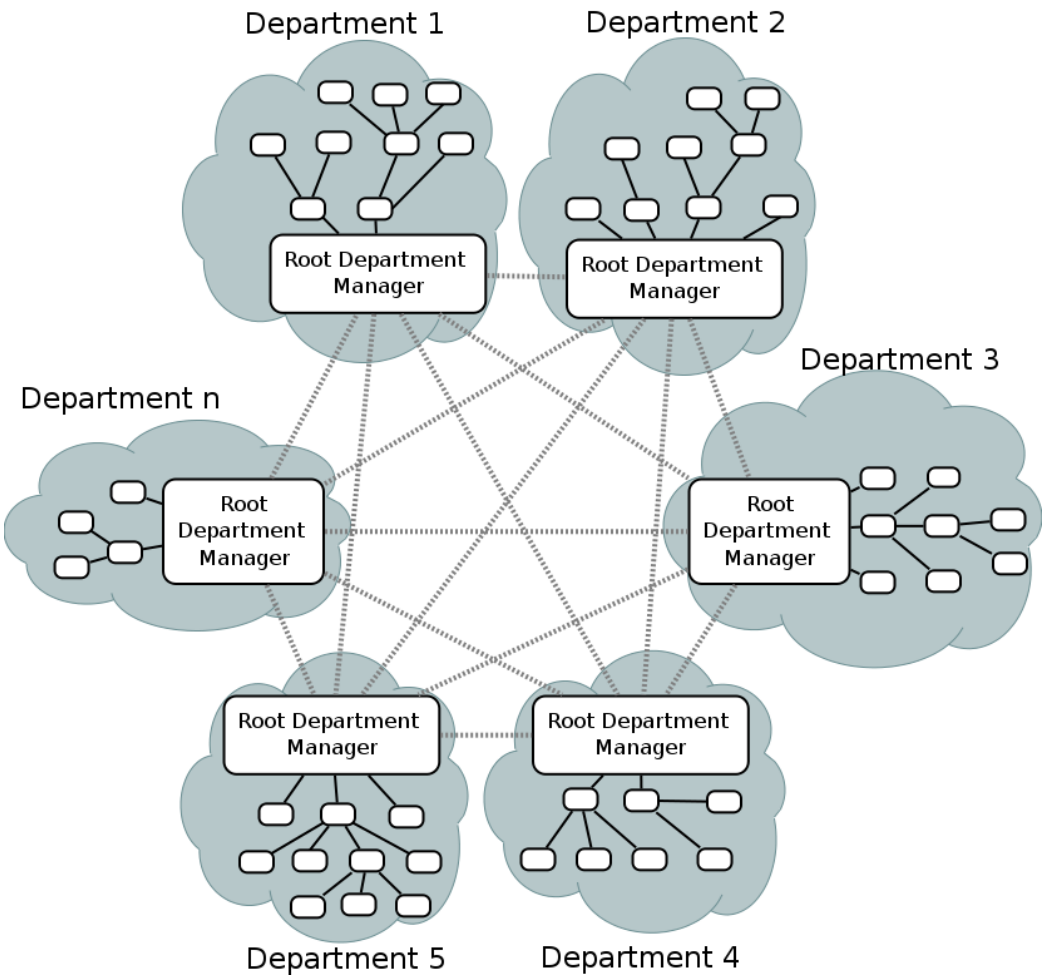

Figure 3.1: High-level view of the proposed architecture

(see Section 3.3), to achieve resiliency to failures, high scalability, and the flexibility required by complex network environments. The alert ranking system enables selective and early warnings sharing among different departments, allowing department administrators to configure filtering options and to focus only on real and most critical threats.

A high-level representation of the proposed architecture is given in Figure 3.1.

The scenario depicted in Figure 3.1 represents a network consisting of n departments, each of them hosting several nodes divided in different subnetworks. As mentioned before, the complete coverage of this kind of infrastructure requires the deployment of multiple NIDS, once for each subnetwork. All the NIDSs belonging to the same department are part of an intra-department hierarchical architecture, whose structure reflects the topology of the network of the monitored de-

partment and which is described in Section 3.4.1. All the alerts generated within a department are collected, evaluated and relayed to the root of this hierarchy, namely *Root Department Manager*, that provides a comprehensive network security report to the department administrator. As shown by the dotted lines in Figure 3.1, each *Root Department Manager* is connected to the others through a decentralized peer-to-peer overlay network. This communication layer is the base of the inter-department alert sharing service described in Section 3.4.2, which allows the department administrators to receive early warnings related to the most critical threats.

Both the intra-department hierarchical architecture and the inter-department alert sharing service rely on the alert ranking mechanism presented in Section 3.3.

3.3 NIDS alert ranking

The proposed architecture integrates a system which provides ranking and prioritization of security alerts generated by distributed sensor, and is based on the correlation of heterogeneous information coming from:

- network security alerts generated by NIDSs;
- software configuration of the machines deployed in the protected network;
- vulnerability information gathered from external, open and unstructured sources.

The main idea is to combine all the available information related to the detection of malicious activities, in order to distinguish real threats from false positives alerts. Thanks to the integration of this knowledge, when a NIDS raises an alert, the system can check whether a machine contains one or more software packages that are vulnerable to the detected attack. If a match exists, then there is a high probability that the targeted machine has been compromised. Thus, the raised alert needs to be investigated by the department administrator as soon as possible. On the contrary, if no match exists, then the attack failed, because the targeted machine was not vulnerable to the detected threat. The related alert can be ranked as non-critical, thus letting the department administrator to focus on critical ones and postponing possibly useless analysis.

Figure 3.2 illustrates the design of the alert ranking system, which is mainly built of four components: the *External sources crawler*, the *vulnerable software database*, the *Configuration Management DataBase* (CMDB) and the *alert ranking server*.

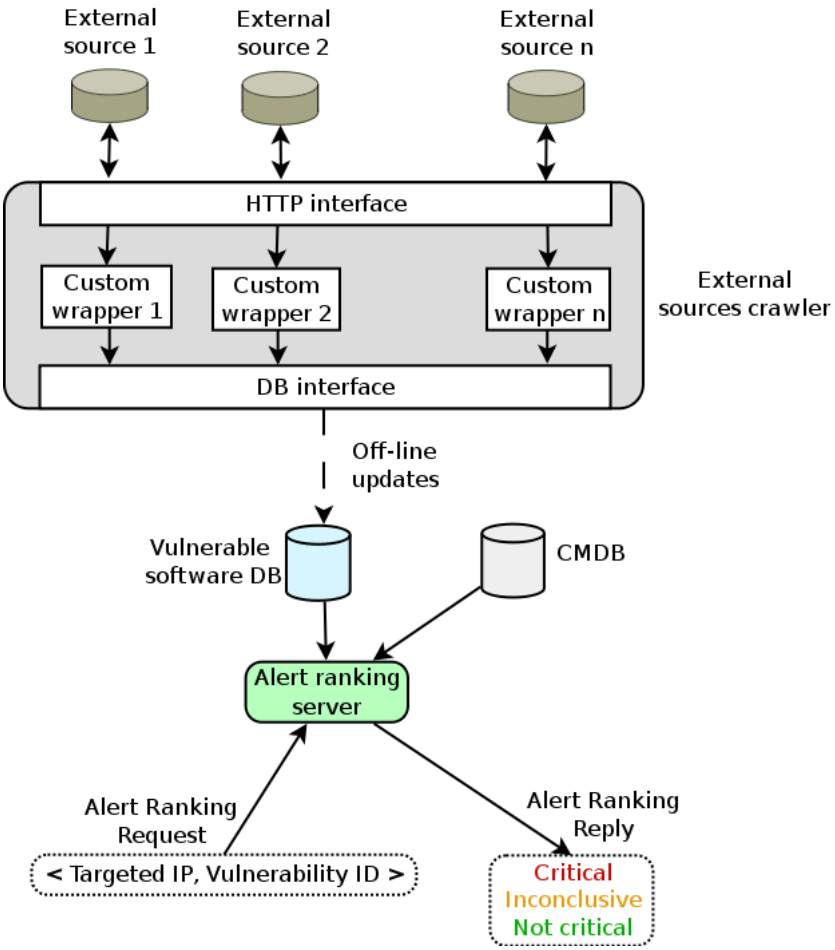

Figure 3.2: Architecture of the alert ranking system

The *External sources crawler* is a custom software able to gather vulnerability information from several heterogeneous and unstructured open sources, that can be freely consulted on the Web[1]. Each source requires an ad-hoc wrapper specifically designed to parse all the vulnerability advisories released and to extract the list of vulnerable software. Despite each source adopts a different format, the large majority of vulnerability advisories are cross referenced, then it is possible to identify the advisories published by different sources, but related to the same

[1]notable examples are: Open Source Vulnerability Database (http://osvdb.org/), Common vulnerabilities and exposures (http:/cve.mitre.org/), National Vulnerability Database (http://nvd.nist.gov/nvd.cfm), Packet Storm security advisories (http://http://packetstormsecurity.org/files/tags/advisory/), SecurityFocus™vulnerabilities (http://www.securityfocus.com/vulnerabilities/)

vulnerability. The complete parsing of all the vulnerabilities represents a heavy workload for the *External sources crawler* due to several factors such as the computational complexity of the operations that the wrapper has to perform and the number of vulnerabilities produced by the source that need to be parsed. However, once a complete scan of an open source has been performed for the first time, only the advisories that have been recently introduced or updated need to be processed and this operation can be performed off-line at regular time intervals, thus without affecting the performance of the ranking process.

The *vulnerable software database* receives the parsed data by the *External sources crawler* at regular time intervals, then it stores associations between known vulnerabilities and the list of vulnerable software. In particular, it offers an easy interface which allows to query for specific vulnerabilities, and replies with the list of the related vulnerable software.

The *Configuration Management Database* (CMDB) stores the complete information of all the components deployed within the departments, the software installed on each host and the services belonging to the IT infrastructure. The stored information needs to be as accurate as possible in order to facilitate the processing step performed by the *alert ranking server*. However, as detailed in Section 3.4.1, the proposed architecture does not require a single CMDB containing configuration information for all the machines connected to the network. Indeed, the hierarchical architecture for intra-department communication facilitates the partitioning of the configuration information in several local databases responsible for single network segments. Thus, it is possible to delegate the CMDBs maintenance to the department administrators.

The *alert ranking server* is the core of the alert ranking system proposed in this chapter. Its main purpose is to rank alerts generated by local NIDS combining the information coming from the CMDB, the vulnerable software database and the NIDS alerts themselves. The *alert ranking server* receives requests containing the information related to the alert produced by the NIDS, including some identifier of the vulnerability associated to the detected attack, and the IP address of the targeted host. While the latter value can be directly used, the former may require further analysis in order to obtain a useful vulnerability identifier. In particular, signature-based NIDSs include in the alert a reference to the description of the attack, which in turn refers to at least one of the open sources monitored by the *External sources crawler*. Once the *alert ranking server* has obtained the vulnerability ID, it performs a query to the *vulnerable software database* using the ID as a key, to retrieve the list of vulnerable software. Concurrently, the IP address is used in a second query executed against the CMDB in order to retrieve the list of software deployed on the targeted machine. Then, the *alert ranking server* compare the two lists of software received as results of the executed queries. If a match exists, then there is a high probability that the attacked host has been com-

promised. Thus, the alert is ranked as *Critical* by the alert ranking server. On the other hand, if no match exists, the alert is ranked as *Not critical*, because none of the softwares installed on the targeted machine was vulnerable to the vulnerability exploited by the attack, then the latter does not represent a real threat. Finally, if the *alert ranking server* is not able to perform the analysis, due to lack of data in one of the two databases, or due to a system or network failure, it ranks the alert as *Inconclusive*. In this case, it is up to the department administrator to further investigate the problem.

While the alert ranking process involves several components and require different processing steps, the computational cost related to the comparison among the two software lists (which usually contain only a few elements each) is very low, and comparable, if not lower, to the computational complexity of the in-depth packet analysis performed by a signature-based NIDS, in which the payload of a packet has to be matched against hundreds of rules. Then, it is possible to perform alert ranking at run-time processing alerts generated by NIDSs.

The integration of *alert ranking server* described in this section into the proposed architecture is detailed in the next Section.

3.4 Cooperation among distributed NIDSs

In this section we describe the two cooperation schemes at the base of the hybrid architecture proposed in this chapter.
At intra-department level we propose a hierarchical architecture to enable cooperation among distributed NIDSs monitoring different network segments, while to enable critical alerts sharing at inter-department level, we propose a peer-to-peer architecture.

3.4.1 Intra-department hierarchical architecture

As mentioned before, the complete coverage of a large network requires the deployment of several distributed NIDS monitoring different network segments. In order to manage all the generated alerts, it is possible to interconnect these sensors through a multi-tiered hierarchical architecture, whose structure reflects the topology of the network of the monitored department. We propose a hierarchical architecture which improves cooperation by combining distributed NIDS and the *alert ranking system* described in the previous section. An example of the intra-department architecture is depicted in Figure 3.3.

The distributed NIDSs, which monitor different network segments, represent the base of the hierarchical architecture. Each NIDS sends the produced alerts to a local alert manager. This component is in charge of collecting and processing all

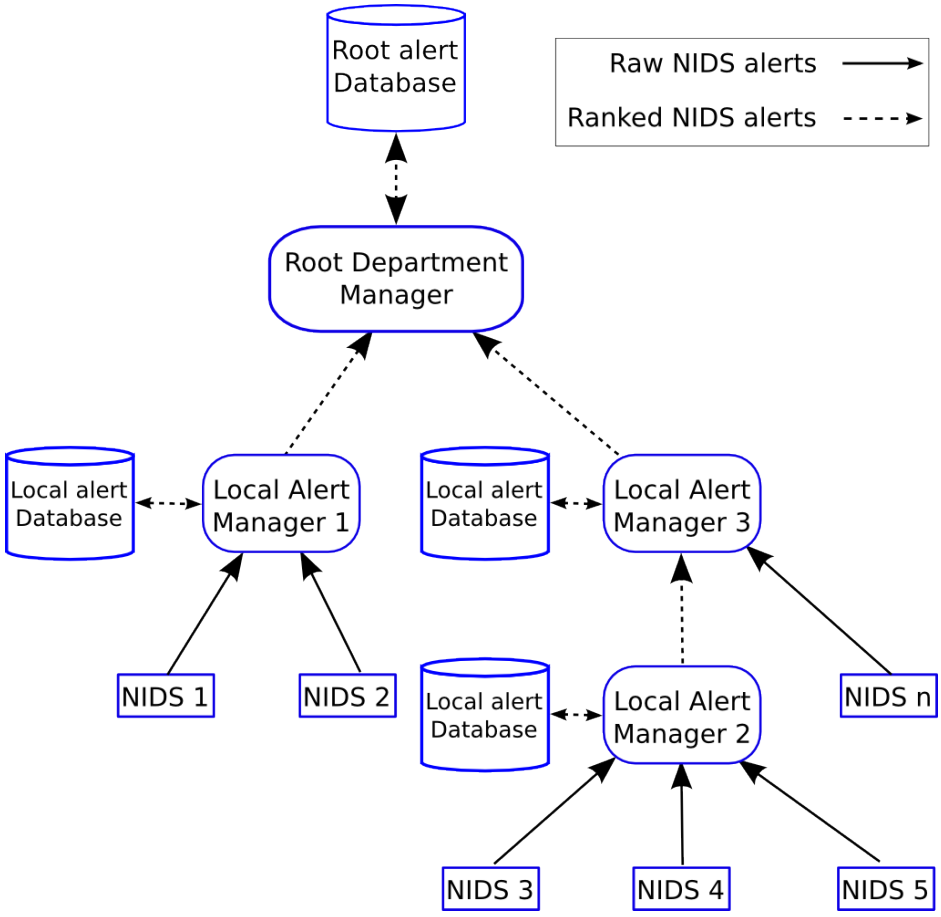

Figure 3.3: Hierarchical architecture for intra-department network monitoring

the alerts received, and then to forward alerts ranked using the ranking process detailed in Section 3.3 to the local alert manager at the upper tier of the hierarchical architecture.

In order to perform alert ranking, each local alert manager integrates an alert ranking server. This server has to access the software vulnerability information database and a CMDB including the software configuration of all the machines that are monitored by the NIDSs to which the local alert manager is directly connected. As an example, the *local alert manager 2* in Figure 3.3 has to be able to access a CMDB containing the software configuration of all the machines monitored by *NIDS 3*, *NIDS 4* and *NIDS 5*, while the *local alert manager 3* only needs software configuration information for the machines monitored by *NIDS n*. Hence there is no need of one centralized CMDB containing the complete configuration of all the machines belonging to the department network. By relying on several local alert managers it is possible to share the load related to alert ranking among

multiple distributed components. Each local alert manager has to rank only the alerts that are generated by directly connected NIDS, while alerts received by lower-level local alert managers have already been ranked, and have to be relayed to the upper tiers of the hierarchy without further analysis. Each local alert manager also stores the ranked alerts in a *local alert database* that can be accessed by the administrator of the corresponding network to monitor the activities carried out within his domain. This chance eliminates the need to ask for this information to the department administrator, or to access department-wide security information that may be classified. Moreover, the deployment of a hierarchy of local alert managers allows to limit the propagation of configuration updates related to changes in topology or in the software configuration of the monitored components. Thus, it is possible for the administrators of large departments to delegate the maintenance of local alert managers, and to shape the aggregation hierarchy on the base of their needs, thanks to the flexibility of the architecture which does not require neither to be balanced nor to be configured as n-ary tree. On the top of the hierarchical architecture, we place the *root department manager* which differently to the other alert managers, is only connected to lower-layer local alert managers, and has no direct connections with any NIDS. Each root department manager stores the received alerts, that have already been ranked, in the corresponding *root alert database*, and integrates a graphical dashboard, to provide a graphical representation of all the alerts that have been generated within the department network. This solution allows the department administrator to easily detect most critical alerts and let him focus on them while delaying the investigation of low ranked alerts.

3.4.2 Inter-department alert sharing service

Cooperation among defensive solutions is the only approach which allows to effectively thwart modern threats. In particular, in a complex IT infrastructure, information sharing among distributed departments is worthy especially in two cases:

- successful attacks targeting machines that provide services to several departments;
- attacks that could compromise the integrity of critical systems.

As an example of the first scenario, let us consider an organization which manages access control through *Single Sign On* (SSO). This system uses centralized authentication servers to provide authentication services to all the applications and components of the organization, in order to allow users to actively enter their credentials only once, which is during the access to the first service. While the machine hosting SSO services is deployed in a single department, a successful

attack would represent a security breach for all the departments that rely on the SSO to authenticate users. To thwart this threat, an alert sharing system represents a precious source of early warnings which allow department administrators to deploy the necessary countermeasures.

In the second scenario, an attack is detected in a department where none of the hosted machines is vulnerable, while a critical vulnerable machine, such as a production server, is deployed within a different department. Without inter-department cooperation, the administrator of the department hosting the critical vulnerable machine would be unaware of the threat, thus he wouldn't deploy any specific countermeasure. On the other hand, by receiving the alert, the department administrator responsible of the critical vulnerable machine could carefully investigate the issue and decide how to protect the machine from that attack.

In order to allow peer departments to share critical alerts and relevant security information, all the root department managers are interconnected through a inter-department peer-to-peer overlay network. This higher level network allows root department managers to share alerts related to network intrusion that could jeopardize the integrity of critical systems within a department, as well as systems that provide critical services to several departments (such as the Single Sign On as described in the previous example). This solution enables departments which detect critical attacks to disseminate early warnings to the other departments, allowing the respective administrators to deploy proactive countermeasures, thus reducing the risk for the systems to be compromised.

In addition to the peer-to-peer overlay network, it is necessary to improve the capabilities of the root department managers. In fact, as detailed in Section 3.4.1, the root department managers receive already ranked alerts forwarded by the local alert managers within the monitored department. However, these ranks are only relevant within the department, since they have been computed using the information of the local CMDB, in particular against the software configuration of the targeted machines. To overcome this limitation, all the root department managers integrate an alert ranking server and have access to the *Critical CMDB* (CCMDB), which is a CMDB that includes the software configuration of a small subset of critical machines deployed throughout the organization's IT infrastructure. All the information inserted into the CCMDB is provided on a voluntary base by the department administrators, that decide which machines are critical and which can be safely added to the CCMDB without disclosure of sensitive information. When a root department manager receives an alert, this alert is processed using the information included in the CCMDB. If at least one critical machine is vulnerable to a detected attack, then the alert is ranked as critical and the root department manager broadcasts the related NIDS alert to all the other root department managers through the peer-to-peer overlay. Each root department manager can add it to its root alert database, possibly after a further validation of the ranked alert,

thus making it available to the administrator through the graphical dashboard.

3.5 Prototype implementation

We implemented a prototype completely based on open source software and custom modules which demonstrates the viability of the proposed architecture. As NIDS sensor we chose Snort [100], the most widely deployed open source IDS, which provides signature-based analysis and an exhaustive description for each signature, including references to open security advisory sources. The validation of the prototype has been performed in a controlled environment by injecting malicious network traffic and by analyzing the behavior of each component of the architecture.

3.5.1 Integration of external sources

Each signature used by Snort comes with a unique identifier (called *Snort Identifier*, or SID) which characterizes all the alerts generated by that signature. The SID can be used as research key to retrieve the rule document containing further information related to the signature. Most of the rule documents provide references to security advisory or to vulnerability description repositories, which in turn contain the list of vulnerable software. Thus, these lists can be extrapolated by custom modules able to parse vulnerability reports. In particular, we focused on CVE [43] and Bugtraq [94], which are freely available and represent two of the most referred source by the Snort rules.

Once selected the external sources, we implemented an *external sources crawler* prototype in Python, mainly composed by three wrappers. The first wrapper is designed to retrieve the rule documents related to the SIDs of all Snort's signatures, by executing HTTP requests to the URL `http://www.snort.org/search/sid/X`, where X represents the SID associated to a specific signature. Then, a parser extracts from the downloaded Web page the links which refer to CVE or Bugtraq websites. If those links are found, the related vulnerability advisories are downloaded and sent to the CVE and Bugtraq wrappers respectively. These wrappers analyze the downloaded Web page, and extract the vulnerable software list, distinguishing, if possible, the producer, the name, and the vulnerable versions of the software. The wrappers normalize the information by converting all the letters to lowercase, and by replacing white spaces with underscores, then they submit the processed data to the vulnerable software database, in the form of a SID and the respective vulnerable software list. The python module in charge of inserting the received data into the database checks whether an old list already exists or not. In the former case, it merges the old and new lists of vulner-

able software and then updates the database entry, otherwise it directly inserts the received data.

3.5.2 Alert ranking server

The alert ranking server has been implemented in Python. It receives as input an IP address, which is the IP of the targeted machine, and a SID, which is the one related to the raised alert. While the former is used to contact the CMDB looking for the software configuration of the attacked machine, the SID is used to query the vulnerable software database. If both the queries are successful, the two lists are compared with the goal of looking for at least one common element. If a software exists in both lists, then a reply is generated in which the alert is ranked as *Critical*, otherwise, the alert is ranked as *Not critical*. Finally, if at least one of the two queries fails (i.e. does not return a non-null list of software), a reply is immediately generated, marking the alert as *Inconclusive*.

3.5.3 Local alert manager

The local alert managers are based on the Prelude [84] hybrid IDS, which is able to manage alerts generated by several IDS, including Snort. The received alerts are stored in a local alert database and can be forwarded to other instances of Prelude, thus facilitating the implementation of a hierarchical architecture. In order to provide ranking functions to the local alert managers, we modified the Prelude correlator module, enabling it to submit all the alerts received from Snort to the alert ranking server. Then the ranked alert is stored as IDMEF [39] message, which is the standard used by prelude to manage different kind of alerts.

3.5.4 Root department manager

Similarly to the Local Alert Managers, the Root Department Managers are implemented through a modified version of Prelude and integrate a local alert ranking server. They receive already ranked alerts by lower-level local alert managers and repeat the ranking process against the CCMDB, which contains the software configuration of the few machines that have been considered critical for the whole organization of the different department administrators. All the alerts ranked as critical by a root department manager are formatted as IDMEF messages and disseminated to the other departments through a peer-to-peer overlay network. In order to enable inter-department cooperation, we implemented a custom communication module based on the freepastry [51] libraries, which provide the peer-to-peer overlay and also a publish/subscribe component, namely Scribe [91]. Root department managers are also provided with a custom version of the Prewikka [5]

graphical user interface, modified to sort NIDS alerts by their rank and to clearly separate the alerts received from one of the local alert managers to the alerts issued by other root department managers.

3.5.5 System validation

The validation of the implemented prototype has been performed in a controlled environment. Our experimental testbed comprises 30 machines equipped with one Intel Xeon 2.4 GHz CPU, 1 GByte RAM, a 32 bit and 33 MHz PCI bus and a Gbit Ethernet NIC. We emulate a network configuration consisting of three departments, each composed by four network segments. To test the ability to issue early warnings, we simulated three heterogeneous network departments: one populated with Windows-based servers, one with Linux-based servers, and the last one with a mixed Windows and Linux environment. Three machines are configured as root department managers. Each root department manager is the head of a hierarchy of three local alert managers. Moreover, four NIDSs are installed within each department. We deploy the external sources crawler and the vulnerable software database on the same machine. Another machine hosts three CMDBs (one for each department) and one CCMDB. The remaining 4 machines of our testbed are used to generate the network traffic analyzed by the NIDSs. We use *tcpreplay* [8] to replay the publicly available IDEVAL [66] traffic traces. On top of this background traffic, we added network packets crafted to match specific rules of the Snort signature database. We carried out several experiments for different workload scenarios. Our prototype has always been able to highlight the most dangerous attacks by flagging the related alerts as critical. The graphical user interfaces of the three root department managers denoted *critical* alerts with red flags, *inconclusive* alerts with yellow flags, and *not critical* alerts with green flags.

3.6 Concluding remarks

In this chapter we proposed a hybrid distributed architecture suitable for large organizations, characterized by complex IT infrastructures and multiple departments with several network segments. The proposed architecture is characterized by three novel features:

- An alert ranking system, which selectively highlights the most relevant security alert thanks to an innovative alert ranking scheme that is distributed among several components of the proposed architecture;

- A hybrid architecture, which combines hierarchical and peer-to-peer communication schemes, thus being suitable to large and complex networks composed by several independent departments, without requiring a centralized authority;
- A selective alert sharing service, which is able to disseminate early warnings to all department administrators, informing about security threats before that vulnerable and critical machines hosted within their departments are targeted by network attacks.

Finally, the feasibility of the proposed architecture has been verified through the implementation of a prototype based on open source software.

Chapter 4

Collaborative attack detection using Distributed Hash Table

4.1 Introduction

In this chapter we introduce a new architecture for distributed and collaborative event processing suitable for the cooperation among different organizations [49]. The proposed architecture is characterized by the combination of a communication overlay based on a Distributed Hash Table (DHT) and the usage of distributed cooperative Complex Event Processing (CEP) engines to analyze high volume of events. In particular, all the CEP engines are connected to a completely decentralized peer-to-peer overlay network, based on a DHT, which allows the CEP engines to forward received events, to share intermediate results for further analysis, and to disseminate final results. The communication overlay is agnostic to both the CEP engine technology and the events processing logic. Thus, the proposed architecture can be used with different CEP engines, ranging from custom ones to well known CEP engines such as Esper [48], and with different cooperation algorithms used by the CEP engines. Moreover, the usage of DHT based overlay provides the whole architecture with a high scalability, fault tolerance, and load balancing capabilities [72]. A prototype of the architecture presented in this chapter has been implemented and subsequently validated using a distributed attack detection algorithm devoted to the early detection of Man-in-the-Middle attacks [71]. An overview of the proposed architecture is presented in Section 4.2, while the implementation details of the prototype, based on open source software, are described in Section 4.3. The attack-specific event processing logic used by CEP engines is detailed in Section 4.4.

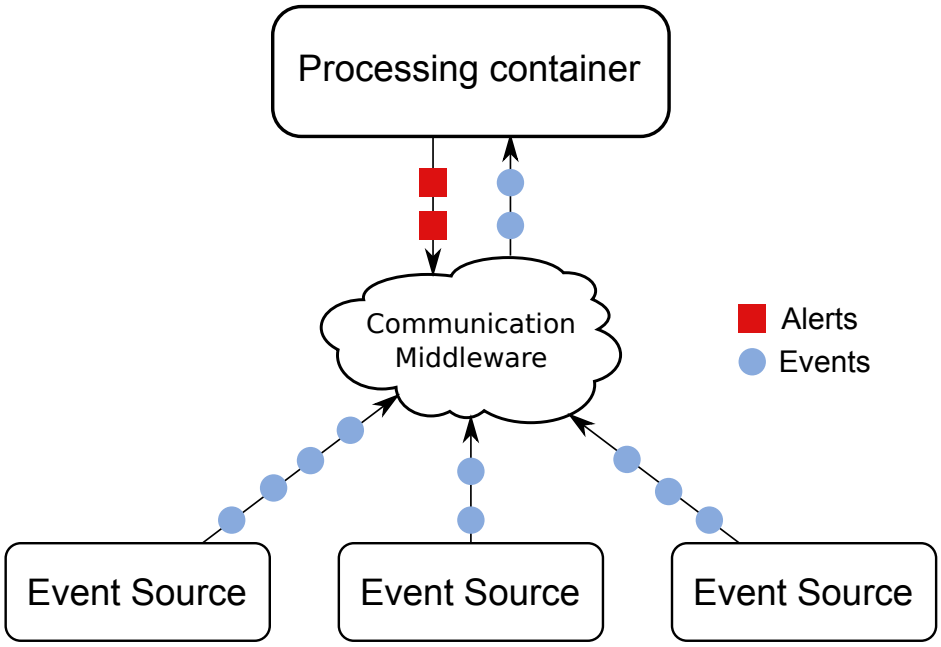

Figure 4.1: High level representation of the distributed collaborative architecture

4.2 Architecture design

4.2.1 Overview of the proposed architecture

A high-level overview of the architecture proposed in this chapter is given in Figure 4.1.

The Event Sources are at the base of the architecture. They represent the gateway through which each cooperating organization constantly submits the events generated within its networks. In order to enable subsequent analysis, each event is normalized following the rules established among the participants of cooperating events processing. In particular, each event is characterized by type, meaning that each event has a specific type, and a set of fields, consisting of pairs: field-name and field-value, which are the same for all the events of the same type. The events generated by the event sources are forwarded to the Processing Container through a Communication Middleware, which can be used also to disseminate processing results obtained by the Processing Container. The latter, which in figure 4.1 is depicted on the top of the architecture, represents the logical block that implements a given event processing algorithm. It receives all the events which can be used by the implemented CEP algorithm and publishes the analysis results through the same Communication Middleware. The processing container described in this chapter has been designed to integrate several distributed pro-

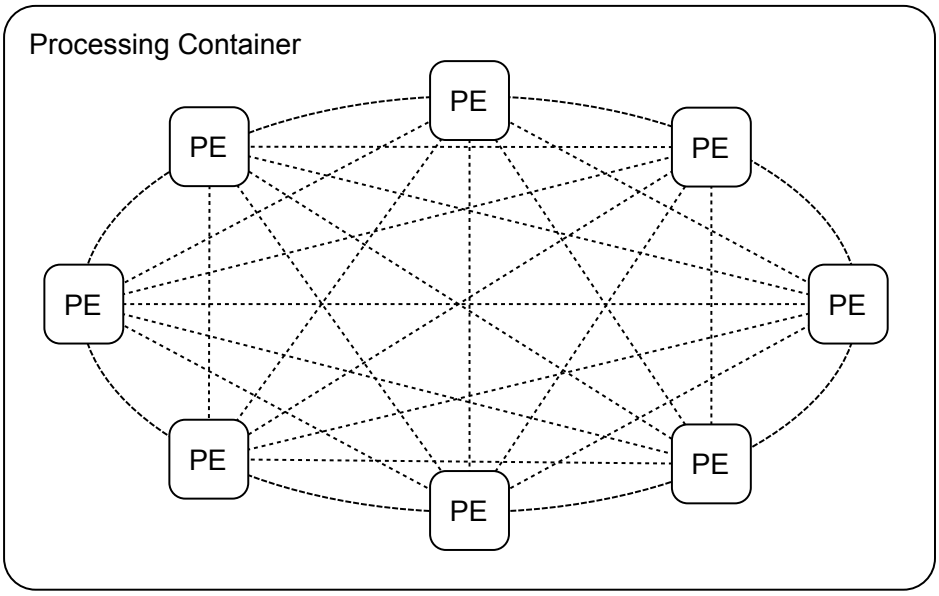

Figure 4.2: DHT overlay of the Processing Container

cessing elements connected by a DHT overlay network, in order to guarantee high scalability and dependability [49]. On the top of this peer-to-peer architecture we implemented an event processing logic, described in Section 4.2.6, which is used by CEP to detect Man-in-the-Middle attacks [71]. An overview is provided in Section 4.4.

The internal architecture of a Processing Container is represented in Figure 4.2, where several Processing Elements (PE) join a fully connected network, typical of peer-to-peer communication schemes. This solution provides dynamic management of the available resources and lacks single points of failures. All the PE are characterized by the same functionalities, in particular:

- they process events received through the Communication Middleware shown in Figure 4.4;
- they distribute events among all the active Processing Elements over the DHT overlay network;
- they analyze both events and partial results received from the DHT overlay network;
- they produce results emerged from the CEP analysis in the form of alert published to the Communication Middleware.

As depicted in Figure 4.5 each Processing Element is made up of three main components: the Event Manager, the Overlay Manager and the CEP Engine, described in Sections 4.2.4, 4.2.5 and 4.2.6, respectively.

4.2.2 Event Sources

In the proposed architecture, the event sources represent an abstraction of the main point of contact between the Processing Container and the IT infrastructures of the cooperating organizations. They actually are gateways which allow events generated within individual organization networks to reach the processing container. In order to reduce ambiguity between the Event Source, a specific element of the proposed architecture, and the sources of events deployed within each IT infrastructure, in this section we refer to the former as Gateway. The main capabilities provided by a Gateway are:

- collection of raw data from heterogeneous sources;
- preprocessing of received data;
- submission of preprocessed events to the appropriate Processing Container.

The logical architecture of a Gateway is represented in Figure 4.3 and includes four main components:

1. one or more SourcePlugins, which collect and normalize heterogeneous events received by different local event sources;
2. a Wrapper, which manages the SourcePlugins and collects normalized events generated by them;
3. a set of Preprocessors, that receive normalized events from the Wrapper and preprocess those events in compliance with the requirements of the Processing Container which will analyzes the events.
4. the communication module, that receives preprocessed events ready to be submitted to the Processing Container and sends them through the Communication Middleware.

The features characterizing the SourcePlugin and the preprocessors have to meet the Processing Container requirements. For this reason, these and the other components have been designed adopting a modular architecture, in particular plugin-based, that allows the administrator in charge of managing the gateway to easily adapt and extend the components on the base of the selected Processing Container. Moreover, thanks to this approach, it is also possible to integrate an

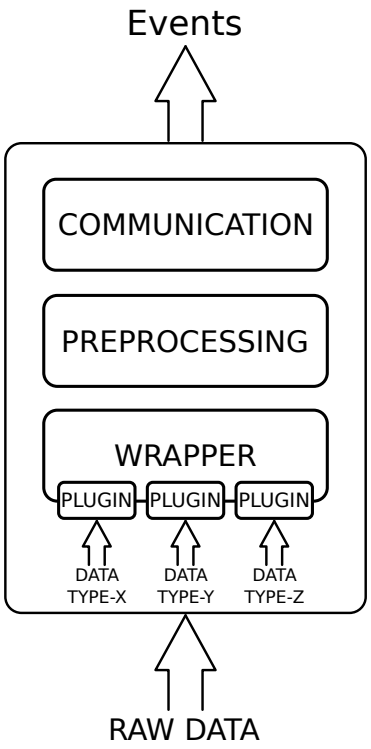

Figure 4.3: High level representation of the Gateway architecture

additional component with the communication module in order to receive all the relevant alerts generated by the Processing Container. In the following sections we provide a brief overview of each Gateway component.

SourcePlugins and event normalization

The SourcePlugins are a set of software modules which represent the interfaces through which the Gateway receive alerts from different and heterogeneous sources. In particular, for each source of alerts a specific SourcePlugin needs to be designed and implemented in order to access and manage the new data. Due to the heterogeneity of the sources, the events gathering approaches may vary, ranging from listening servers waiting data to active clients querying remote databases or analyzing local log files. The aim of these components is to extract the relevant information from the collected data and to produce normalized events representation, compatibles with the preprocessor engines of the following steps. In brief, the four main tasks of SourcePlugin are:

- implementation of specific event gathering strategy for event collection (passive listening vs active polling);

- extraction of useful information on the base of exact syntax and semantic of collected events;
- normalization of each alert on the base of preprocessor requirements;
- forwarding of normalized alerts to the Wrapper component.

Wrapper

The Wrapper component manages the set of deployed SourcePlugins and collects normalized events they produce in order to make them available for the preprocessing step. In particular, due to the requirements fixed by the Processing Container, the Preprocessor component may have to preprocess different event types generated by different SourcePlugins. In this scenario, the Wrapper component acts both as collector of events generated by the deployed SourcePlugins and as dispatcher of those events to the Preprocessor components, on the base of the event types they require. Then, we selected a Message-Oriented Middleware (MOM) approach. This solution supports sending and receiving of messages between distributed systems, including asynchronous communications, thus allowing SourcePlugins, both passive and active ones, to be distributed within the network infrastructure and, possibly, over heterogeneous platforms. During the registration phase, the SourcePlugin informs the Wrapper about the event types it produces and to each event type a label is assigned. Then, when a Preprocessor requests particular data types it has only to specify the label to the Wrapper, in order to receive the desired events.

Event Preprocessing

The IT infrastructure of a single organization may generate a huge amount of events, especially in case of large networks and several local event sources. In order to avoid dissemination of sensitive information and to reduce the volume of data transmitted to the Processing Container, the Gateway implements an Event Preprocessing component which offers filtering, anonymization and aggregation algorithms. Indeed, the administrator in charge of managing the Gateway may configure filtering (or anonymization) rules which allow do delete (anonymize) specific fields from generated events, or possibly to discard whole alerts which match specific values, in order to meet custom privacy requirements. On the other hand, bandwidth consumption improvements can be reached by configuring aggregation rules, which allow to group several events with specific values in common, or on a regular time interval, thus reducing the overhead. However, event preprocessing logic strictly depend on the specific requirements of the PC and

on the related data that have to be preprocessed. Then, each PC requires an ad-hoc preprocessor configuration, or possibly a specific preprocessor engine, that gathers normalized events collected by the Wrapper and preprocesses them obtaining well-formed events which respect the data format specified by PC. Each Preprocessor follows a sequence of preprocessing steps, logically organized in a preprocessing stack. Each preprocessing step receives the events from the underlying step, performs a specific preprocessing activity and, according to its internal logic, forward the preprocessed events to the upper step (e.g., a preprocessing step that performs event filtering activities forwards to the upper step only a subset of the events received from the preceding step). The last step of the preprocessing stack consists on forwarding messages to the communication component that allows the Gateway to submit preprocessed events to the PC for processing, through the communication middleware.

Communication Module

The communication module is the interface used by the Gateway to submit preprocessed events to the Processing Container. This component receives preprocessed events from one or more Preprocessors, depending by the selected PCs, and transmits received events through the Communication Middleware depicted in Fig 4.4. On the base of the latter element, which is chosen by the cooperating organizations, the communication module has to integrate the functionalities specific to the implemented middleware, which are based on publish/subscribe schemes and on Message Queue schemes. More details can be found in Secion 4.2.3.

4.2.3 Communication Middleware

The Communication Middleware is at the base of all the communications between the Event Sources and the Processing Containers. Considering the different characteristics of the communications depending on the transmitted content (i.e. events generated by Event Sources or alerts generated by Processing Elements), we propose a Message Oriented Middleware providing two different communication schemes: the first one based on a Message Queue paradigm and the second one based on a publish/subscribe pattern. We propose the former approach to forward preprocessed events generated by Event Sources. In particular, each Processing Container is associated to a queue hosted by the Communication Middleware and the Communication Modules of the Event Sources forward each event to the specific queue, or possibly to multiple queues in case the same events are used by different Processing Containers. On the other hand, the publish/subscribe scheme is used to publish alerts generated by the Processing Containers. In particular each Processing Elements belonging to them may publish processing results on the spe-

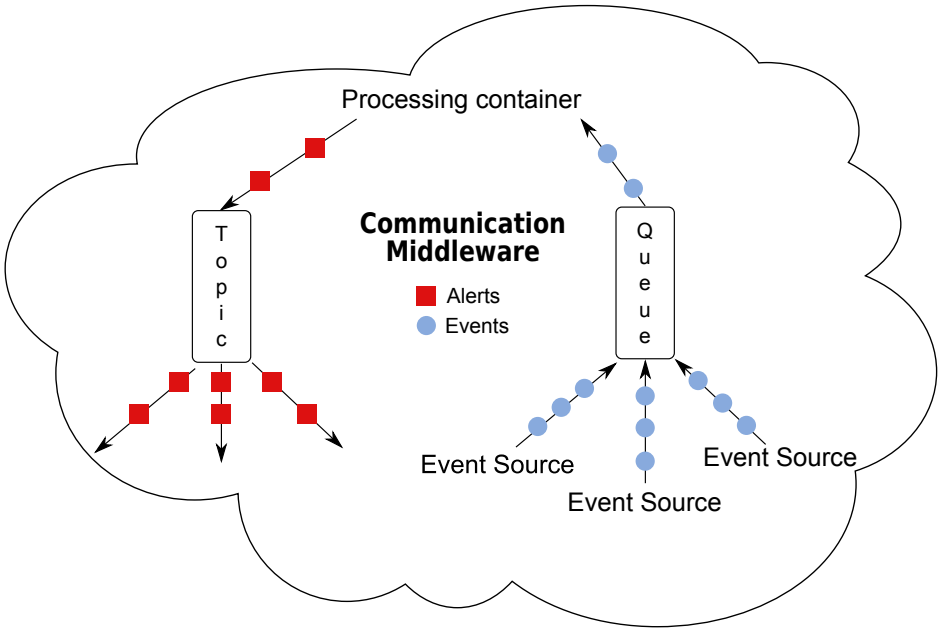

Figure 4.4: Architecture of the communication middleware

cific topic, allowing all the interested parties to subscribe only to interesting ones. Possibly, different Processing Containers may publish results on the same topic, as well as one processing container may disseminate alerts on different topics.

4.2.4 Event Manager

The Event Manager is the component in charge to exchange information with the Communication Middleware, thus it represents an interface used by the Processing Element to receive events generated by the Event Sources and to disseminate results obtained by CEP engines.

Each Processing Container receives events through a queue hosted by the Communication Middleware. In particular, all the Processing Elements have to register themselves to the queue as consumers, then they retrieve messages following a round-robin approach. This allows to share the load related to event gathering among all the Processing Elements belonging to the Processing Container. When an Event Source publishes an event, the communication middleware appends the event to the queue related to the specified Processing Container and maintains the event in the queue until it is retrieved. Once the message is read by the Event Manager of a Processing Element, it is removed from the queue (i.e. it is received by only one PE). Then, the Event Manager forwards the message to the local Overlay Manager, which is in charge of distributing the events within the

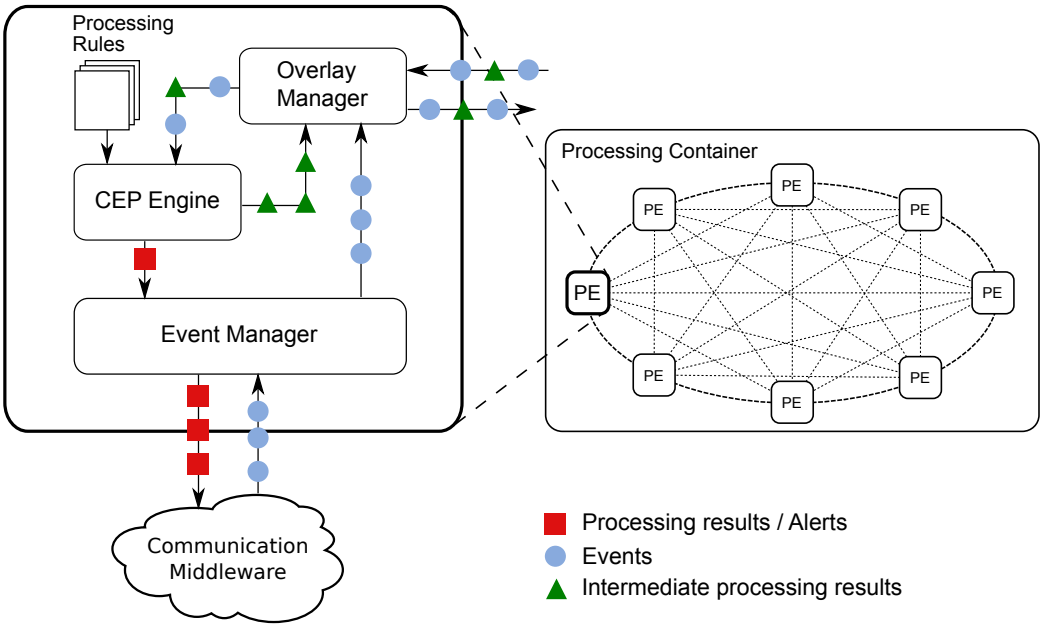

Figure 4.5: Architecture of a Processing Element (PE)

Processing Container. The Event Manager has also to disseminate alerts generated by the local CEP Engine, thus making them available to all the interested parties. In order to reach this target, the Event Manager publishes events using a publish/subscribe scheme. In particular, it registers itself to a topic hosted by the Communication Middleware, which is used by all the Processing Elements of the same Processing Container, and possibly by different Processing Containers. Then, the Communication Middleware makes the alert available to all those that subscribed to this topic (i.e. the Event Sources through their Communication Modules).

4.2.5 Overlay Manager

The Overlay Manager is the component through which all the Processing Elements exchange data within a Processing Container. In particular, within each Processing Element, it is directly connected to the CEP engine, to exchange events and alerts, and to Event Manager, only to receive events. Each Overlay Manager represents a node of a peer-to-peer network, which offers a fully connected mesh overlay and is based on a Distributed Hash Table. This DHT overlay is used to equally distribute all the messages sent by each Overlay Manager, thanks to its intrinsic characteristics.

The main tasks of the Overlay Manager are:

- to receive events retrieved by the local Event Manager;

- to forward partial results, and possibly alerts, generated by the local CEP engine to other Processing Elements for further analysis;
- to receive partial results, and possibly alerts, generated by other Processing Elements connected to the same DHT overlay;
- to provide to the local CEP engine an event stream, which is a portion of the data received from the Event Manager or from other Processing Elements.

When the Overlay Manager joins the DHT overlay, it receives a unique identifier, namely a nodeID, which is used to route all the messages exchanged within the DHT. In particular, each member of the overlay is responsible of a portion of the DHT domain and the nodeIDs and messageID, an identifier assigned to each message, are used to determine which node is responsible of handling each message. When the local Event Manager forwards an event, or the local CEP engine sends a partial result, to the Overlay Manager, it transforms the received data into one or more messages which are distributed through the DHT overlay. The number of messages generated starting from a single event depends on different factors, such as the event type and the event processing logic implemented by the Processing Container. For each received event, the Overlay Manager analyzes its content and selects the fields which are used to build one or more messages, then each message is labeled with a messageID and sent over the DHT overlay. Both the fields selection and the labeling operation depend on the event processing logic.

To better explain these steps, let us consider a very simple example where the aim of the Processing Container is to identify the IP addresses which represent the source of a large number of events. A simple approach to distribute the load of this task is to partition the IP address space among all the Processing Elements belonging to the Processing Container. Then, the CEP engine of each Processing Element has to maintain a counter for the IP addresses that belong to its portion of address space and that have been inserted in at least one event. In this context, each event has only one field of interest, that is the IP address of the machine that triggered the alert by attacking one of the Cooperating Organizations. The architecture proposed in this chapter can easily satisfy both the requirements of this algorithm: all the events related to an IP address have to be received always by the same Processing Element and the whole workload has to be equally shared among the Processing Elements. In order to meet these requirements, for each event received from the Event Manager, the Overlay Manager prepares a message containing the original event and labeled with a messageID consisting of the hash of the IP address. The messageID is used to route the message over the DHT overlay and to decide which Processing Element is responsible of receiving and analyzing the message. Indeed, a portion of the hash space is associated to each

Processing Element on the base of the nodeID assigned to the Overlay Manager, which usually means that each message is routed to the Overlay Manager having the minimum distance between its nodeID and the messageID. When the sender Overlay Manager has to forward an event, it submits the message to the DHT and notifies the recipient Overlay Manager about the new message. This method is useful when a message with the same messageID has already been sent over the DHT. In this case, the sender Overlay Manager has only to notify the recipient that another copy of that message has been issued, thus avoiding to send the same message more than once. Then the recipient Overlay Manager is responsible for extracting the event from the message, and forwarding the content to the local CEP engine for the event analysis.

The proposed cooperation scheme provides an effective and resilient distribution of the event processing workload. In particular, the ability to self-adapt to variations in the number of Processing Element deployed within a Processing Container, the fault tolerance and the load balancing, are all features inherited from the implemented DHT overlay.

4.2.6 CEP Engine

The Complex Event Processing Engine (CEP Engine) is the component in charge of processing all the data received through the Overlay Manager, which can be events generated by Event Sources or partial results obtained by other Processing Elements, and communicating results both to the Overlay Manager and to the Event Manager. The proposed architecture is agnostic to both the CEP engine technology and the events processing logic, thus it is possible to choose different CEP engines for each Processing Container, or possibly for each Processing Element in case of Hybrid Processing Container.

Each Processing Element has to analyze a portion of the events received by the Processing Container from the Event Sources. In particular each CEP engine processes an event stream, which is a series of events characterized by some value in common, and is received from the local Overlay Manager. Analyzing the received events, a CEP engine can generate an alert or a partial result. In the former case, the alert generated is immediately forwarded to the Event Manager, that will disseminate the alert to the external parties through the Communication Middleware, while a copy is passed to the Overlay Manager. Indeed, both alerts and partial results may represent a new information which can be further analyzed by a (possibly different) CEP engine, exactly as events generated by external event sources, depending on the target of the analysis, each Processing Container deploys a specific event processing logic, thus the CEP engines can be configured with different processing rules.

4.3 Prototype implementation

The architecture proposed in this chapter and and described in Section4.2 has been implemented using existing open source software, where possible, and developing new modules from scratch, where necessary. In this section we describe how the main components have been implemented. In particular, the implementation of the internal components of the Event Source (SourcePlugins, Wrapper and Preprocessors) and of the Processing Element (Overlay Manager, Event Manager and CEP engine) is described in Section4.3.1 and Sectionch:dht-cep:prototype:event-manager, ch:dht-cep:prototype:overlay-manager, ch:dht-cep:prototype:cep-engine, respectively.

4.3.1 Implementation of the Event Source

In the following sections we describe the implementation details of the Event Source components. All the modules implemented from scratch are written in Java, in order to obtain multiplatform software components suitable for different IT environments.

SourcePlugins

The SourcePlugins have to retrieve information from heterogeneous sources, normalize the acquired data and forward well formed messages to the Wrapper. We implemented two different SourcePlugins: the first one is able to retrieve alerts generated by an IDS from a remote DBMS, while the second one is able to sniff network traffic and to read traffic traces. The access to the DBMS is performed through a JDBC driver, thus it is independent of the deployed DBMS. The configuration files allow the operator to specify the query used to retrieve alerts and the options used to connect to the DBMS. The SourcePlugins able to sniff traffic leverages libpcap libraries [55] to extract data from network traffic, and on the base of the configuration file selects the useful fields, which vary from simple IP addresses to complete packet's headers. For both the SourcePlugins the normalization process consists of preparing ad-hoc Java classes, recognized by the Preprocessor, filled with the data gathered from the related sources. Also the interaction with the Wrapper is the same independently of the source. Each SourcePlugin has to register itself with the Wrapper, which implements the control logic that allows it to control the execution of the SourcePlugin. In order to allow a distributed deployment of the Gateway's components, the interaction between a SourcePlugin and the Wrapper component occurs through Java RMI [57] method calls and all the events produced by a SourcePlugin are relayed to the Wrapper component through Java Message System (JMS) [58].

Wrapper

As mentioned in the previous section, the Wrapper component acts as a collector for the events generated by the deployed SourcePlugins and, according to the event types they are interested in, as a dispatcher for the Preprocessor components. In order to support this interaction model, we decided to leverage a Message Oriented Middleware (MOM), then we embeded in the Wrapper component implementation a lightweight JMS server, in particular it manages a JBoss HornetQ JMS server instance [54]. The choice of JBoss HornetQ is motivated by its design that consists in a set of simple Plain Old Java Objects (POJOs). Thus, it can be easily and directly embedded, instantiated and run without setting up a heavy application server. When a SourcePlugin registers with the Wrapper, it specifies the event types it produces and each event type corresponds to a JMS Topic. Therefore, when the Wrapper receives a registration request from a SourcePlugin, it creates a JMS topic for each new event type the SourcePlugin will produce (i.e. multiple SourcePlugins can use the same JMS topic). This allows a SourcePlugin to submit produced events by publishing them on the corresponding JMS topic, while Preprocessor subscribes to one or more JMS Topic related to the required event types.

Preprocessors

Depending on the Processing Container which will process the events, the Preprocessor has to read events generated by different SourcePlugins and to perform different tasks in order to prepare well formed messages usable by the Processing Elements. The two main tasks of this component are the communication of the event (raw or preprocessed) and its preprocessing. The first operation is provided by the Preprocessor class which implements the communication logic to read events from the JMS Topic and to relay preprocessed events to the communication component of the Gateway. All the preprocessing operations are performed by smaller modules called PreprocessSteps. All the PreprocessSteps have to be registered with the Preprocessor. Then the Preprocessor links them together in order to build the preprocessing stack structure according to the registration order. Thus meaning that a event received from the Wrapper traverses the all PreprocessSteps in the same order as they have been registered. We implemented four steps: two of them implement two different event filtering techniques, one is an aggregator and the last one is the anonymizer.

- The event filtering step is based on boolean conditions defined on event fields and allows to filter whole events by discarding those which match the rules. The number of events in output is smaller or equal to those in input.

- The event fields filtering step provides a more granular control with respect to the previous step. It analyzes each single field of an event discarding those without useful (or with private) information. The number of events in output is equal to those in input, but with possibly fewer fields per event.
- The Aggregator collects and aggregates events having the same value for one or more fields (batch based approach) or creates batches of messages received in the same configurable time interval (time based approach). This step allows to decrease message size, by reducing the overhead caused by multiple single events.
- The Anonymizer is the preprocessor step in charge of anonymizing fields which contains sensitive data. It implements different approaches, such as hashing or masking and performs its task on the base of the configuration rules.

Communication Module

The Communication Module receives events preprocessed by the Preprocessor and forwards them to the Processing Container through the Communication Middleware. As described in Section4.2.3, the latter component is based on a Message Oriented Middleware approach. Then, the Communication Module has to integrate a client able to publish content on Message Queues related to the desired Processing Container and an optional client which subscribes to topics related to interesting alerts. However, both the clients implementation strictly depends on the implemented Communication Middleware, which is described in the next section.

4.3.2 Implementation of the Communication Middleware

All the communications between Event Sources and Processing Containers are managed by a Message Oriented Middleware, that provides two different communication schemes: Message Queues and publish/subscribe. For this middleware implementation our choice has fallen on Java Message Service (JMS). JMS is a standard component of the Java 2 Enterprise Edition platform and provides a Java Message Oriented Middleware (MOM) API that enable Java application to implement message-based interfaces. In particular, it has been specifically designed for the implementation of reliable, loosely coupled and asynchronous interfaces between different components of distributed applications, and provides both the communication schemes our architecture requires: JMS Queue and JMS Topic. Depending on the type and number of Processing Containers, the JMS hosts an adequate number of JMS Topics, at least one, and JMS Queues, at least one per

Processing Container. All the Event Sources that want to submit events to a Processing Container need to know from which JMS Queue on the JMS the related Processing Elements read messages. Preprocessed events are then submitted by sending well-formed messages on that queue. In order, to process different types of events, the Processing Container can also listen to multiple queues, possibly provided by different JMS servers. When a Processing Element wants to disseminate its processing results it has to publish the alerts on the related JMS Topic, and possibly on more topics depending on the configuration. Then, the JMS sends the alerts to all the interested parties, such as the Event Sources which have subscribed to that topic.

4.3.3 Implementation of the Event Manager

Each Processing Element interacts with the Communication Middleware through the Event Manager, which performs two different tasks: the first one is to forward to the local Overlay Manager the events received from the Event Sources, while the second task consists of publishing, through the Communication Middleware, the alerts generated by the local CEP engine. The Event Manager gathers all the events generated from Event Sources by listening to a JMS queue. In particular, when a Processing Element is started, the Event Manager connects to the JMS server and has to register itself to the queues and the topics specified in the configuration files. All the Processing Elements that belong to the same Processing Container have to listen to the same JMS queue, or possibly more than one, waiting for incoming events. Due to the implementation characteristics of queues in JMS, each event is received by exactly one Processing Element, then, in order to distribute the gathering load among all the Processing Elements, the event retrieval is performed according a fair round-robin policy. Once the Event Manager has read an event from the JMS Queue, it has to analyze the event and possibly preprocess it before forwarding the content to the Overlay Manager. Indeed, as mentioned before, Event Source preprocessors are able to aggregate multiple simple events into a single message, in order to reduce overhead in network communications. In this case, the Event Manager has to unpack the composite event before passing it to the Overlay Manager. When the Event Manager receives processing results from the local CEP engine, it prepares a well formed alert, then it publishes the alert on one or more JMS Topics hosted by the JMS server, depending on the type of alert and on the configuration rules of the Processing Container. This mechanism allows the alerts to be received by all interested parties which have subscribed to the related topics.

4.3.4 Implementation of the Overlay Manager

The Overlay Manager is the component that allows events distribution and more in general information sharing among all the Processing Elements deployed within the same Processing Container. In order to provide a good flexibility, we implemented this component in Java, obtaining a multiplatform software, thus allowing deployment in different environments. As described in Section 4.2.5, the Overlay Manager is a node of a DHT. In particular, we developed the Overlay Manager using two different applications: PAST [44, 90], a large-scale, persistent peer-to-peer storage utility, and Scribe [25, 91], a large-scale and decentralized application-level multicast infrastructure. Both the applications are included in the FreePastry [51] libraries, a Java implementation of the Pastry [24, 92] DHT overlay network. In a distributed environment, network delays or message loss may represent a real problem each time one Processing Element has to interact with another one. In this environment, continuations represent a well known abstraction, which works by reifying the program control state enabling the implementation of non blocking methods. We used this solution to allow the Overlay Manager to invoke network based methods and continue its work without (directly) taking care about delays or message loss. PAST is a distributed persistent storage utility based on peer-to-peer network. It is based on a Pastry layer for routing messages and network management, and offers an automatic replica management of inserted data. Thanks to this solution it is able to balance the storage space among all the nodes and to guarantee high fault tolerance. Each PAST node, identified by an ID of 128 bits (nodeID), acts both as a storage repository and as a client access point. Any file inserted into PAST uses a 160-bit key, namely fileId (messageID in our case). When a new file is inserted into PAST, Pastry directs the file to the nodes whose nodeID is numerically closer to the 128 most significant bits of messageID, each of which stores a copy of the file. The replica factor can be tuned to meet the desired availability and persistence requirements. For example is possible to increase the fault tolerance of the whole cooperating architecture at the cost of a greater number of nodes storing each file. Also Scribe is built on top of Pastry, and leverages Pastry's reliability, self-organization, and locality properties in order to create and manage groups and to build efficient multicast trees for the dissemination of messages to each group. When the Overlay Manager receives an event from the Event Manager, it analyzes the event and then generates a variable number of messages depending on the event processing logic implemented by the Processing Container. The generated messages, which may vary from one per event to one for event's fields, are labeled with a messageID, computed by hashing the relevant fields of the original event, and may contain also less relevant information. Once the message has been prepared, it is inserted by the Overlay Manager into PAST, hence it is stored on the Processing Elements whose nodeID is more

similar to the message ID. This makes the message available to all the nodes of the DHT. In order to guarantee an high dependability, we rely on the built-in content replication feature provided by PAST, which allows us to easily manage replicas of the messages exchanged among the Processing Elements belonging to the Processing Container. Indeed, as mentioned before, PAST automatically generates multiple copies of each stored files, according to a configurable replication factor. Thus, in case of one or more Processing Elements fail (at most k-1, where k is the number of stored replicas) their content is not lost, and their analysis can be completed by neighbor nodes (i.e. the Processing Elements having nodeID numerically closer to the nodeID of the failed Processing Elements). While each message inserted into PAST is stored by multiple Processing Elements, only one of them is responsible of event analysis. The Overlay Manager which inserts the content into PAST receives as feedback a list of the nodes that successfully stored the message, then it selects one of them as responsible on the base of the configuration rules. The only requirements of the election model is that different Overlay Managers inserting the same message, elect as responsible the same Processing Element. In our implementation, the node on the list which is characterized by the highest nodeID is elected. Since file insertion into PAST storage is transparent to the storing nodes, the Overlay Manager has to explicitly notify the responsible Processing Element each time it submits a new message into the DHT. In particular, the submitting Overlay Manager sends a unicast message to the Overlay Manager of the responsible Processing Element. The message is sent through the Scribe overlay, which allows to route messages using the same Pastry overlay used by PAST (i.e. the Processing Elements are identified by the same nodeID in both the applications). If a message with the same messageID has already been sent over the DHT, then PAST already contains a copy of the message. In this case, when the submitting Overlay Manager tries to submit the file, it directly receives the list of the nodes that already store a copy of the file, then it has only to notify the Processing Element identified by the highest that another copy of that message has been issued. This solution helps the sender Overlay Manager to reduce network traffic by submitting only new messages, and avoid Processing Container to waste storage space. Once the responsible Overlay Manager receives the notification from the sending Processing Element, it retrieves the message from its local storage, which is a portion of the PAST storage, using the messageID as key. Then, both the messageID and the message content are forwarded to the local CEP engine for event processing. It is also possible for the Overlay Manager to receive messages from the local CEP engine. Indeed, if the CEP engine generates only a partial results, which means that it requires further analysis by another Processing Element, it forwards the processed data to the Overlay Manager in the form of a well formed message. Then, the Overlay Manager would processes the message as if it has been received from the Event Manager.

4.3.5 Implementation of the CEP engine

The CEP engine is the component of the Processing Element in charge of performing real-time analysis of the event stream received from the local Overlay Manager. The implementation of our CEP engine is based on Esper [48], which is an open source software for complex event processing and event stream processing. It is written in Java and is a widely used CEP engine thanks to a rich Event Processing Language (EPL), used to write processing rules, the support to different event format, such as Java beans and XML documents. In particular, the syntax of processing rules written in EPL is very similar to the syntax of standard SQL queries. Hence, it offers a favorable learning curve for operators already familiar with SQL. Using a CEP engine based on Esper and EPL, the Processing Element is able to perform complex event processing that take advantage of several useful abstractions and functions. In particular, our CEP engines support different processing logic including time-based and row-based batched windows, time based and row-based sliding windows, and transactions composed of multiple low-level events. An example of processing rules and CEP engines configuration, is presented in Section4.4.4 with regards to the detection of Man in the Middle attacks.

4.4 A use case: collaborative detection of Man-in-the-Middle attacks

In this section we present a use case of the architecture proposed in this chapter, which takes advantage of the distributed event processing described in the previous sections. The use case focuses on the Man in the Middle (MitM) attacks, introducing a brief description of the scenario and describing how to configure the Processing Elements to obtain an early detection of the attacks.

4.4.1 Attack description

MitM attacks can be briefly resumed as the act of intercepting communications between a legitimate user of a service and the legitimate server, in order to collect information submitted by users and use it to access the service. The first part of the attack is performed by inducing legitimate users of a service (e.g., a customer of a bank) to interact with a server, rogue server, specifically configured to look like and behave as the legitimate server. Several techniques exist that allow an attacker to hijack communications of a legitimate users, and they vary from exploitation of software vulnerabilities to social engineering, also depending on the targeted services or class of users. Four well known techniques are:

- DNS cache poisoning. This technique targets DNS server used by costumers where the IP address of the legitimate server is substituted with the IP address of the rogue server. When the legitimate user tries to contact the legitimate server, the logical name is resolved with the rogue IP address, then from the user viewpoint there is no difference in the service, and it is not possible to detect the attack. Unfortunately, the recent discovery of several vulnerabilities in many DNS implementations [37, 62], as well as the slow adoption of DNSSEC protocol [17], which is intended to solve several security issues of the present DNS protocol, make this strategy relatively easy to apply.

- Compromise of an intermediate routing node. By compromising an intermediate routing node, the attacker can redirect the interested traffic versus the rogue sever. Home routers are among the preferred targets of the attackers, because they are widely deployed and poorly protected. Indeed, different techniques exists to remotely attack these components (i.e. exploiting Cross Site Scripting [4] or Cross Site Request Forgery [3] of the Web interface). Similarly to the previous technique, also in this case for the attacker is very difficult to detect the attack.

- Phishing. While Phishing is an independent attack strategy, it could be used during MitM attaks to induce users to connect to rogue servers. It consists on sending a huge amount of SPAM messages containing malicious urls, which refer to the rogue servers. In this case, normal users could be saved by antispam filters, while skilled ones can also recognize the attack by detecting the suspicious urls.

- Authentication vulnerabilities. Considering services which require specific software clients to connect to the server, it is possible for the attacker to target this component. In case of vulnerable software clients [47], the attacker can exploit the vulnerability to obtain direct access to the server or redirect users to malicious ones.

Once the attacker receives the client connection, it has to show to the user a copy of the service as similar as possible to the original one, in order to avoid rising of suspects by the user. Then, when the user insert the credentials to log-in to the service, which usually consist of username, password and, possibly, tokens, the rogue server relays the information to the legitimate server and, at the same time, make it available to the attacker. This approach serves a twofold purpose. With respect to traditional phishing strategy, it offers to the user a more realistic interaction, thanks to the replies received from the legitimate server. Moreover, through the rogue server, the attacker is able to analyze all the data exchanged between

the user and the server, and, possibly, to modify the content of the transaction on-the-fly (i.e. by altering sensitive and critical information). While a single instance of a MitM attack is almost impossible to detect, since it is not distinguishable from a normal transaction, large-scale MitM activities that involve multiple customers and/or multiple Financial Institutions, can be detected if the Institutions share some basic information. The main difference between normal transactions and MitM activities, is the relation between IP addresses and customers. For the former, the typical ratio is one-to-one, while for the latter the relation is one-to-many. This reflects the rogue server which start a high number of connections on behalf of several different customers. This anomaly can be easily detected also by single Institutions, but, at the same time, can be also avoided by the attacker by distributing the attack on multiple Institutions. For this reason cooperation among different organization offers more security with respect to individual solutions.

4.4.2 Attack scenario

We simulated a realistic deployment of the architecture proposed in this chapter, that included three Event Sources, the Communication Middleware and a Processing Container. All the components were deployed in a controlled laboratory environment, and the attack scenario was a MitM attack targeting different Financial Institutions. The three Event Sources represent three different Financial Institution, which cooperate forwarding to the Processing Container messages related to internal events. In particular, each time a customer logs-in to the home banking Website, the Financial Institution submits a message containing the following fields:

- The IP address from which the customer logged into the home-banking Website.
- An identifier of the service accessed by the customer. While it is mandatory for the identifier to be unique among all the identifiers associated to the services offered by the same Financial Institution, different Institutions may use the same identifier for different services. (This allow Financial Institution to autonomously manage internal service identification.)
- An identifier of the customer that accessed the service. As for the service identifier, also this element has to be unique within the same Institution, but can be associated to different services offered by different Institutions.
- An identifier of the Financial Institution which registered the log-in event. This identifier has to be unique within the Processing Container and should be unique in case of multiple Processing Containers.

- The date and time at which the customer accessed the service. In order to avoid ambiguous values, all the date and time are expressed as UTC.

Moreover, a minimum clock synchronization is required by the event processing logic. Thus, two simultaneous log-in events have to be registered with the same time. However, they can be submitted in different instants, within a reasonable time interval. It is worth to be highlighted that it is not required to identify the customer and the service using the real values registered by the Financial Institution. Hence, they can be anonymized preserving their uniqueness by configuring the Preprocessors described in Section4.2.2.

4.4.3 Event preprocessing

The Event Processing of an event generated by a cooperating Financial Institution starts when the Event Manager of a Processing Element reads the message from the related JMS Topic. Then, the Event Manager analyzes the message, extracts the content and forwards the data to the Overlay Manager that inserts it into the DHT. As mentioned before, a MitM attack is characterized by single IP addresses used by several different customers in a short time interval. Then, a simple solution is to have the Processing Elements able to analyze a subset of messages having the same IP address and the same service identifier. This allows each Processing Element to detect a likely MitM attack by checking whether the number of accesses, from the same IP address to the same service within a bounded time window, exceeds a certain (static or dynamic) threshold. In order to correctly route all the events, the messageID is computed as the result of a cryptographic hash function of the concatenation of the identifier of the source IP address and of the identifier of the service.

$$messageID = hash(sourceIP + serviceID)$$

4.4.4 CEP rules

The component in charge of performing the event analysis is the Esper CEP engine, which can be configured using EPL processing rules. While a comprehensive explanation of all the processing rules used to detect MitM attacks is out of the scope of this thesis, in the following we provide a short description of the most important EPL statements, in order to give an idea of both the expressiveness of EPL rules, and the capabilities of the proposed architecture for collaborative event processing.

Listing 4.1: EPL statement for analyzing the event stream and raising MitM alerts

```
@Name: RaiseAlert
@Statement:
select *
from pattern
[
  ( every e1=Event) ->
  [threshold] e2=Event(e2.id=e1.id )
  while ( e2.log_time-e1.log_time < temporal_window)
]
```

As shown in Listing 4.1, the statement is similar to a SQL query. It defines the rule to raise an alert specifying a pattern of events that correspond to a MitM attack. Whenever this pattern is detected within the Event event stream, it raises an alert. This pattern is verified if there are at least a configurable number (threshold) of different events having the same identifier and contained within a specified time frame.

Listing 4.2: EPL statement for handling partial results from other Processing Elements

```
@Name: EnhanceWithSuspectedIP
@Statement:
insert into EnhancedEvent
select
Event.raw.id as id,
Event.raw.source_ip as source_ip,
Event.raw.userid as userid,
Event.raw.log_time as log_time,
Event.raw.OriginFI as OriginFI,
Event.raw.service as service,
SuspectedIpEvent.count as
count from Event unidirectional left outer join
SuspectedIpEvent.std:unique(source_ip) on
Event.raw.source_ip = SuspectedIpEvent.source_ip
```

An other example of EPL statement presented in Listing 4.2. This statements merges the Event data stream with the SuspectedIpEvent data stream, which is the data stream that includes the IP addresses which have been identified as suspicious by another Processing Element. This data stream may consist of the partial results of CEP engine (i.e. with respect to the previous statement, a similar one characterized by a lower threshold can generate events containing suspicious IP). Since these IP addresses are suspected to misbehave, they will undergo a stricter scrutiny

with respect to other IP addresses. This EPL statement augments the Event data stream by adding a count property to all the events that have been generated by a suspected IP address. The resulting event stream is named EnhancedEvent. Finally, the EnhancedEvent data stream is analyzed to detect likely MitM attacks. In this case the statement may be improved by automatically increasing the length of the time frame if the considered IP addresses have already been suspected by another CEP engine.

4.5 Experimental validation

The prototype described in Section 4.3 has been validated using a testbed composed by several (both physical and virtual) networked machines, geographically distributed among Italy, Hungary and Israel. The three cooperating financial institution were simulated by deploying three distinct instances of the Gateway described in Section 4.2.2 and 4.3.1. Each Gateway sent a stream of Event to the Processing Container devoted to MitM detection, at a rate of about 1000 events per minute. In our tests, this Processing Container comprised 8 Processing Elements. A custom dataset, composed by a baseline of normal activities involving legitimate users, has been build to verify whether the proposed CEP algorithm was able to detect MitM attacks without triggering false positives. Events related to normal activities do not match the pattern described by the EPL statement presented in Figure 4.1. Hence, any alert raised by the Processing Container and related to one of these events has to be considered a false positive, and should be avoided. Then, we artificially introduced into the dataset events related to three different instances of MitM attacks. Two of them were characterized by "high volume" activities, thus meaning that the number of customers login activities to the same service and IP address from distinct users were above the detection threshold. The third instance of MitM attack represented a "low volume" MitM attack. In particular, events related to this attack were below the normal detection threshold, but above the extended detection threshold used for the analysis of events generated by suspicious IP for which a partial result has already been generated by a CEP engine. The positive results obtained by the experiments performed on the testbed, confirmed the validity of the proposed architecture. The introduction of suspicious IP addresses allows to finely tune threshold avoiding false positive alerts. Indeed, none of them has been raised during the experiment, and both "high volume" and "low volume" instances of MitM attacks were detected as soon as a sufficient number of events had been received by the Processing Container. Due to testbed limitation, the 8 Processing Elements were deployed in 8 different virtual machines deployed on 2 physical hosts, thus we cannot provide an exact performance evaluation in terms of time consumed per event processed

or viceversa number of elements processable per Processing Element. However, during the experiment the Processing Elements were able to sustain an aggregated event throughput of about 3000 events per minute (1000 events per minute from each Gateway), and, most important thing, the resource monitoring confirmed that the workload was evenly balanced among all the Processing Elements.

4.6 Concluding remarks

In this chapter we proposed an innovative solution for collaborative geographically distributed intrusion detection and alert dissemination [49]. We proposed an architecture suitable for the cooperation among different organizations, which provides analysis of huge amount of data leveraging Complex Event Processing. It can be configured to use different CEP engine technologies and to implement different events processing logics. The components deployed within each organization offer a multilayer design that allows to perform different preprocessing tasks before submitting data to the Processing Container. On the other hand, the Processing Container, leveraging a DHT overlay, avoids single points of failures, guarantees high scalability, load balancing and self organization capabilities.

Chapter 5

Network Intrusion Detection Systems for Mobile Environments

5.1 Introduction

The wide and constantly growing diffusion of mobile devices is changing the way Internet is accessed. These devices allow users to be always connected, thus encouraging the diffusion of new services as well as mobile versions of existing ones, and bringing the fraction of traffic generated by mobile devices to overtake the portion generated by static hosts [74]. This relentless transformation is drawing the attention of both industry and researcher towards security issues affecting mobile network activities. Several protocols exist [9,10,79–82] that support *seamless mobility* [53], or transparent mobility, hence allowing users to roam among different networks without interrupting established connections. Furthermore, it is worth to highlight that another form of node mobility is represented by live migration of Virtual Machines [30], which is a widely adopted solution for load balancing and performance management both within an individual datacenter [15,67] and between geographically distributed locations [23,26].

While transparent mobility is an important requirement for all the network applications which require a stable connection, it also represents a source of weakness, due to its negative impact on different network security technologies designed to protect only static nodes. In this chapter, we present a new attack strategy, called *mobility-based NIDS evasion technique* [31], and the related countermeasures. Attackers leverage mobility-based NIDS evasion technique to perform "stealth" attacks undetectable even by modern state-of-the-art NIDS. This kind of attack exploits the differences in routing paths which characterize mobile nodes and is not related neither to specific vulnerabilities of some protocol definition nor to bad protocol implementations. We describe the key aspects of this new attack

strategy and then analyze the specific issues related to three different protocols: Wi-Fi [9], Mobile IP [81] and IPv6 [82][1]. For each of them we present the main steps which allow an attacker to perform attacks undetectable even by NIDSs based on stateful analyses [100]. There are two main problems which prevent defensive solutions to thwart this treat: existing NIDSs do not take into account specific characteristics of mobile environments, and distributed solutions, consisting of multiple sensors deployed to monitor complex networks, are not able to effectively cooperate to detect these "stealth" attacks. Indeed, most of the existing distributed architectures of cooperative NIDSs (e.g., centralized [60, 99, 110], hierarchical [41, 83, 102, 106, 116], peer-to-peer [56, 68, 70, 72, 118]) support information sharing through the exchange of pre-processed data, such as alerts, lists of IP addresses or ad-hoc messages, and are not able to share low level information required to detect mobility-based evasion techniques. On the other hand, few architectures have been proposed which leverage some sort of internal state migration (e.g. [32, 34, 101, 105]), but they have been designed to perform load balancing, and their state migration algorithms are not effective against this new attack strategy.

In order to solve these problems, we propose a new cooperation scheme which allows to exchange internal state information among multiple cooperative NIDSs to detect attacks which exploit mobility-based NIDS evasion technique [36]. We present a novel *state migration* protocol, highlighting the main characteristics and describing the specific issues related to the protocol which provides seamless mobility. We then introduce a complete framework based on the proposed cooperation scheme, which enable distributed NIDS to monitor mobile nodes roaming within different networks and support the most popular protocols for node mobility, such as Mobile IP, Mobile IPv6 and Wi-Fi. The framework has been designed as a modular solution in order to guarantee great flexibility both in terms of deployment and of future extension. In particular, the implemented prototype consists of a patch to the Snort [100] source code, which extends its capabilities, a lightweight agent, which interacts with Snort, and a set of plugins that are in charge of managing different protocols.

The validation of the proposed solution has been performed running several experiments involving multiple mobile protocols in different network scenarios. Both efficacy and efficiency have been evaluated analyzing the detection rate in presence of "stealth" attacks and the impact on performance for different migration rates and protocols.

The rest of the chapter is organized as following. In Section 5.2, we briefly introduce some basic concepts concerning NIDS functioning and common eva-

[1] In this chapter, we use Mobile IP and Mobile IPv4, as well as IPv6 and Mobile IPv6, interchangeably.

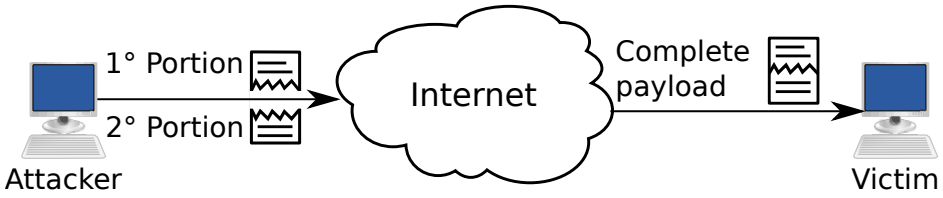

Figure 5.1: NIDS evasion through simple fragmentation

sion techniques. The mobility-based NIDS evasion technique is described in Section 5.3, while in Section 5.4 we present the solution proposed to thwart this evasion technique for different roaming events. We analyze the protocol specific characteristics of the evasion technique and related solution focusing on three popular protocols: Wi-Fi in Section 5.5, Mobile IPv4 in Section 5.6, and Mobile IPv6 in Section 5.7. We describe the implementation details of the prototype used to validate the entire solution, in Section 5.8, followed by the performance evaluation results in Section 5.9. We outline main conclusions in Section 5.10.

5.2 Background

Before introducing the mobility-based NIDS evasion technique we briefly explain how normal fragmentation works and how existing NIDS solved this problem. As described in Section 2.1.1, NIDSs analyze sequence of network packets representing live network traffic or previously registered network traces. In particular, signature-based NIDSs analyze network packets looking for strings of byte matching the signatures of known attacks, which are provided in the form of regular expressions. However, several techniques exist [85, 95, 97, 113] whose aim is to avoid NIDS detection. A well known technique is packet fragmentation. It consists in partitioning the malicious payload and transmitting it using multiple packets, in order to obtain fragments which are too short to match the related signature. This technique was particularly effective against former NIDSs that were based on stateless analysis. Indeed, stateless NIDSs analyze only one packet at once as a self-contained information source, without correlation with other network packets captured by the network interface or read from a file. Then, all the fragments smaller than a certain threshold are able to evade the regular expressions used for the signatures.

In Figure 5.1 we propose an example of fragmentation-based NIDS evasion. The node on the left represents the Attacker which is trying to exploit a remote vulnerability of the Victim, the node on the right, by sending a malicious payload. All the network traffic is analyzed by the NIDS which is monitoring the network hosting the Victim. Assuming that the Attacker is aware of the NIDS and

of the signatures it uses, detection can be evaded by splitting into two or more fragments the malicious payload, which are labeled *Fragment 1* and *Fragment 2* in Figure 5.1. Both the portions of the two fragments are small enough to avoid detection, then the Attacker is able to perform its attack evading the NIDS.

Fortunately, all modern NIDSs are based on *stateful* analysis, meaning that they are able to build and maintain *state* information extracted from the analyzed network traffic. The information contained within each network packet is used to create and update the NIDS state, which is then analyzed using the detection algorithm. Hence, even if none of the individual fragment contains enough information to match the related signature, it is still possible for a stateful NIDS to detect an intrusion by analyzing the state, which is composed by the information extracted from all the gathered network packets. However, node mobility may compromise the NIDS capabilities to monitor all the network packets related to a mobile node, then also stateful NIDS can be evaded. We describe how it can be done in the next Section 5.3

5.3 Mobility NIDS evasion

In this section we describe mobility-based NIDS evasion [31], a tecnique that allows an attacker to exploit network mobility and traditional NIDS evasion techniques [85] based on attack fragmentation in order to avoid detection by state-of-the-art stateful NIDS.

In order to clarify the concepts explained in this section, we firstly introduce a brief description of the main terms:

Mobile Node. A node that can migrate from one network to another, while still being reachable due to the mobility support provided by some protocols.

Correspondent Node. A peer node with which a mobile node is communicating. If not specified, we are not interested in its connectivity, then it could be either a stationary host or another Mobile Node.

Attacker. The attacker is the node that tries to exploit a vulnerability of the victim. In particular, it could be the Mobile Node or The Correspondent Node, in both cases it performs the attack leveraging mobility NIDS evasion.

Victim. The Victim is the target of the attack performed by the Attacker. It could be either the Mobile Node or the Correspondent Node.

Home Network. We use this term to indicate the first network joined by the Mobile Node.

Foreign Network. It is the network reached by the Mobile Node when it roams from the Home Network to another network.

Correspondent Network. It is the network where the Correspondent Node is deployed.

We identified three different scenarios where mobility-based NIDS evasion can be applied: *mobile victim and fixed attacker*, *mobile attacker and fixed victim* and *mobile victim and mobile attacker*, presented in Sections 5.3.1, 5.3.2, and 5.3.3, respectively.

In the next sections the assumptions are the following:

- both the *Home Network* and the *Foreign Network* provide support for *seamless mobility* (i.e. without interruption of transport level connections);
- both the *Home Network* and the *Foreign Network*, and possibly the Correspondent Network, are monitored by a state-of-the-art stateful NIDS;
- the attack targeting the victim is based on a malicious payload built to exploit a remote vulnerability;
- it is possible to divide the malicious payload in (at least) two portions;
- a NIDS is not able to detect the attack by analyzing only a portion of the malicious payload.

The first assumption is satisfied by several technologies and network protocols, such as Mobile IPv4 [79–81], IPv6 [82] and protocols for layer-2 handover across wireless access points [10, 29] or across heterogeneous networks [11]. The second assumption can be easily met in realistic network topologies and finally the latter assumptions represent a typical scenario of network-based attacks. We also highlight that malicious payload fragmentation is not related to packet fragmentation at the network layer or below and the attack is not exploiting a programming flaw of the deployed NIDS nor a vulnerability of a particular protocol implementation.

5.3.1 Mobility-based NIDS evasion: mobile victim

Considering the scenario shown in Figure 5.2 and Figure 5.3, the *Mobile Node* on the left is the *Victim*, it is deployed in a network that allows node mobility and is communicating with a Correspondent Node, which is an *Attacker* trying to exploit a remote vulnerability. The two clouds on the left represent the origin and destination network, labeled as *Home Network* and *Foreign Network* respectively. They offer either network or lower-layer mobility support and all the traffic

related to a node within them is monitored by a stateful NIDS, in particular the Home NIDS and Foreign NIDS. In order to exploit the mobility-based evasion technique, the attacker needs a method to detect the migration of the victim from the *Home Network* to the *Foreign Network*. Different communication protocols providing mobility support offer different ways to obtain this information, such as the exchange of particular packets or the introduction of noticeable delays. Specific details related to WiFi, Mobile IP and IPv6 will be discussed on the next sections.

The attack proceeds as follows:

1. the Correspondent Node sends the first attack fragment;
2. the Correspondent Node waits for the *Mobile Node* to roam to the *Foreign Network*;
3. the Correspondent Node sends the second attack fragment.

Depending on how mobility is implemented, the second attack fragment may or may not be routed to the *Mobile Node* through the *Home Network*. We distinguish between two cases: *Tunneling* and *Direct Communication*.

Tunneling

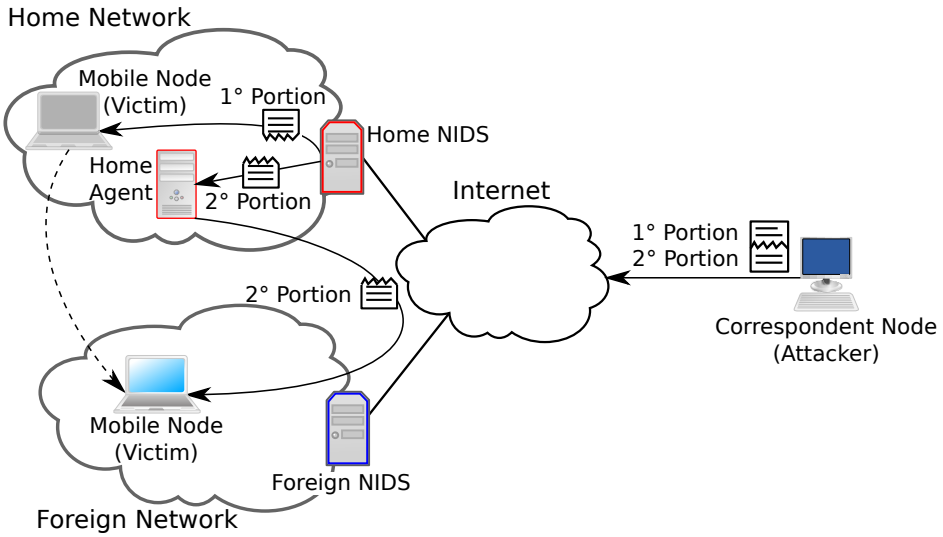

Figure 5.2: First evasion scenario: mobile victim and fixed attacker use tunneling communication

Some protocols use a tunneling routing scheme, meaning that when a *Mobile Node* is within a *Foreign Network*, all the packets are routed through the Home

Network, where an Home Agent is in charge to process them. This scheme grants to the *Mobile Node* to migrate from one network to another one while the Correspondent Node continues to reply to an address within the *Home Network*. The mobility based NIDS evasion technique that exploits this routing scheme is represented in Figure 5.2. The first portion of the attack is sent directly to the *Mobile Node* through the *Home Network*, where the Home NIDS receives and processes the packet without rising any kind of alert, but storing its content for further analysis combined with the content of upcoming packets. The second portion of the attack is sent only after the migration of the *Mobile Node* and reaches the *Foreign Network* passing through the *Home Network*. The Home NIDS is able to process the second fragment and the analysis combined with previous packets detects the attack and eventually rises an alert. On the other hand, the Foreign NIDS receives and processes only the second portion of the attack, which, without the first fragment, is not enough to trigger the detection of the NIDS. This means that the *Mobile Node* has been compromised within the *Foreign Network*, but the Foreign NIDS was unable to detect the attack. Only the Home NIDS, and possibly the NIDS of the Correspondent Network, detected the attack, but they cannot act neither to sanitize the *Mobile Node* nor to protect the nodes within *Foreign Network* from the compromised machine.

Direct Communication

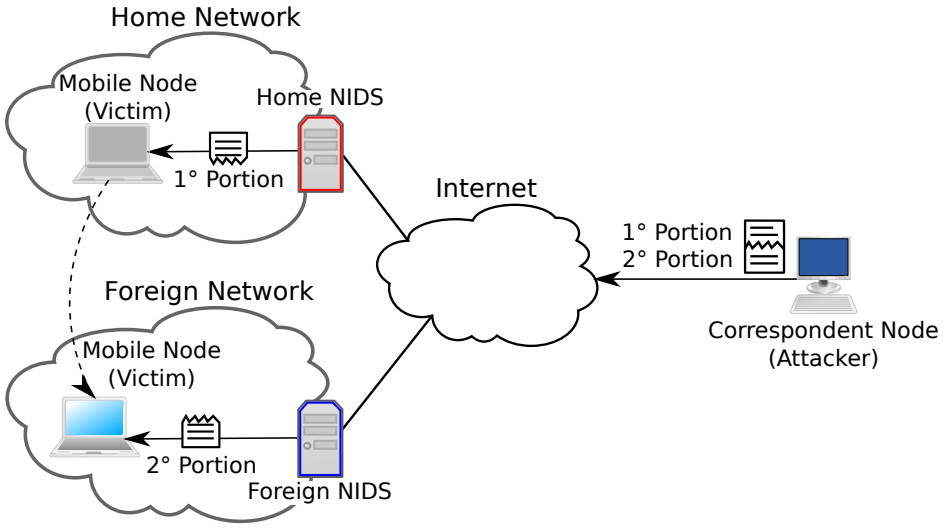

Figure 5.3: First evasion scenario: mobile victim and fixed attacker use direct communication

Some protocols providing mobility support enable a *Mobile Node* hosted in

a *Foreign Network* to directly send and receive packets to/from its Correspondent Node. This case is represented in Figure 5.3. With respect to the tunneling scheme, the main difference is that the second portion of the attack sent by the Correspondent Node reaches the *Mobile Node* without passing trough the *Home Network*. The result is that the Home NIDS receives only the first fragment, while the Foreign NIDS receives only the second one, then both the NIDSs are unable to detect the attack.

5.3.2 Mobility-based NIDS evasion: mobile attacker

With respect to the previous case study, in this scenario the roles are reversed: the *Mobile Node* is the *Attacker* that takes advantage of its own mobility to evade detection, while the Correspondent Node is the *Victim*. Figures 5.4 and 5.5 represent this scenario in case of direct communication or tunneling, respectively.

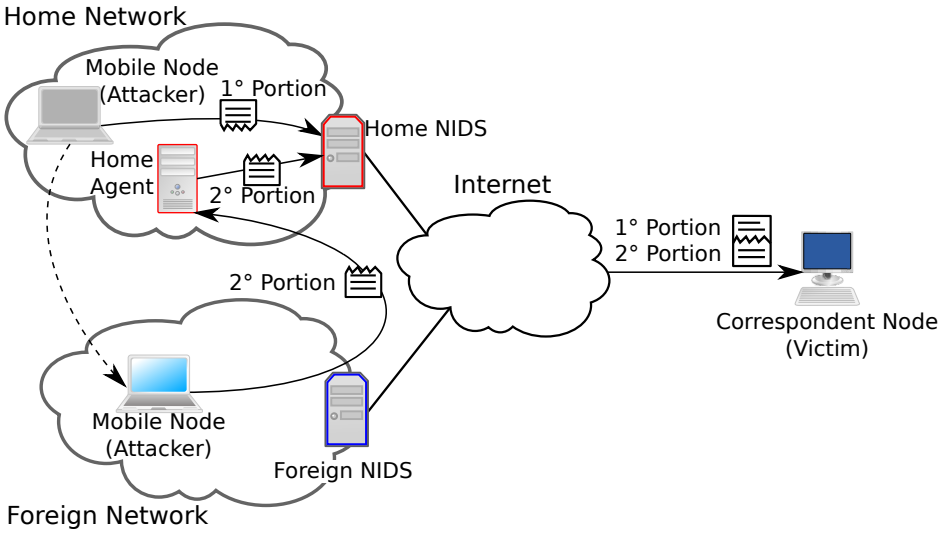

Figure 5.4: Second evasion scenario: mobile attacker and fixed victim with Tunneled Communication

The sequence of activities performed by the attacker is as follows:

1. the *Mobile Node* sends the first attack fragment;
2. the *Mobile Node* roams to the *Foreign Network*;
3. the *Mobile Node* sends the second (last) attack fragment.

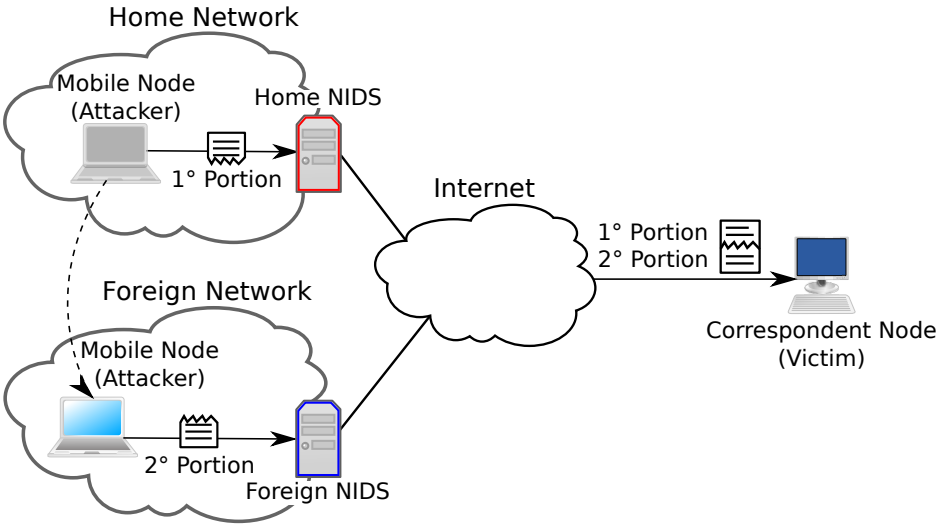

Figure 5.5: Second evasion scenario: mobile attacker and fixed victim with Direct Communication

Tunneling

The first step of the attack takes place when the *Mobile Node* is within the *Home Network*. The first fragment of the attack is sent to the Correspondent Node and is intercepted and analyzed by the Home NIDS, which does not have enough information to detect the attack, then it simply updates its state information.

After the migration to the *Foreign Network*, the second portion of the attack is sent to the Correspondent Node through the *Home Network* as shown in Figure 5.4. This fragment is intercepted by both the Home NIDS and the Foreign NIDS. While the former is able to detect the attack combining the two portions of the attack, the Foreign NIDS misses the first fragment, hence is unable to detect the attack. As a result, the *Mobile Node* is able to exploit mobility evasion in order to complete remote attacks while connected to a network unaware of its actions.

Direct communication

Similarly to the previous case, the first fragment of the attack is sent through the *Home Network*, and is intercepted and analyzed by the Home NIDS, which updates its state information, but it does not have enough information to detect an intrusion attempt. The main difference is that the second portion of the attack is directly sent from the *Foreign Network* to the Correspondent Node. Then only the Foreign NIDS receives and processes the second fragment, but as seen before, it is unable to recognize the attack.

As a result, none of the two NIDSs deployed in the Home and Foreign net-

works can detect the attack. This is caused by the lack of information extrapolated by the two NIDS from the single fragments analyzed.

While a NIDS deployed within the Correspondent Network will eventually detect the attack, a Mobile *Attacker* can exploit mobility to perform stealth attacks without the administrators of the visited networks being able to prevent, stop, or even detect him.

5.3.3 Mobility-based NIDS evasion: mobile target and attacker

While in the previous scenarios a NIDS possibly deployed within the Correspondent Network was able to detect the attack, in the last scenario, both the *Attacker* and the *Victim* are *Mobile Nodes* and the former takes advantage of a combination of the firsts two scenarios, to completely avoid detection. As previously stated, we suppose that the attacker is able to detect when the victim migrates to a new network, and we assume that all the involved networks (attacker's *Home Network*, attacker's *Foreign Network*, victim's *Home Network* and victim's *Foreign Network*) are monitored by stateful NIDSs.

The attack proceeds as follows:

1. the attacker sends the first attack fragment;
2. the attacker waits for the victim to roam to the *Foreign Network*;
3. the attacker roams to a (different) *Foreign Network*;
4. the attacker sends the second attack fragment.

Note that the order of steps 2 and 3 can be inverted without changing the attack outcome.

The consequence of victim roaming from victim's *Home Network* to victim's *Foreign Network* is that the Home NIDS's and the Foreign NIDS's possibility to analyze the whole malicious payload are reduced as described in subsection 5.3.1 and at least the Foreign NIDS is unable to detect the attack. At the same time, the migration of the attacker to a *Foreign Network* before sending the second portion of the attack denies to the NIDS deployed in the attacker's *Foreign Networks* to receive the first attack fragment, as shown in Figure 5.4. Hence, it is not able to detect the malicious behavior of the attacker, as described in subsection 5.3.2.

If a tunneling routing scheme is implemented by both victim networks and attacker networks, the second portion of the attack passes trough all the four networks. Then the two Home NIDSs are able to process the whole malicious payload and detect the attack, while the *Foreign Network*s which are hosting both the attacker and the victim cannot take any kind of specific countermeasure because they are unaware of the successful attack.

On the other hand, if both victim networks and attacker networks enable direct communication schemes, both the portions of the attack are directly routed to the victim as shown in Figure 5.3 (victim side) and Figure 5.5 (attacker side). In this case all the deployed NIDSs receive only one fragment of the malicious payload, then neither the victim network nor the attacker network are able to detect the attack, which can be considered completely stealth.

5.4 Solution through NIDS cooperation

The main problem of mobility-based NIDS evasion is caused by the fragmentation of relevant state information among distributed NIDSs, thus preventing them to build a complete state, which is necessary to detect a fragmented attack.

To address this issue we propose a cooperative solution, based on distributed stateful NIDSs which exchange state information among each other. The concept behind this proposal is to extract state information concerning active connections of the *Mobile Node* from the NIDS deployed in the *Origin Network*, and to merge this data with the state information stored in the NIDS that monitors the *Destination Network*. These actions are called *state export* and *state import* respectively, while the entire process is defined as *state migration*. This solution allows the NIDS that monitors the *Foreign Network* to receive all the state information related to the previous network activities of the *Mobile Node*, thus preventing an attacker to exploit mobility to evade detection.

In order to manage the process of *state migration* we introduce the *External Agent (EA)*, a modular software able to interact with the NIDS deployed within its network and with other *EAs* deployed in different networks. This solution requires only limited changes to the code of the NIDS and at the same time, being external and modular, it enhances its flexibility and possibly its capability to support different mobile protocols, by adding new modules, namely *Plugins*.

To better explain our solution we divide our discussion in three different cases, corresponding to the possible migrations performed by the *Mobile Node*:

1. **First Migration:** the *Mobile Node* roams from the *Home Network* to a *Foreign Network*;
2. **Return to Home:** the *Mobile Node* returns to its *Home Network*;
3. **Further Migration:** the *Mobile Node* roams from a *Foreign Network* to another *Foreign Network*.

In the following descriptions of these scenarios, we assume that the state migration takes place via a secure channel in order to guarantee authentication of the *External Agents*, non-repudiation, message integrity and confidentiality of exchanged

information. With authentication of the *External Agents* we also imply that a trust relationship already exists between the networks among which the *Mobile Node* is roaming. In fact, importing into a NIDS untrusted data could compromise its integrity and thereby the network security.

We assume the *External Agent* is able to analyze the whole network traffic, or at least the portion generated by the *Mobile Nodes*, within its network. This is necessary to detect when a *Mobile Node*, arriving from its *Origin Network*, joins the network where the *External Agent* is deployed.

The last assumption is that the *External Agent* deployed in the *Destination Network* reached by the *Mobile Node* is able to contact the *External Agent* of the *Origin Network*, or at least of the *Home Network* (if they are not the same). The latter requirement is easy to achieve when the roaming is performed at the network layer exploiting the mobility provided by Mobile IP and IPv6. In fact, these protocols require the presence of an *Home Agent* deployed in the *Home Network* which is in charge of managing *Mobile Nodes* migrations. Thus deploying the *External Agent* (our proposed module) together with the *Home Agent* (defined by protocol specifications) it results simple to reach the former with *State Export* requests. At the same time, it is also possible to use an a priori knowledge acquired from the information related to the trust relationship established among the involved networks.

5.4.1 First Migration

In the first scenario the *Mobile Node* roams from the *Home Network* to a *Foreign Network*. The messages exchanged during the State Migration process are shown in Figure 5.6: on the left we represent the *Mobile Node* migration omitting the specific messages required by the protocol. On the right we report each single message exchanged among the deployed *External Agents* and the NIDSs monitoring the *Home* and the *Foreign Network*.

When the *Foreign External Agent*, in the following *Foreign EA*, detects the arrival of a new *Mobile Node*, it retrieves information concerning its *Home Network*. This operation could be performed in different ways depending on the protocol used by the *Mobile Node* for migration (more details are reported in Sections 5.5, 5.6 and 5.7 focusing on WiFi, Mobile IP and IPv6, respectively). Once the *Foreign EA* has obtained the address of the *Home External Agent*, in the following *Home EA*, it sends to it a request asking for the state information related to the *Mobile Node*. This request is analyzed by the *Home EA* which verifies its consistency (whether the cited Node has really left the *Home Network* and joined the *Foreign Network*), then it exports the requested state information stored by the Home NIDS and sends the extracted data to the *Foreign EA*. This last element pre-processes the received information accordingly to the used protocol and guar-

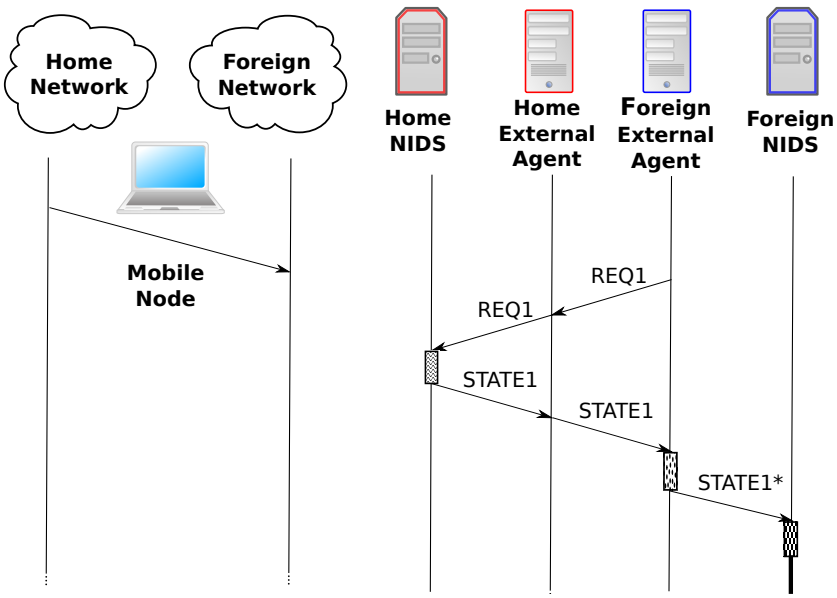

Figure 5.6: First Migration sequence diagram: *Mobile Node* roams from the *Home Network* to a *Foreign Network*

antees the correct normalization of the content and that the format is the same used by the Foreign NIDS. This task is simple if both the *Home Network* administrator and the *Foreign Network* administrator deploy the same NIDS. Once the Foreign NIDS merged the received data into its internal state, all the communications related to the *Mobile Node* are monitored correctly. If an attacker tries to exploit one of the mobility-based evasion scenarios described in Section 5.3, the fragments of the malicious payload are merged into the state information of the Foreign NIDS, which now is able to analyze the whole payload and detect the attack. Hence, *state migration* driven by mobile activities prevents mobility-based evasion.

5.4.2 Return to Home

In the second scenario we consider the *Mobile Node* returning to its *Home Network*. The messages exchanged during the State Migration process are shown in Figure 5.7.

With respect to the previous scenario, all the roles, but the *Mobile Node*, are reversed. When the *Home EA* detects the return of the *Mobile Node*, it has to contact the *Foreign EA* in order to ask the state information related to the *Mobile Node*. If the *Home EA* stored the address of the *Foreign EA* when the *Mobile Node* left the *Home Network*, it can directly contact the *Foreign EA*; otherwise, it has to retrieve the necessary information from the captured network traffic. When

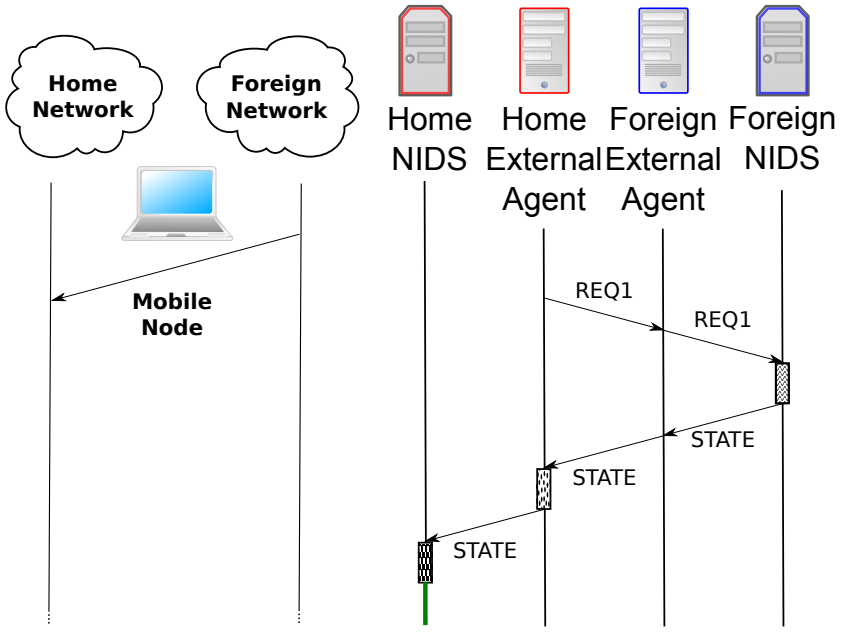

Figure 5.7: Return to Home sequence diagram: *Mobile Node* returns to its *Home Network*

the request reaches the *Foreign EA*, the following steps are executed. They are similar to those described in the previous scenario: the *Foreign EA* replies to the *Home EA* with the state information exported from the Foreign NIDS. Then, the *Home EA* pre-processes the received data and imports the state information into the Home NIDS.

5.4.3 Further Migration

The migration between two *Foreign Networks*, without returning to the *Home Network* is the last activity that a *Mobile Node* can perform. The main difference between this scenario and the First Migration phase described in Section 5.4.1 is that the *Mobile Node* reaches the *Destination Network* from another *Foreign Network* (*Origin Network*). Hence the relevant state information is only known by the Foreign NIDS deployed in the *Origin Network* (Origin NIDS). This could be a problem, because the chance exists that the *Foreign EA* of the *Destination Network* (*Destination EA*) cannot extract the address of the *Origin EA* from messages sent by the *Mobile Node*. As a consequence, the *Destination EA* is unable to issue a request for the relevant state information directly to the *Origin EA*.

To solve this problem, the *Destination EA* integrates two concurrent approaches. When the *Destination EA* detects the arrival of a new *Mobile Node*, it analyzes

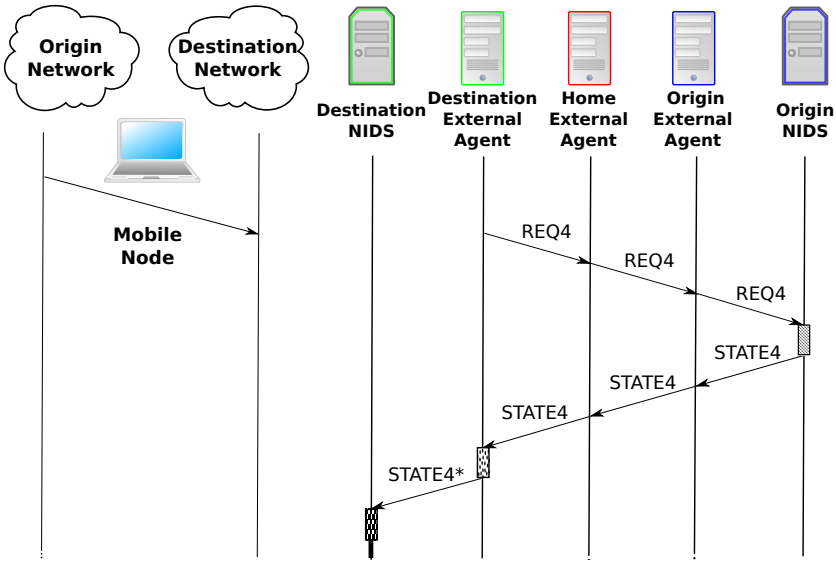

Figure 5.8: Further Migration sequence diagram: *Mobile Node* roams from a *Foreign Network* to another *Foreign Network*

network packets sent by the *Mobile Node* and tries to extrapolate some useful information to detect the *Origin EA*. At the same time it looks for packets related to *Home Network*. If the former approach succeeds, the *Destination EA* directly contacts the *Origin EA* and the sequence of steps is the same described for the First Migration in Section 5.4.1. Otherwise, the *Destination EA* issues a *state export* request (REQ4 in Figure 5.8) to the *Home EA*. Indeed, the *Home EA* knows the address of the *Origin EA*, since the *Origin EA* has already contacted the *Home EA* asking for the state information related to the *Mobile Node* when it roamed to the Origin Network. Hence, the *Home EA* can forward the state export request to the *Origin EA*. Then, the *Origin EA* sends the state export request to the Origin NIDS, that contains the relevant state information. This state information (STATE4, in Figure 5.8) is finally forwarded through the *Origin EA* and the *Home EA* to the *Destination EA*, where it is pre-processed. The pre-processed state information, ready to be imported, is then delivered to the Destination NIDS, that is now immune to mobility-based NIDS evasion techniques.

5.5 WLAN environments

In this section, we describe a mobility-based NIDS evasion technique and the related countermeasures with respect to Wireless LANs based on WiFi.

The protocols defined by the 802.11 working group are used for Wireless Lo-

cal Area Networks (WLAN), and represent one of the most used medium to connect mobile devices to the Internet. In WLAN the main elements are the nodes equipped with wireless network interfaces and the Access Points, which allow those nodes to connect to the (wired) network infrastructure. In order to join the network, a node has to successfully complete the authentication and the association processes by exchanging specific packets with the Access Point. This procedure has to be repeated each time a Mobile Node decides to migrate (i.e. preferring an AP with a stronger signal).

5.5.1 WiFi evasion

With respect to mobility-based evasion technique described in Section 5.3, here we describe the specific issues of WiFi, which provides only the direct routing scheme. When a Mobile Node roams from an AP to another one belonging to the same network, the migration is completely transparent to the network layer and the levels above, thus allowing the *Mobile Node* to maintain uninterrupted connections with other nodes. The mobility-based evasion technique can be exploited when the APs involved in the roaming process route the network traffic through different paths monitored by different NIDSs.

This scenario reflects large organizations where the network is divided into different and independent departments. The WLAN is one for the entire network (same *essid* and same subnetwork), thus enabling *Mobile Nodes* to freely move around within the organization, but each AP routes the network traffic trough the gateway of its department. Therefore, the NIDSs deployed in each department are able to monitor only a portion of the network traffic generated within the WLAN.

A Mobile Attacker can easily roam from an AP to another simply forcing his Wireless NIC to associate to the desired AP, possibly without physically moving. Then the Attacker can evade the NIDSs monitoring his network by splitting the malicious payload into two portions and sending them through two different APs. On the other hand, the Attacker can target a Mobile Node. To detect the migration of a Mobile Node from one AP to another one, an Attacker can monitor network traffic looking for messages related to the roaming process, such as the packets exchanged during the association to the new AP. In this case, the Attacker sends the first portion of the attack while the Victim is associated to an AP. Then, it waits until the association process to the second AP has been completed, and sends the second portion of the attack.

5.5.2 WiFi cooperation

In order to enable the NIDS cooperation scheme proposed in the previous Section 5.4, the External Agents need to detect the migration of a Mobile Node from

one network to another one. In addition, the External Agents require information about the Home/Origin Network in order to request the State Information related to the previous network activity of the Mobile Node. The 802.11 standard does not require a specific agent in charge of managing the roaming processes of the Mobile Nodes. In fact, the Access Points have to maintain a local table containing the information of the associated nodes and they only need to know when one of them is no more associated, while other components of the network do not know neither through which Access Point it is possible to reach a wireless node nor in which department the wireless node is operating. This means that it is difficult to trace the movements of Mobile Nodes migrating from network to network.

In order to solve this problem several approaches exists, including (but not limited to):

- Introduce two extra software components: an agent deployed within each department, in charge of reporting the detected wireless nodes, and a centralized module, which collects information from each agent and is able to reply to the requests concerning wireless nodes locations. Then, the External Agent can be configured to contact the central collector to obtain information about the origin network of the Mobile Nodes.

- Provide to each External Agent a list of the nearby External Agents. The assumption is that a Mobile Node can migrate only between two Access Points in the same neighborhood. When an External Agent detects a new Mobile Node within the monitored network segment, it broadcasts a state export request to all the neighbors. Then, it receives a reply only from those in possession of state information related to the Mobile Node.

- Leverage the characteristics of specific protocols. For instance, in a WiFi network which requires user authentication and provides AAA services through the RADIUS protocol [89], it is possible to trace Mobile Nodes position by deploying an extra agent within the RADIUS server. This agent is in charge of registering the origin of the accepted authentication requests and the External Agents can be configured to use it to obtain information about the origin network.

Once the Foreign External Agent retrieved the information about the Home Network, it sends a request to the Home External Agent, starting the process described in Section 5.4.1.

5.6 Mobile IPv4 networks

The goal of Mobile IP is to allow a mobile host to communicate with other nodes after changing its point of attachment to the Internet without changing its IP address. In particular, a *Mobile Node* is always reachable at its Home Address also when attached to a network different from the Home Network. In order to reach this result, the *Mobile Node* needs to support Mobile IP and both the Home Network and the Foreign Network have to deploy a mobility agent, the Home Agent and the Foreign Agent respectively, while nothing is required to the the correspondent nodes which communicate with the *Mobile Node*.

5.6.1 Mobile IP evasion

With respect to mobility-based evasion technique described in Section 5.3, here we describe the specific issues of Mobile IPv4. In particular, while the description of the attack for the tunneling routing scheme is the same, Mobile IPv4 lacks of a full direct routing scheme, providing an half-direct/half-tunneling scheme called triangular routing. The idea is to enable the *Mobile Node* to directly send packets to the Correspondent Node, while receiving the replies through the Home Network. The main advantage is to reduce the latency of the communication at least in one direction without extra requirements for the Correspondent Nodes. However, despite the effort to reduce the latency, it is worth to be highlighted that both the routing scheme supported by Mobile IPv4 introduce a noticeable delay in the communication. Thus the attacker can exploit this timing difference in order to detect the migration of the victim from his Home Network to a Foreign Network. With respect to the three scenarios described in Section 5.3:

Mobile Victim : From the attacker's point of view, when the *Mobile Node* is the victim, both the tunneling mode and triangular routing scheme are the same, because both the two fragments sent to the *Mobile Node* pass trough the Home Network. In fact, with triangular routing enabled, only the packets sent by the Victim are directly routed to the attacker, instead the packets sent by the latter need always to pass trough the Home Network.

Mobile Attacker : When the attacker enables triangular routing, he can directly send the second portion of the attack to the victim, thus the NIDSs monitoring the attacker are unable to detect the malicious payload, which instead may be detected by the NIDS possibly deployed in the Correspondent Network.

Mobile Victim and Attacker : If both the Attacker and the Victim are *Mobile Node*s which use triangular routing, the attacker can avoid detection by the

NIDSs monitoring himself and by the victim's Foreign NIDS, but the whole malicious payload is eventually detected by the victim's Home NIDS. In fact, from the attacker's point of view, the second portion of the attack is directly sent to the Victim, but it actually passes trough the victim's Home Network because the victim receives packet trough the Home Agent.

As a result, using Mobile IPv4, the attack is always detected at least by the victim's Home NIDS, but never by the victim's and/or attacker's Foreign NIDS.

5.6.2 Mobile IP cooperation

In a Mobile IP network, when a *Mobile Node* joins a Foreign Network it has to register its new position to its Home Agent. To accomplish this task there are two different procedures. When the *Mobile Node* receives its Care-of Address from the Foreign Agent, for the *Mobile Node* it is mandatory to send registration messages to its Home Agent through the Foreign Agent; Otherwise, if the *Mobile Node* uses a co-located Care-of Address, obtained without the intervention of a Foreign Agent, it has to exchange registration messages directly with the Home Agent. In both cases, two specific messages are involved: Registration Request and Registration Reply. The main fields contained in the former are the Home Address of the *Mobile Node*, the Home Agent's address, the identification field, the Care-of Address and its lifetime. The request is sent to the Home Agent which may or may not accept the registration, but in both cases it has to reply to the *Mobile Node* using the Registration Reply message.

The presence of these two registration messages can be exploited by the *Foreign External Agent* to detect the arrival of a new *Mobile Node* and to extrapolate the necessary information to start the State Migration process. In particular the Home Address, the Care-of Address and the Home Agent address can be retrieved from the Registration Request message, while from the Registration Reply the *Foreign External Agent* can verify if the registration succeeded or failed. In the former case the *Foreign External Agent* sends the State Export Request to the *Home External Agent* asking for the state information related to the Home Address of the *Mobile Node*. Then, the received data is pre-processed by the *Foreign External Agent* and imported into the Foreign NIDS.

5.7 Mobile IPv6 networks

Mobile IPv6 (MIPv6) is an extension for IPv6 which allows nodes to remain reachable while migrating from one network to another through the usage of its Home Address and a new Care-of Address obtained from each joined networks.

The Mobile IPv6 protocol provides two different modes for the communications between *Mobile Node* and *Correspondent Node*. The first, bidirectional tunneling, is similar to the reverse tunneling introduced with Mobile IPv4, where *Mobile Node* and *Correspondent Node* communicate through the Home Agent, and can be used when the *Correspondent Node* does not support Mobile IPv6. The second, called "Route Optimization", introduces the opportunity to establish direct communication between *Correspondent Node* and *Mobile Node* also when the latter is visiting a Foreign Network. This operation leverages the Care-of Address, an IPv6 unicast routable address within the Foreign Network that is automatically assigned to the *Mobile Node* when it reaches the network and that can be used to route packets directly to the *Mobile Node*.

5.7.1 Mobile IPv6 evasion

With respect to mobility-based evasion technique described in Section 5.3, here we describe the specific issues of Mobile IPv6. The Bidirectional Tunneling scheme is almost identical to the Tunneling mode already described, hence there are no substantial differences. Also the Route Optimization scheme is quite similar to the Direct communication case already presented, but it is worth to be described due to some specific details that characterize the transition from Tunneling communication to Direct communication.

In compliance with Mobile IPv6 specifications [82], when the Mobile Node migrates from the Home Network to the Foreign Network, it generates a unique *Care-of Address* belonging to the Foreign Network address space, and transmits it to the Home Agent through a *Binding Update* message. The Mobile Node also challenges the Correspondent Node's ability to support IPv6 mobility, If the Correspondent Node supports Mobile IPv6, then it replies to the *Binding Update* message received from the Mobile Node, and all the following network packets exchanged between the Correspondent Node and the Mobile Node are directly routed between them and they donnot pass through the Home Network. On the other hand, if the Correspondent Node does not support Mobile IPv6, it continues to send packets to the Home Address of the Mobile Node. The packets are received by the Home Agent and tunneled to the Mobile Node in the Foreign Network.

From the attacker's point of view, the messages exchanged to enable Route Optimization, such as the Binding Updates, are really useful in order to detect the migration of the Mobile Node, and they represent a more accurate solution compared to the time based detection cited for Mobile IPv4 in Section 5.6.

Moreover, the time interval between the instant in which the Mobile Node join the Foreign Network and the establishment of the Route Optimization can be used to plan a more complex attack. Indeed, the attacker could split the malicious pay-

load in three different portions and send the first fragment before the migration of the Mobile Node, the second after the migration, but before the Route Optimization has been initialized, and the final fragment after the Route Optimization initialization has been completed. In this case the three packets follow three different paths, which could require an extra work for the deployed NIDS.

5.7.2 Mobile IPv6 cooperation

With respect to the cooperation scheme described in Section 5.4, when the mobility support of IPv6 is used, there are two main characteristics which are worth to be described in detail: the method to detect node mobility and the procedure required to manage the Route Optimization.

Migration detection in IPv6

As briefly introduced earlier, when a Mobile Node leaves its Home Network and joins a Foreign Network, it automatically receives a new address called Care-of Address (CoA). Then, the Mobile Node needs to inform the Home Agent and the Correspondent Node of its new address in order to be reachable, this task is called Binding Update. The Binding Update of the Home Agent is quite simple: the Mobile Node send a special packet (BU) containing its Home Address and its Care-of Address to the Home Agent. Then, the Home Agent replies with a Binding Acknowledge (BUA), confirming the reception of the information. While the latter message of acknowledgement is optional, the former is strictly necessary for the Mobile Node to maintain alive the active connections. Thus the Foreign External Agent can start the process of state migration as soon as it detects the message of Binding Update addressed to the Home Agent. Furthermore, from the same packet it can extract the IP address of the Home Agent that, accordingly to our last assumptions in Section 5.4, corresponds to that of the Home External Agent. Then, the Foreign External Agent can start the State Export process by sending a request to the Home External Agent specifying the Mobile Node's Home Address in order to obtain the related state information.

Route Optimization

As seen at the beginning of this Section, IPv6 provides the Route Optimization scheme which enables direct communications between the Mobile Node and the Correspondent Nodes. However, the Route Optimization can be enabled only if the Correspondent Node supports it and only after the Binding Update has been completed. Until that moment, all the Mobile Node's active connections to and from any Correspondent Node are tunneled through the Home Agent using an

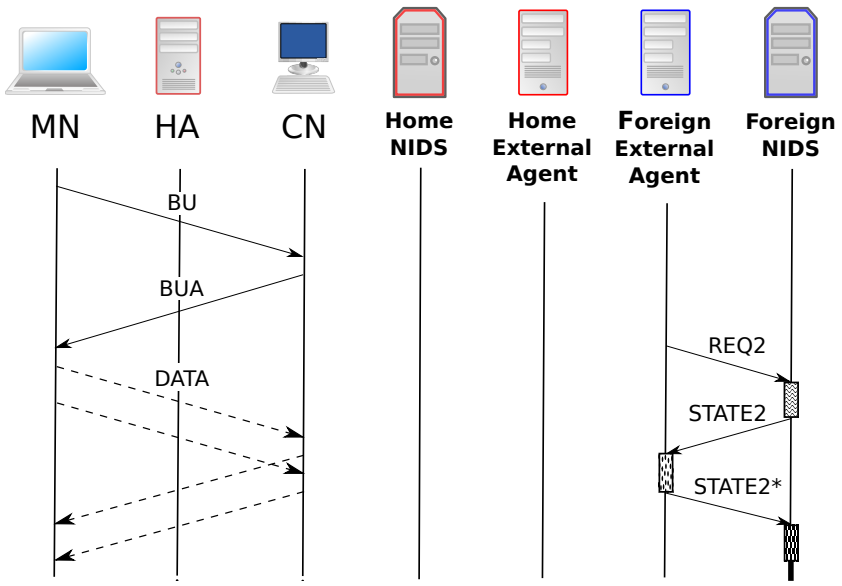

Figure 5.9: Route Optimization sequence diagram: *Mobile Node* and *Correspondent Node* enable Route Optimization

IPv6-in-IPv6 encapsulation. Thus, the Foreign NIDS, which is monitoring the Foreign Network, sees some already active connections between the Mobile Node and its Home Agent and some new connections between the Mobile Node and the Correspondent Node. However, if the NIDS does not explicitly support Mobile IPv6, it is not able to merge together the state information of those connections also if they are actually the same (at least at transport level). For this reason a new scenario is required to manage mobile migrations based on IPv6. The messages exchanged during the state migration after Route Optimization are shown in Figure 5.9.

When the Foreign External Agent detects the BU and BUA messages exchanged between the Mobile Node and the Correspondent Node, it launches the state migration process. The sniffed messages contain several information, in particular the Home Address of the Mobile Node, the address of the Correspondent Node and the address of Home Agent. Then the Foreign External Agent is able to issue an export request (REQ2 in Figure 5.9) to the Foreign NIDS (i.e. the NIDS deployed within its own network), asking for all the state information related to the Home Agent. This state information (STATE2 in Figure 5.9) is pre-processed by the Foreign EA in order to extract only the state information related to the network packets exchanged between the Mobile Node and the Correspondent Node that have just performed Route Optimization. This selection enables the management of connections established with different Correspondent Nodes, which may

or may not support Route Optimization and which in the former case switches to direct communications in different time interval. Finally the pre-processed state information (STATE2* in Figure 5.9) is sent to the Foreign NIDS in order to be re-imported.

This operation of state information pre-processing and re-import allows the Foreign NIDS to analyze all the connections established between the Mobile Node and the Correspondent Node as belonging to the same session (as actually is at transport level). Hence, the Foreign NIDS is able to reassemble a fragmented malicious payload and detect an attack even if portions of it were transmitted before Route Optimization or even while the Mobile Node was still in the Home Network (the latter due to the operations performed during the First Migration scenario in Section 5.4.1).

5.8 Prototype implementation

In order to demonstrate the viability and the effectiveness of the cooperation scheme described in the previous Section 5.4, we developed a prototype implemented in the form of a framework that provides state migration support among geographically distributed NIDSs. This framework consists of three main components:

Patch We modified a small portion of the source code of *Snort*, the most widely deployed open source network intrusion prevention and detection system (IPS/IDS). The patch adds the *state.export* and the *state.import* functions, which, respectively, extract and insert state information related to a specific IP address.

External Agent The External Agent, described in Section 5.4, is a modular software that directly interacts with *SNORT*, invoking the functions introduced through the patch, and with others External Agents. It can be extended thanks to its modularity, thus it can manage different protocols and scenarios and new ones can be added.

Plugins We developed a set of Plugins in charge of managing the specific details of each protocol that provides mobility support. In particular we implemented three different plugins in order to manage WiFi, Mobile IPv4 and Mobile IPv6 environments.

5.8.1 Snort patch

We use Snort 2.8.6.1 (the latest stable version at the moment of development) as base NIDS. The choice of this software is motivated by its open source nature and

its wide adoption, that simplifies the use of the proposed prototype in real world environments.

Snort can be logically divided into multiple components [88]:

- Packet decoder: gathers network packets from a network interface and makes them available for preprocessing.
- Pre-processors: a collection of components and plugins that can be used to analyze or modify network packets before forwarding them to the detection engine.
- Detection engine: an optimized regular expression checker that compares the network traffic with the known attack signatures.
- Logging and alerting system: the layer that generates security alerts and system logs
- Output modules: a collection of components that processes alerts and logs to generate final output.

We modified Snort by enabling each pre-processor plugin to export its own state data and import external state data. Since the state representation used by a pre-processor depends both on the protocol analyzed by the pre-processor and on the pre-processing algorithm, each pre-processor has to implement its own export and import functions. Each pre-processor can then register its import and export functions when it is loaded.

We then added import and export capabilities to the `Stream5` pre-processor. In our implementation, when we export all the state information related to a particular node, we extract the ordered list of network packets that are maintained by the `Stream5` pre-processor and that have the node as one of the communication endpoints. On the other hand, when we import state information, we make Snort analyze the list of packets previously extracted by a Snort instance. Since Snort is able to reorder out-of-sequence network packets, the NIDS that imports the state information is able to merge the new packets with the ordered list of reassembled network packets that have already been analyzed.

Finally, we implemented a *Remote Procedure Call* (RPC) server, using the XML-RPC protocol [109]. The server makes available two remote procedures: `state.export` and `state.import`. A client can request the state information related to one or more hosts by issuing a `state.export` call to Snort, specifying the host's addresses as a parameter. This RPC call triggers the state export functions of all the registered pre-processors, encapsulates the extracted state information in XML format, and sends it back to the caller.

On the other hand, the import of state information is executed when a client issues a `state.import` RPC, passing as parameter a serialized state representation previously exported. The XML-RPC server then unpacks the state representation, and calls the import function of all the pre-processors for which some state information has been exported. Finally, the RPC server replies to the caller with an error code that is used to check whether the import operation has been successful or not.

5.8.2 External Agent module

The External Agent is an autonomous modular software, written from scratch in C language. It has four main tasks:

- detection of the Mobile Nodes migration through the usage of protocol specific plugins;
- cooperation with other External Agents deployed in different networks;
- pre-processing of the state information received from other External Agents;
- import/export of state information into/from the the Snort instance belonging to its network through RPC invocation.

The first task is in charge of the External Agent's plugins, which are described in the next section.

Once a plugin has detected a new Mobile Node, the External Agent starts the state migration procedure contacting the External Agent of the Mobile Node Home Network. The communication between the two External Agents is based on RPC, in particular each External Agent embeds a XML-RPC server similar to that added to SNORT and described in the previous section. This solution enables the External Agent to contact indifferently a NIDS or another EA, thus simplifying both the implementation and the communication scheme.

When the External Agent needs the state information of a Mobile Node arrived from a different Network, it issues a `state.export` request to the External Agent deployed in the Mobile Node Origin Network. This EA analyzes the received request in order to verify the syntactic correctness and its consistency (i.e. the cited Node has really left the Home Network). There are also different security checks that can be applied, such as limit the request rate or control the trustiness of the External Agent that issued the request. If the request passes all the checks, the External Agent forwards it to the local NIDS and sends back the received results to the External Agent that requested the state information.

When the External Agent receives the state information related to the Mobile Node, it pre-processes the content before importing it into the NIDS deployed

within its network. The pre-process steps required before the import may vary mainly depending on three factors: the scenario (considering those described in the previous sections 5.4), the protocol that provides mobility support and the differences between the NIDS that exported the state information and the NIDS that has to import it[2]. While the latter may be performed by the External Agent itself, the former require interaction at the protocol layer that is providing mobility, thus the pre-processing work is performed by the plugins.

Finally, once the pre-processing steps are finished, the External Agent can import the state information into the NIDS, using the `state.import` method provided by the modified version of Snort.

5.8.3 Plugins

As introduced earlier, for each protocol supported by the External Agent, a specific plugin is required in order to manage the detection of new mobile nodes and the pre-processing of related state information before the `state.import` invocation. In the following subsections we analyze the details related to the plugins which enable the support to Wi-Fi, MIPv4 and MIPv6.

WiFi plugin implementation

With respect to the cooperation scheme described in Section 5.5, we focus on the implementation of a plugin for the scenario of RADIUS based authentication with different credentials for each Mobile Node.

Mobility Detection

For the detection of new Mobile Node, the plugin has to sniff the network traffic waiting for two specific RADIUS packets: *Access Request* and *Access Accept* response. The former is used by the External Agent to detect the arrival of a new Mobile Node in the network, while the latter provides the confirmation. Usually the *Access Request* contain the MAC address of the MN in the *Calling-Station-ID* attribute. In this case, the External Agent can retrieve the Mobile Node IP address simply with an ARP request. If that attribute is not present, the External Agent can extract the username used for the authentication and then use it as keyword to retrieve information related to the Home Network of the Mobile Node. In both cases, in order to provide to the External Agents the information related to the last authentication of the Mobile Node, it is necessary to deploy a new component near the RADIUS server (i.e an External Agent companion) in charge of registering all

[2]Due to our focus on Snort, our prototype only checks whether the received data is compatible with the NIDS which has to receive it or not.

the successful authentication requests. In particular, it is necessary to register the information related to the Mobile Node (username and IP address) and the identifier of the associated AP, which is reported in the *NAS-IP-Address* attribute or a *NAS-Identifier* attribute of the *Access Request*.

Pre-processing

Moving from one AP to another, the Mobile Node maintains the same IP address, then the received state information does not require any kind of manipulation. The unique pre-processing step necessary before importing the state information is to analyze the received data in order to verify the correctness of the format.

MIPv4 plugin implementation

Mobility Detection

For the detection of new Mobile Nodes, the plugin has to sniff the network traffic waiting for two specific UDP packets: *Registration Request* and *Registration Reply*. From the former packet the plugin can extrapolate the Home Address, the Home Agent address and the Care-of Address, while the latter is analyzed in order to understand if the registration succeed or failed. In the first case the plugin forwards to the External Agent the information gathered from the *Registration Request* packet, while in the second case it waits for another registration attempt.

Pre-processing

The state information received by the External Agent require to be preprocessed before the submission to the NIDS. The preprocessing required by the data depends on the scenarios presented in Section 5.4: first migration, return to home and further migration, and is described in the following:

First Migration The Foreign External Agent issues a request for the state information related to the Home Address of the Mobile Node. From the received data, it extracts all the packets, adds an extra IP header to simulate an IP in IP tunnel between the Mobile Node and the Home Agent, and rebuilds an XML representation ready to be imported. Once the pre-processed state information has been imported into the NIDS, the tunneling communication used by the Mobile Node to communicate with Correspondent Nodes appears to the NIDS as the continuous of the already existent connections.

Return to Home The Home External Agent issues a request for the state information concerning the care-of address used by the Mobile Node during the

stay in the Foreign Network. Then, Home External Agent extracts all the network packets from the received data and removes all the outer IP header added by the Home Agent to perform IP in IP tunneling. Finally it rebuilds an XML representation and imports it into the NIDS.

Further Roaming It is the most complex scenario of the three presented in section 5.4. When the Destination External Agent detects a new Mobile Node sniffing the Registration Request and Registration Reply packets, it obtains information only about the Home Network and nothing about the Origin Network. Thus the Destination External Agent issues a `state.export` request to the Home External Agent. When the latter analyzes the request and checks its consistency, it notices that the Mobile Node was already in a Foreign Network, then forwards the request to the Origin External Agent. The received reply is directly forwarded by the Home External Agent to the Destination External Agent, which imports the pre-processed data into the NIDS. In this case the pre-processing steps consists in two tasks: the first is to remove all the outer IP header used for the IP in IP tunneling, the second step is to add an extra IP header to simulate an IP in IP tunnel between the Mobile Node and the Home Agent. Fortunately these tasks may be reduced into a single operation of substitution performed on the field containing the Mobile Node's Care-of address. In particular the old Care-of address is replaced by the new Care-of address. The results is the same described for the First Migration: the NIDS sees a unique session between the Mobile Node and its Home Agent.

MIPv6 plugin implementation

Mobility Detection

For the detection of new Mobile Nodes, the plugin has to sniff the network traffic waiting for the *Binding Update* and *Binding Acknowledge*, which are two packets containing all the information required by the External Agent. These packets are characterized by a Mobility Header that is an IPv6 extension header used by mobile nodes.

Pre-processing

The pre-processing required by the data depends on four different scenarios: first migration, return to home, and further migration, presented in Section 5.4, and Route Optimization, described in Section 5.7.

First Migration The Foreign External Agent issues a request for the state information related to the Home Address of the Mobile Node. Once the infor-

mation has been received, it extracts all the packets from the received data and adds an extra IPv6 header in order to simulate an IPv6-in-IPv6 tunnel between the Mobile Node and the Home Agent. Then, the Foreign External Agent rebuilts an XML representation of the state information and imports it into the Foreign NIDS inovking the `state.import` method added to the NIDS trough the patch. Once the pre-processed state information has been imported into the NIDS, the tunneling communication used by the Mobile Node to communicate with Correspondent Nodes appears to the NIDS as the continuous of the already existent connections.

Route Optimization During the Route Optimization phase (Section 5.7.2) the Mobile Node does not migrate to a different network and is monitored by the same NIDS, then the Foreign External Agent directly invokes both the `state.export` and `state.import` methods provided by the local NIDS. From the received data it extracts all the network packets and checks whether they belong to an IPv6-in-IPv6 tunnel or not. This control is intended to distinguish the packets belonging to communications initiated before the migration, from those belonging to sessions started after the migration. While the latter kind does not require any pre-processing operation and can be discarded, the former group of packets requires to be pre-processed by the Foreign External Agent in order to remove the outer IPv6 header. Then, the Foreign External Agent has to substitute all the occurrences of the Home Address in the inner IPv6 header with the Mobile Node *Care-of Address*. Finally the Foreign External Agent rebuilds an XML state representation which is imported into the local NIDS.

Return to Home When the Mobile Node return to the Home Network (Section 5.4.2), the Home External Agent requests to the Foreign External Agent the state information related to the Care-of Address of the Mobile Node. Depending on whether the Route Optimization was enabled or not, there are two different preprocessing steps required by the state information received by the Home External Agent. In the former case the Home External Agent has to substitute all the occurrences of the *Care-of Address* in the IPv6 header with the Home Address of the Mobile Node. In the latter case, the Home External Agent simply removes the outer IPv6 header previously added by the Home Agent to perform IPv6-in-IPv6 tunneling. Then, the state information can be imported into the Home NIDS.

Further Migration In the Further Migration phase (Section 5.4.3) the Destination External Agent receives the requested state information (STATE4 in Figure 5.8) and extracts all the network packets. If the Mobile Node was not using Route Optimization in the Origin Network, then the Destination

External Agent has to substitute all the occurrences of the Origin *Care-of Address* in the outer IPv6 header with the Destination *Care-of Address*. If Route Optimization was enabled, the Destination External Agent adds an extra IPv6 header to simulate a IPv6-in-IPv6 tunnel between the Mobile Node and the Home Agent. The pre-processed packets are then used to rebuild an XML state representation that is imported by the Destination NIDS.

5.9 Validation and performance evaluation

In this section we describe our testbed, the experimental validation for different scenarios of the prototype described in Section 5.8 and then the performance evalution.

5.9.1 Testbed

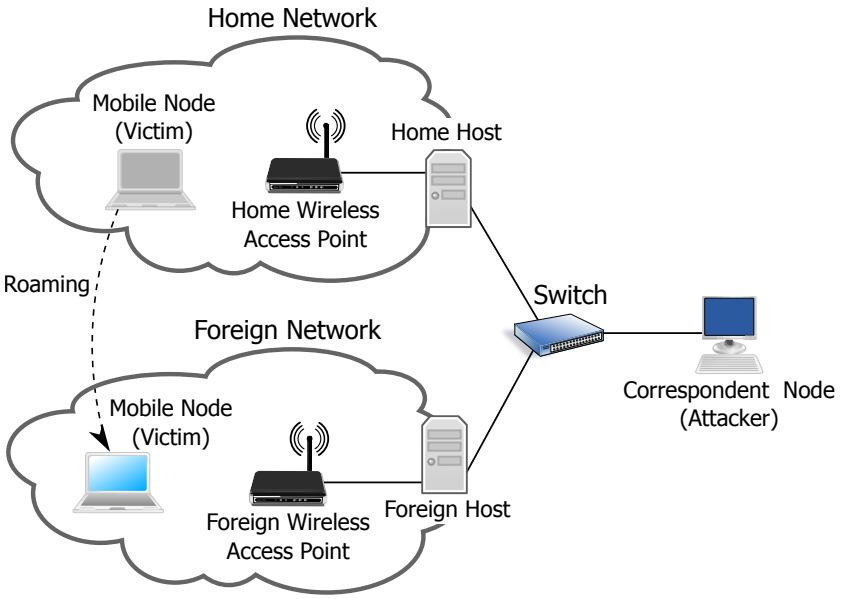

Figure 5.10: Experimental testbed

Our experimental testbed is represented in Figure 5.10. The two clouds on the left represent two networks, the Home Network and the Foreign Network, which provide mobility support based either on Wi-Fi, MIPv4 or MIPv6. Each network is monitored by the Home Host and the Foreign Host respectively, which, for simplicity, execute both the modified version of Snort described in Section 5.8.1 and an instance of our External Agent described in Section 5.8.2. Hence, the machine

named Home Host acts as Home NIDS and Home External Agent, while the Foreign Host acts as Foreign NIDS and Foreign External Agent. These machines are equipped with an Intel(R) Core2 6600 with 2GB RAM and an AMD Athlon(tm) 64 Processor 3000+ and 1 GB RAM, respectively, and are directly connected to a Cisco Aironet 1100 access point that provides wireless connectivity to mobile nodes. The Mobile Node is a laptop equipped with a wireless network interface, while the Correspondent Node is a fixed host. All the machines are GNU/Linux hosts with the kernel (version 2.6.35 or 2.6.39) compiled with the modules necessary to support mobility. In addition, to enable the support to Mobile IPv4, the Mobile Node, the Home Host and the Foreign Host run the *Dynamics Mobile IP* [46] daemon. While to enable the support to Mobile IPv6 and Route Optimization the Home Host, the Mobile Node and the Correspondent Node run the *mip6d* [16] daemon, and the Home Host and Foreign Host run the *radvd* [93] daemon.

5.9.2 Validation

In order to validate our prototype, we executed several experiments covering all the possible scenarios resulting from the combination of the roles described in Section 5.3, the roaming events considered in Section 5.4 and the different protocols supported by the plugins analyzed in Section 5.8.3. We now describe the standard approach adopted to execute these experiments. For each experiment we verify the capability of the Mobile Node to roam from the Home Network to the Foreign Network and viceversa, keeping alive a connection with the Corresponding Node. We also tested the ability of the two nodes to enable the Route Optimization when using Mobile IPv6. To validate our prototype we selected a suspicious sequence of bytes from the signatures recognized by snort and we use it to test the ability of the Home NIDS and the Foreign NIDS to detect the simulated attack.

Wireless LAN

In the WLAN scenario, we deployed a FreeRADIUS [52] server on a machine reachable by both the Home Host and the Foreign Host. All the involved machines belong to the same network (10.11.13.0/24). The two Access Points are connected to the Home Host and the Foreign Host, advertise the same essid and are configured to use the RADIUS server to manage the authentication of the Mobile Nodes. When the Mobile Node, which represents the Mobile Victim, tries to associate to the Access Point connected to the Home Host (AP1), it has to insert a username and a password. The AP1 prepares an *Access-Request* packet filled with the credentials and send it to the FreeRADIUS server, which replies with

an *Access-Accept* packet (if the credentials are correct). Once the Mobile Node is associated to the AP1, we create a TCP connection between it and the Correspondent Node and start to transmit some packets to check the stability of the communication channel.

For the Correspondent Node, which represents the Fixed Attacker, could be quite difficult, if not impossible, to detect the roaming of the Mobile Node from an AP to another, only by analyzing the network traffic. However, if it is near to the Mobile Node or near to the AP1, it could configure a wireless NIC in Monitor mode and analyze the 802.11 traffic looking for Association/Deassociation (or Auth/Deauth) packets. Once the two nodes have established a connection, we use the attacker to send the first portion of the attack, then we wait for the roaming events. To lead the Mobile Node to dissociate from the AP1 and associate to the AP2, we dynamically configured the transmission power of the two appliances through their internal commands and enforced the attenuation of the AP1's signal with an aluminum enclosure. From the Mobile Node point of view this is the same as a movement towards the AP2, then the re-association process is natural and is not forced with external commands. By receiving the *Access-Accept* packet sent by the RADIUS servers, the Mobile Node completes the re-association process and is newly available to receive packets from the Correspondent Node. The same packet is sniffed by the Foreign External Agent that retrieves [3] the necessary information related to the Mobile Node and contacts the Home External Agent to obtain the state information from the Home NIDS. In parallel with this automatic process, we send the last portion of the attack to the Mobile Node using the same connection established with the CN before the roaming process.

We perform this experiment at least two times, with and without the two External Agents, in order to verify the effectiveness of our solution. While in the second experiment none of the two NIDS detect any kind of attack, in the former the Foreign NIDS obtains the state information from the Home NIDS, which contains the first portion of the attack. Then, by reassembling this information with its internal state, the Foreign NIDS is able to reconstruct the complete session and to detect the attack.

We obtain the same results also with different roaming scenarios and inverting the roles of the two nodes (Mobile Attacker and Fixed Victim). This means that the cooperation scheme proposed in Section 5.4 and the implemented prototype are effective against mobility-based NIDS evasion.

[3] To simplify the testbed and the validation experiments, we provided the Foreign (Home) External Agents with the association between the IP address of the MN and the IP address of the Home (Foreign) External Agent. This avoided us to deploy a further element in charge of register the successful authentication of the Mobile Nodes

Mobile IPv4

In the Mobile IPv4 scenario, we installed the *Dynamics Mobile IP* [46] daemon on the Mobile Node, the Home Host and the Foreign Host. The former two represent the Home Agent and the Foreign Agent required by Mobile IPv4 and are deployed in different networks. They provide wireless connectivity trough an Access Point connected to them, in particular the Home Agent is deployed within the Home Network and its AP broadcasts the essid *Home*, while the Foreign Agent is deployed in the Foreign Network and its AP broadcasts the essid *Foreign*. The Mobile Node, which represents the Mobile Victim, is configured with a static IP address (192.168.1.101) belonging to the Home Network domain (192.168.1.0/24) and is connected to the same network through the *Home* WiFi network provided by the Access Point. We establish a connection with the Correspondent Node, which is the attacker, and transmit some packets to test the stability of the connection. Then, the Correspondent Node sends the first portion of the attack and at the same time it starts pinging the Mobile Node. This measure is used by the Attacker to detect the roaming of the Victim to a new network.

To start the roaming process, we command to the Mobile Node to connect to the *Foreign* WiFi network, by using its network manager. This represents a real scenario where a user decide to connect to a new network by choosing one of the available *essids*. Once the Mobile Node is associated to the *Foreign* AP, it receives the ICMP Router Discovery Protocol (IRDP) packets sent by the Foreign Agent. These packets are sent by the Home and Foreign Agent to advise the Mobile Nodes about their availability. Receiving these packets the Mobile Node detects that it has moved to a Foreign Network (192.168.2.0/24), then it starts the registration procedure by sending a Registration Request (RRQ) to the Home Agent, through the Foreign Agent, informing about its Care-of Address (192.168.2.101). The Home Agent replies by sending a Registration Reply (RRP), through the Foreign Agent, containing the result of the registration, in particular 0 or 1 in case of successful registration [4]. When the registration has been completed, the Attacker can measure a variation in the ping timings due to the new routing scheme of the packets exchanged between the Mobile Node and the Correspondent Node. Then the Attacker sends the last portion of the malicious payload in order to complete the attack.

Running this experiment without enabling the External Agents allows the Attacker to perform a stealth attack because the first portion of the attack is received only by the Home NIDS while the Foreign NIDS receives only the second portion of the malicious payload, then it is not able to detect the attack. This result can be subverted by enabling the two External Agents deployed in the Home and Foreign

[4] Actually the RRQ and RRP contain also the lifetime, the identification number and other information, that we omitted because they are not considered by the External Agents.

network. Indeed, when the Mobile Node joins the Foreign Network and starts the registration process, the Foreign External Agent detects the new arrival and sends a `state.export` request to the Home External Agent asking for the Home Address of the Mobile Node extracted from the RRQ packet. When the Foreign External Agents receives the state information, it processes the data by adding an external header to each packet simulating an IP-in-IP tunnel between the Mobile Node and the Home Agent, then it insert the information into the Foreign NIDS. The latter is able to process out-of-order packets, then it eventually detects the attack as soon as it receives both the portions of the malicious payload sent by the attacker. Similar experiments have been performed different times targeting all the scenarios (i.e. Mobile Attacker and Both Mobile Nodes) and the roaming events (i.e. Return to Home and Further Migration), where the main differences are the pre-processing steps performed by the External Agent before importing the received state information into the local NIDS. The results confirm that by adopting our prototype it is possible to detect the mobility-based stealth attacks perpetrated exploiting Mobile IPv4 networks.

Mobile IPv6

In the Mobile IPv6 scenario, we run the *mip6d* [16] daemon on all the machines, but the Foreign Host, in order to enable Route Optimization, while only on the Home Host and Foreign Host we installed the *Router Advertisement Daemon* (radvd), which sends Router Advertisement messages, which advertise routing prefixes, and provides support to *stateless autoconfiguration*, a method which enables IP address autoconfiguration of the host connected to the network without requiring the deployment of a DHCP server. We replicated the wireless infrastructure used to test Mobile IPv4, then we connect the Mobile Node, which is the Victim, to the *Home* WiFi network, and its Home Address (2001:db8::beef) belongs to the Home Network's address space (2001:db8::/64). We establish a connection to the Correspondent Node, which is the attacker, and transmit some packets to test the stability of the connection. Then the Correspondent Node sends the first portion of the attack.

As seen in the previous experiment, we command the Mobile Node to connect to the *Foreign* WiFi network, by using its network manager. Once the Mobile Node is associated to the *Foreign* AP, it receives the Router Advertisements messages which notify the Foreign Network prefix (2001:db8:1::/64). The Mobile Node create a Care-of-Address starting from its MAC address, the obtained IP address (2001:db8:1:0:218:deff:fe25:599) belongs to the Foreign Network's addresses and can be used to communicate with other nodes external to the Foreign Network. Then the Mobile Node starts the Binding Update procedure by sending Binding Updates messages to the Home Agent and, only after completing the pro-

cedure involving the HA, to the Correspondent Nodes. These messages are used by the Mobile Node to advertise the Home Agent and the Correspondent Nodes about its new position. While the concept behind the BU sent to the Home Agent is similar to the registration of the Mobile Node described for Mobile IPv4, the BUs sent to the Correspondent Nodes are used to enable the Route Optimization, the direct communication scheme described in Section 5.7.

When the Correspondent Node receives the first BU message, it understands that the Mobile Node has migrated to a new network, thus it sends the second portion of the attack, which reaches the Mobile Node in the Foreign Network passing through the Home Agent. These steps allow the Attacker to complete the attack as soon as possible and avoiding the detection by the Foreign NIDS, because it does not receive the first portion of the malicious payload, then it is not able to match the related signature. At the same time, the Home NIDS receives and analyzes both the portions of the attack, thus it is able to detect it and eventually rise an alert, but it can poorly interact with the Mobile Node, because it is within the Foreign Network. The Attacker can avoid the detection by the Home NIDS, sending the last portion to the Mobile Node only after the Binding Update procedure has been completed. This does not allow both the Home NIDS and the Foreign NIDS to receive the complete payload, thus the Attacker can perform a completely stealth attack.

By deploying and enabling our proposed framework it is possible to thwart this attack independently of the usage of the Route Optimization. Indeed, by using the plugin developed to support Mobile IPv6, the Foreign External Agent is able to detect the arrival of a new Mobile Node by sniffing the BU messages directed to the Home Agent, from which it extracts the Home Address, the Home Agent address and the Care-of Address. Then it sends a `state.export` request to the Home External Agent, asking for the state information related to the Home Address. The received data is preprocessed encapsulating each packet with an external header which simulates an IPv6-in-IPv6 tunnel between the Mobile Node (Care-of Address) and the Home Agent (Home Address), then the modified information is imported in the local NIDS. These operations allow the Foreign NIDS to analyze the complete malicious payload, hence to detect the attack.

5.9.3 Performance evaluation

In order to evaluate the performance of our prototype we selected one environment, Mobile IPv6, and analyzed two different aspects: the timings of the state migration procedure and the impact of the migrations on the NIDS performance.

Timings

We measured the time required by the state migration process in several experiments.

In our experiments we separate the measures of the time required by Snort to execute the `state.export` and the `state.import`, from the time required to complete the state migration process, which includes the network delays. These measures are performed by the External Agents, in particular, we modified the XML-RPC client by adding a measurement of the time elapsed between the request sent to Snort and the reception of the reply containing the state information, in case of `state.export`; and the time elapsed from when the External Agent sends the state information to Snort to when it receives a confirmation of reception, in case of `state.import`. We repeated several times the test used to validate the Mobile IPv6 plugin, varying the network conditions. In particular, we generated synthetic network traces characterized by a different number of simultaneously active TCP connections, a factor which has a direct impact on the resource consumption of the two NIDSs monitoring the networks. Indeed, for each TCP connection monitored by the NIDS, it has to create and maintain a dedicated session. The measures registered during these tests are plotted in Figure 5.11.

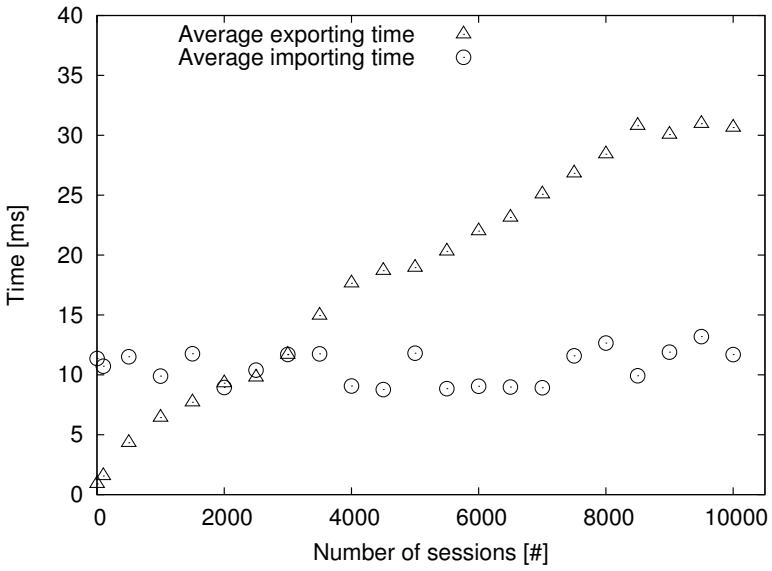

Figure 5.11: State importing and exporting times per session

The X-axis represents the number of concurrent TCP connections characterizing the network traces monitored by Snort, while the Y-axis represents the time, expressed in milliseconds. Triangles and circles represent the duration of

`state.export` and `state.import` operations respectively, which are performed with different numbers of concurrent connections. In particular, each point in the graph represents the average computed over five different measures.

As shown in Figure 5.11, the time required to import the state of the Mobile Node in the Foreign NIDS is independent of the number of concurrent connections and it is as low as less than 14 milliseconds. Instead, the time required to export the state of the Mobile Node from the Home NIDS increases linearly with respect to the number of active TCP connections until Snort reaches the limit of concurrent connections tracked by `Stream5`, which is 8192 by default. After that, the state export time remains constant. These results are consistent with the internal architecture of Snort. Indeed, during the import operation the received data is appended to the existing state information, while the export operation requires a sequential scan of all the state information until a match is found.

We also measured the time required to complete the state migration process, defined as the time elapsed between the `state.export` request sent to the Home External Agent by the Foreign External Agent, and the receiving of the result of the `state.import` operation from the Foreign NIDS.

Table 5.1: Times required by state migration activities

	Average [ms]	σ [ms]	Peak [ms]
State import	12	1	13
State export (worst case)	30	1	31
Complete state migration	409	176	765
Network roaming	8835	3495	13209

All the experimental results are summarized in Table 5.1. The average state migration time is 409 milliseconds and it is dominated by network delays. We highlight that this time value is one order of magnitude lower than the time required by the Mobile Node to complete the roaming from the Home to the Foreign Networks (on average 8.835 seconds in our experimental testbed). These results and the ability of our prototype to handle out-of-order network packets [5] demonstrate that our solution is compatible with real time analysis of live network traffic.

Performance comparisons

In order to evaluate the impact of our patch and the presence of the External Agent on the performance of Snort, we run several experiments focusing on the capabil-

[5]The ability to handle out-of-order network packets is inherited from Snort and improves tolerance to network delays

ity of the NIDS in terms of maximum bandwidth which the NIDS can handle maintaining a low packet loss rate. When testing Network IDS, the choice of the network traffic to analyze is not simple and there are no standard traces unanimously accepted, since the IDEVAL set [66], that had been released exactly for this purpose, is widely considered deprecated. Thus, in our experiments we used network traffic that had been captured during a Capture-the-Flag (CTF) Hacking Contest held in 2010 [38], which is recent, publicly available and contains several attacks.

In the first experiment we focus on comparing the performance of the original Snort against our framework, which includes a modified version of Snort and the External Agent, and that we run on the same machines. We configured Snort adopting the default configuration, then we replayed the registered network traces at different speeds ranging from few Mb/s up to 160Mb/s using *tcpreplay* [8].

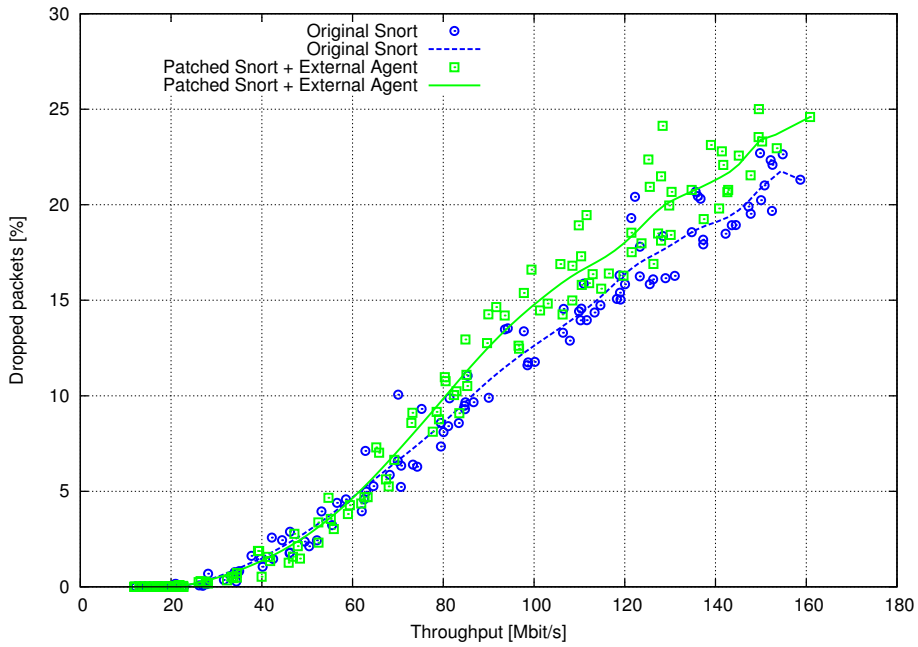

Figure 5.12: Comparison between the original Snort and our framework (patched version of Snort + External Agent), measuring the percentage of dropped packets per throughput.

During the tests, we measured the actual average throughput generated by *tcpreplay* and the number of dropped packets registered by Snort. The results are plotted in Figure 5.12. As can be seen, on our hardware, in both the scenarios Snort is able to process 100% of packets up to 20-25 Mb/s, then it start to drop packets. The behavior of the Original Snort and Patched Snort is the same up to

70-80 Mb/s and the results of the two versions are overlapped reaching about 10% of dropped packets. Then, increasing the throughput of the network traces up to 160Mb/s, the Patched Snort shows a slightly but clear performance degradation reaching about 25% of dropped packets versus the 22% of the Original Snort. However, both these values are not suitable for production environment because they are too high. This means that in the range of throughputs manageable by Snort, our framework offers quite a similar performance. We also remark that despite the External Agent can be installed on a separate machine, we run both Snort and the External Agent on the same host. While this choice may bias the results in terms of negative impact on resource consumption, it guarantees that our results are conservatives and realistic.

The goal of the second set of experiments is to determine the impact of the state migration processes on the performance of our framework. For the tests, in addition to the traffic traces used in the previous experiment, we prepared several traces simulating the migration of different Mobile Nodes. In particular these traces contain some TCP packets exchanged between the Mobile Node and the Correspondent Node, the Binding Update packets used by Mobile IPv6 and finally the *state.export* request sent by the Foreign External Agent to the Home External Agent. We focused on the *state.export* mechanism due to its heavier impact on the resources of Snort than the *state.import*, as shown in the timings Subsection 5.9.3.

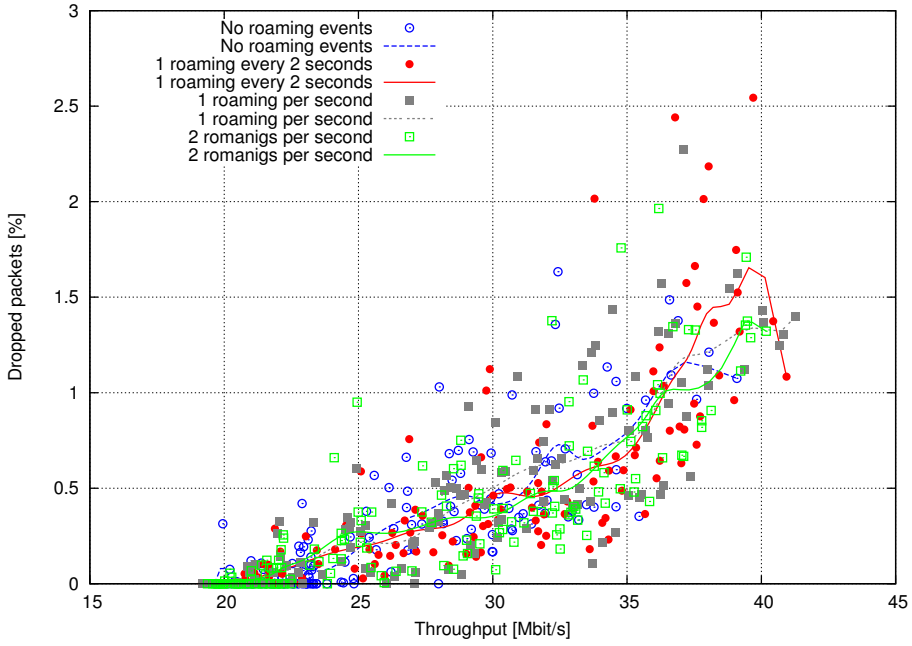

Figure 5.13: Percentage of dropped packets per throughput measured for different frequency of roaming events, from zero to two per second

We replayed the registered network traffic at different speeds ranging from about 20 to 40Mb/s, a small range focused on the throughput during which Snort starts to drop packets, then we replayed the synthetic network traces simulating Mobile Nodes roamings at regular intervals: 0.5, 1 and 2 seconds. As in the previous experiment, we measured the dropped packets registered by Snort and the actual average throughput generated by *tcpreplay*. The measurements are plotted in Figure 5.13. As can be seen, there is not a clear difference between the experiments with different roaming intervals. This result reflects the low impact of the state migration on the performance of Snort. In particular, the *state.export* operation keeps Snort busy for 10-30 milliseconds, a limited time period during which the received packets fill the buffer. As soon the operation ends, Snort restarts to process events and it frees the buffer prior to the next roaming.

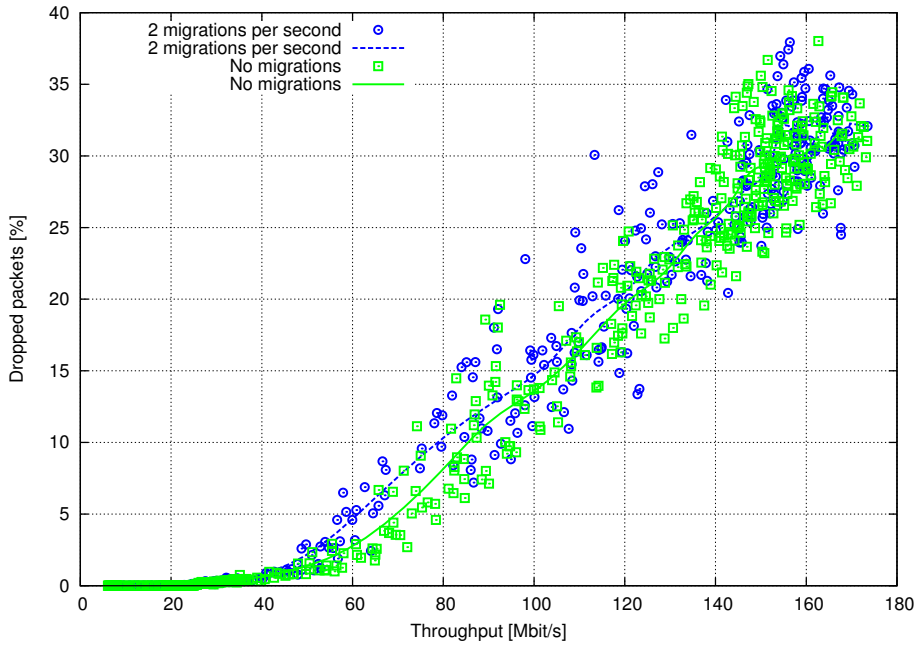

Figure 5.14: Impact of migrations on the performance of our framework. Analysis on a wide range of throughputs.

The goal of the last set of experiments is to verify if the impact of state migration process could become more distinguishable at higher bitrates. We compared the performance of our framework in case of two migrations per second (time interval between migration of 0.5s) and in case of no migration at all, considering a range from about 10 Mb/s up to 170 Mb/s. The results are presented in Figure 5.14 while Figure 5.15 is a zoom on the range 15-35 Mb/s. While the average results obtained in the two scenarios are comparable and really close to each other,

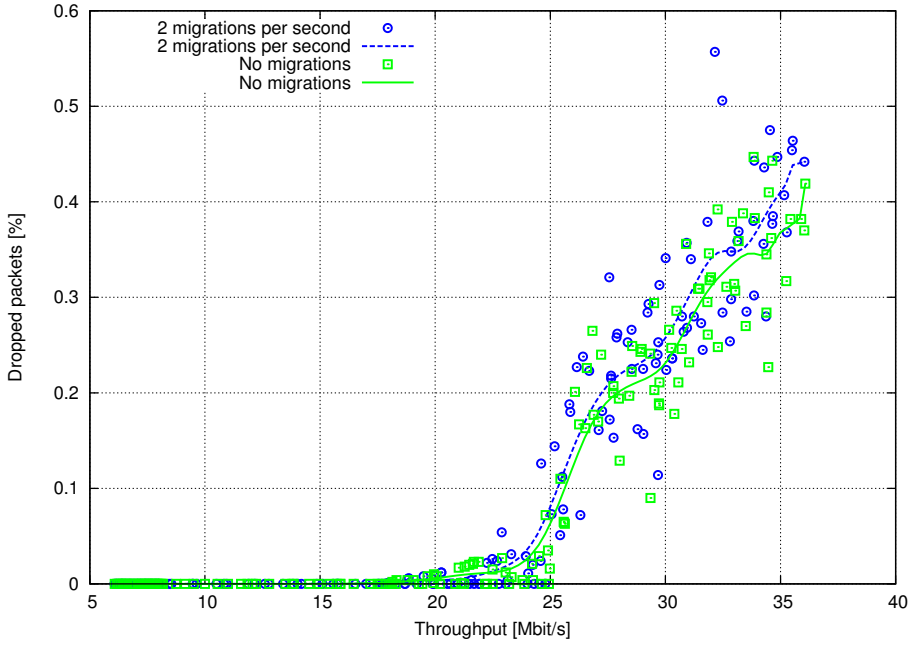

Figure 5.15: Impact of migrations on the performance of our framework. Focus on manageable throughputs.

the impact of the state migration process emerges by the grater dispersion of its measures with respect to those taken without state migration.

These results clearly confirm the effectiveness and the validity of our framework, which offers almost the same performance of the original Snort and manages node mobility with a negligible impact on the resources.

5.10 Concluding remarks

In this chapter we described a new NIDS evasion strategy that allows an attacker to evade detection in all network environments that support node mobility. In particular, we highlight that this new evasion strategy, called mobility-based NIDS evasion technique, leverages seamless mobility and targets roaming events, while it does not exploit neither flaws in a NIDS implementation, nor protocol specific vulnerabilities.

In the second portion of the chapter, we proposed the first NIDS cooperation scheme that can be used to thwart mobility-based NIDS evasion in all mobile environments. The viability of the proposed solution is demonstrated through a prototype implementation, whose performance are compatible with real-time

analysis of network traffic. This cooperation scheme is immediately applicable to real networks because it does not require neither the modification of the protocols that provide mobility nor mobile node software. Furthermore, the modular design of the proposed framework and its integration with Snort simplify its deployment within production environment already monitored by the most widely deployed open-source IDS.

Chapter 6

Conclusions

The complexity of modern network architectures, the widespread diffusion of self replicating malware and the increasing number of large scale attacks pose new challenges for administrators in charge of defending IT infrastructures. Cooperation among different architectures is emerging as the most valid solution to face modern threats coming from multiple sources. In this thesis we present novel collaborative architectures for intrusion detection that aim to improve existing security solutions. We consider three different scenarios characterized by particular requirements and constraints.

Large networked systems. Administrators of large organizations have to manage complex IT infrastructures consisting of multiple departments and heterogeneous networks. Monitoring of this kind of environments requires the deployment of several NIDS distributed in each network segment. However, the huge amount of alerts generated by distributed NIDS forces administrators to spend a lot of time to discover the real threats. On the other hand, splitting the task among different departments could introduce relevant delays that affect reaction time.

Cooperation among different organizations. Helping different organizations to collaborate in order to thwart new emerging threats presents several issues that require particular attention. Each organization has to work autonomously and the collaboration must not affect its daily activities. Moreover, among the cooperating organizations, two or more of them could be competitors, thus they could be reluctant on sharing any kind of information, or they could submit misleading data in order to directly damage their adversary and eventually degrade the overall effectiveness of the whole cooperation.

Mobile environments. The number of mobile nodes connected to Internet is constantly increasing, but not all defensive solutions are ready to manage this kind of user, due to limited support to mobile protocols or incomplete coverage of the network area used by the mobile nodes. However, relevance and impact of mobile devices on business activities can not be ignored for long. Thus, mobile environments require design and deployment of effective solutions as soon as possible, in order to protect and secure mobile users and visited networks.

In the first scenario, we focus on helping administrators to manage the huge amount of irrelevant alerts generated by distributed NIDS deployed in multiple departments, which overwhelms the critical alerts related to real security threats. We propose a novel distributed alert ranking scheme. It integrates different information gathered from multiple sources, ranging from security advisory to system configuration of monitored machines. Then, the alert ranking server analyzes NIDS alerts and ranks them in order to provide the department administrators with a clear distinction between relevant alerts, which are those related to attacks that succeeded with a high probability, and false positives, which are alerts related to attacks targeting specific vulnerabilities that are not present on attacked machines. The proposed solution is based on a hybrid architecture that leverages both hierarchical scheme, for intra-department communications, and a peer-to-peer overlay network for inter-department cooperation. The latter is also improved by a privacy preserving alert sharing service that allows department administrators to configure dissemination rules in order to propagate alerts related to critical components avoiding disclosure of sensitive information. The proposed architecture has been implemented and validated in a controlled environment, demonstrating the viability of our solution.

With respect to cooperation among different departments, the collaboration of multiple organization poses different constraints, in particular there is a strong need of filtering submitted information and processing large number of events. We propose a collaborative architecture based on several distributed Event Sources and a Processing Container that leverages Complex Event Processing and a Distributed Hash Table overlay to analyze a huge amount of data. The former component is deployed within each organization and, thanks to its multilayer architecture, is able to gather events from different sources, perform several preprocessing steps, such as filtering, anonymization and normalization, and submitting normalized events to the Processing Container. The latter is composed by several processing elements that share different information through a robust peer-to-peer overlay and analyze received data through Complex Event Processing obtaining important alerts. Our proposal allows organizations to participate on a voluntary base, guaranteeing fine grain control on shared information, and supports differ-

ent processing algorithms and different CEP engines in order to provide tailored defensive solutions. In particular, we validated the effectiveness of our solution against Man-in-the-Middle attacks. We deployed the Processing Container leveraging several Virtual Machines that analyzed events received from geographically distributed Event Sources (i.e. Italy, Hungary and Israel) and detected the IP addresses of the attackers among a large number of events.

We, finally, focus on mobile environments, which are characterized by the presence of mobile nodes able to move among different networks, or network segments. We present a new attack strategy, called mobility-based NIDS evasion technique, which allows an attacker to leverage roaming events in order to perform "stealth" attacks, that are not detectable even by state-of-the-art stateful NIDS. We describe the characteristics of this new evasion strategies and how it can be exploited by malicious users, detailing the differences for multiple attack scenarios. In particular, we consider mobile attackers, mobile victims and mobile attackers and victims. We then propose a cooperation scheme based on information sharing among distributed NIDS monitoring different networks. The aim of our solution is to allow distributed NIDS to share internal state on the base of mobile node migration, in order to provide NIDS monitoring these nodes with the information related to their previous network activities. We also describe the specific details related to three different protocols supporting mobility: Wi-Fi, Mobile IPv4 and Mobile IPv6. For each protocol we explain the attack steps and the necessary countermeasures. In order to thwart this new threat, we designed a modular framework that enhances Snort capabilities and allows different NIDS to share their internal state information in order to detect also mobility based attacks. We validate our proposal in a real testbed performing migrations among different networks with a laptop while attacking or being attacked from another notebook. All the attacks have been detected by our framework thanks to the information sharing between the deployed NIDS. We also present an analysis of the impact of our proposal on the performance of a standard Snort, and the impact of the migrations on the monitoring capabilities of the deployed NIDS.

In this thesis we proposed different defensive solutions and presented several interesting results, however network security threats are continuously evolving and increasing. Furthermore, while criminal organizations are the main actors behind the vast majority of the attacks, hence the main investors, also governments started to spend resources on military programs aimed to improve cyber-warfare capabilities. In this scenario, any interconnected device may represent a valuable target, ranging from modern tablet PC to old-style desktops, from new full-featured smartphones to old basic SCADA networks. Existing defensive solutions are not effective against these new threats and our proposals represent a clear

improvement of the state-of-the-art. In particular, cooperation among different organizations is drawing attention of several companies aimed to offer large scale security solutions, while low level information sharing used to monitor mobile environments could represent a standard feature for future NIDS.

Bibliography

[1] The bro network security monitor. http://bro-ids.org/.

[2] Ossec - open source security, host-based intrusion detection system. http://www.tcpdump.org.

[3] Owasp page on cross-site request forgery attacks, available at https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF).

[4] Owasp page on cross-site scripting attacks, available at https://www.owasp.org/index.php/Cross-site_Scripting_(XSS).

[5] Prewikka source code, available online at https://github.com/g2p/prelude-prewikka.

[6] The samhain file integrity / host-based intrusion detection system. http://http://la-samhna.de/samhain/.

[7] Suricata ids by open information security foundation (oisf). http://www.openinfosecfoundation.org/.

[8] tcpreplay home page, available online at http://tcpreplay.sourceforge.net.

[9] Ieee standard for information technology-telecommunications and information exchange between systems-local and metropolitan area networks-specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999)*, pages C1–1184, 12 2007.

[10] Ieee standard for information technology-telecommunications and information exchange between systems-local and metropolitan area networks-specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 2: Fast basic service set (bss). *IEEE Std 802.11r-2008 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008)*, jul. 2008.

[11] Ieee standard for local and metropolitan area networks - part 21: Media independent handover services. *IEEE Std 802.21-2008*, nov. 2008.

[12] P. Albers, O. Camp, J. marc Percher, B. Jouga, and R. Puttini. Security in ad hoc networks: a general intrusion detection architecture enhancing trust based approaches. In *In Proceedings of the First International Workshop on Wireless Information Systems (WIS-2002*, pages 1–12, 2002.

[13] J. P. Anderson. Computer security threat monitoring and surveillance. Technical report, James P Anderson Co., FortWashington, Pennsylvania,USA, Apr. 1980.

[14] M. Andreolini, S. Casolari, M. Colajanni, and M. Marchetti. Dynamic load balancing for network intrusion detection systems based on distributed architectures. In *Proc. of the sixth IEEE International Symposium on Network Computing and Applications (NCA 2007)*, July 2007.

[15] M. Andreolini, S. Casolari, M. Colajanni, and M. Messori. Dynamic load management of virtual machines in cloud architectures. In D. R. Avresky, M. Diaz, A. Bode, B. Ciciani, E. Dekel, O. Akan, P. Bellavista, J. Cao, F. Dressler, D. Ferrari, M. Gerla, H. Kobayashi, S. Palazzo, S. Sahni, X. S. Shen, M. Stan, J. Xiaohua, A. Zomaya, and G. Coulson, editors, *Cloud Computing*, volume 34 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 201–214. Springer Berlin Heidelberg, 2010. DOI: 10.1007/978-3-642-12636-9_14.

[16] M. Aramoto, M. Nakamura, S. Sugimoto, and N. Takamiya. Umip: Usagi-patched mobile ipv6 for linux. home page available at http://umip.linux-ipv6.org/index.php.

[17] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Dns security introduction and requirements. RFC 4033 (PROPOSED STANDARD), March 2005.

[18] S. Axelsson. Intrusion detection systems: A survey and taxonomy. Technical report, 2000.

[19] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, E. H. Spafford, and D. Zamboni. An architecture for intrusion detection using autonomous agents. In *Proc. of the 14th Annual Computer Security Applications Conference (ACSAC 1998)*, December 1998.

[20] T. Bass. Multisensor data fusion for next generation distributed intrusion detection systems. In *Proc. of the 1999 DoD-IRIS National Symposium on Sensor and Data Fusion (NSSDF)*, May 1999.

[21] T. Bass. Intrusion detection systems and multisensor data fusion. *Communications of the ACM*, 43(4):99–105, May 2000.

[22] R. Beyah, S. Kangude, G. Yu, B. Strickland, and J. Copeland. Rogue access point detection using temporal traffic characteristics. In *Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE*, volume 4, pages 2271 – 2275 Vol.4, nov.-3 dec. 2004.

[23] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg. Live wide-area migration of virtual machines including local persistent state. In *Proceedings of the 3rd international conference on Virtual execution environments*, VEE '07, pages 169–179, New York, NY, USA, 2007. ACM.

[24] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. One ring to rule them all: service discovery and binding in structured peer-to-peer overlay networks. In *Proc. of the 10th workshop on ACM SIGOPS European workshop*, pages 140–145, New York, NY, USA, 2002. ACM.

[25] M. Castro, M. B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. J. Wang, and A. Wolman. An evaluation of scalable application-level multicast built using peer-to-peer overlays. In *Proc. of the 22nd Annual Joint Conference of the IEEE*

Computer and Communications Societies (INFOCOM 2003), San Franciso, CA, USA, April 2003.

[26] A. Celesti, F. Tusa, M. Villari, and A. Puliafito. Improving virtual machine migration in federated cloud environments. In *Evolving Internet (INTERNET), 2010 Second International Conference on*, pages 61 –67, sept. 2010.

[27] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41:15:1–15:58, July 2009.

[28] Y. Chu, J. Li, and Y. Yang. The architecture of the large-scale distributed intrusion detection system. In *Proc. of the Sixth International Conference on Parallel and Distributed Computing Applications and Technologies (PDCAT '05)*, December 2005.

[29] Cisco Fast Secure Roaming Application Note. Available online at http://www.cisco.com/en/US/products/hw/wireless/ps4570/prod_technical_reference09186a00801c5223.html.

[30] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[31] M. Colajanni, L. Dal Zotto, M. Marchetti, and M. Messori. The problem of nids evasion in mobile networks. In *Proc. of the 4th IFIP International Conference on New Technologies, Mobility and Security (NTMS 2011)*, Paris, France, February 2011.

[32] M. Colajanni, D. Gozzi, and M. Marchetti. Enhancing interoperability and stateful analysis of cooperative network intrusion detection systems. In *Proc. of the ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ACM/IEEE ANCS 2007)*, Orlando, FL, USA, Dec. 2007.

[33] M. Colajanni, D. Gozzi, and M. Marchetti. Collaborative architecture for malware detection and analysis. In *Proceedings of The IFIP 23rd International Information Security Conference (SEC 2008)*, September 2008.

[34] M. Colajanni and M. Marchetti. A parallel architecture for stateful intrusion detection in high traffic networks. In *Proc. of the IEEE/IST Workshop on "Monitoring, attack detection and mitigation" (MonAM 2006)*, Tuebingen, Germany, September 2006.

[35] M. Colajanni, M. Marchetti, and M. Messori. Selective and early threat detection in large networked systems. In *Proc. of the 10th IEEE International Conference on Computer and Information Technology (CIT 2010)*, Bradford, UK, June 2010.

[36] M. Colajanni, L. D. Zotto, M. Marchetti, and M. Messori. Defeating nids evasion in mobile ipv6 networks. In *World of Wireless Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a*, Lucca, Italy, June 2011.

[37] M. Corporation. Microsoft security bulletin ms07-062 important, available at http://www.microsoft.com/technet/security/bulletin/MS07-062.mspx.

[38] Capture the flag traffic dump, available online at http://www.defcon.org/html/links/dc-ctf.html.

[39] H. Debar, D. Curry, and F. B. The Intrusion Detection Message Exchange Format. RFC 4765, March 2007.

[40] H. Debar, M. Dacier, and A. Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):805 – 822, 1999.

[41] H. Debar and A. Wespi. Aggregation and correlation of intrusion-detection alerts. In *Proc. of the 4th International Symposium on Recent Advances in Intrusion Detection RAID 2001*, October 2001.

[42] D. E. Denning. An intrusion-detection model. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 13(2):222–232, 1987.

[43] Department of Homeland Security. Common vulnerabilities and exposures, 2008. Available at http://cve.mitre.org/.

[44] P. Druschel and A. Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, 2001.

[45] C. L. Dumitrescu. Intctd: A peer-to-peer approach for intrusion detection. In *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, CCGRID '06, pages 89–92, Washington, DC, USA, 2006. IEEE Computer Society.

[46] Dynamics mobile ip home page, available online at http://dynamics.sourceforge.net.

[47] Y. Espelid, L. Netland, A. Klingsheim, and K. Hole. Robbing banks with their own software—an exploit against norwegian online banks. In S. Jajodia, P. Samarati, and S. Cimato, editors, *Proceedings of The Ifip Tc 11 23rd International Information Security Conference*, volume 278 of *IFIP International Federation for Information Processing*, pages 63–77. Springer Boston, 2008.

[48] "esper: Event processing for java", available online at http://www.espertech.com/products/esper.php.

[49] P. Esteves Verssimo, E. Angori, M. Colajanni, M. Marchetti, and M. Messori. Collaborative attack detection using distributed hash tables. In R. Baldoni and G. Chockler, editors, *Collaborative Financial Infrastructure Protection*, pages 175–201. Springer Berlin Heidelberg, 2012. DOI: 10.1007/978-3-642-20420-3_9.

[50] J. M. Estevez-Tapiador, P. Garcia-Teodoro, and J. E. Diaz-Verdejo. Anomaly detection methods in wired networks: a survey and taxonomy. *Computer Communications*, 27(16):1569 – 1584, 2004.

[51] Freepastry, an open source implementation of pastry, available online at url-http://www.freepastry.org.

[52] Freeradius the world's most popular radius server. home page available at http://freeradius.org/.

[53] N. Golmie. Seamless mobility: are we there yet? *Wireless Communications, IEEE*, 16(4):12 –13, aug. 2009.

[54] Hornetq - putting the buzz in messaging, home page available at http://www.jboss.org/hornetq.

[55] V. Jacobson, C. Leres, and S. McCann. Pcap libraries.

[56] R. Janakiraman, M. Waldvogel, and Q. Zhang. Indra: A peer-to-peer approach to network intrusion detection and prevention. In *Proc. of the 12th IEEE International Workshops on Enabling Technologies*, Linz, Austria, June 2003.

[57] Remote method invocation (java rmi), available online at http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html.

[58] Java message service (jms), available online at http://www.oracle.com/technetwork/java/jms/index.html.

[59] O. Kachirski and R. Guha. Effective intrusion detection using multiple sensors in wireless ad hoc networks. *Hawaii International Conference on System Sciences*, 2:57a, 2003.

[60] R. A. Kemmerer. Nstat: A model-based real-time network intrusion detection system. Technical report, University of California at Santa Barbara, Santa Barbara, CA, USA, 1998.

[61] H.-A. Kim and B. Karp. Autograph: toward automated, distributed worm signature detection. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 19–19, Berkeley, CA, USA, 2004. USENIX Association.

[62] A. Klein. Bind 9 dns cache poisoning, available at http://www.trusteer.com/files/BIND_9_DNS_Cache_Poisoning.pdf.

[63] C. Kreibich and J. Crowcroft. Honeycomb: creating intrusion detection signatures using honeypots. *SIGCOMM Comput. Commun. Rev.*, 34:51–56, January 2004.

[64] C. Kruegel and W. Robertson. Alert verification: Determining the success of intrusion attempts. In *Proc. of the First Workshop the Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA 2004)*, 2004.

[65] S. Kumar. Classification and detection of computer intrusions, 1995.

[66] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das. Analysis and results of the 1999 darpa off-line intrusion detection evaluation. In *Proc. of the Third International Workshop on Recent Advances in Intrusion Detection*, Toulouse, France, October 2000.

[67] H. Liu, H. Jin, X. Liao, L. Hu, and C. Yu. Live migration of virtual machine based on full system trace and replay. In *Proceedings of the 18th ACM international symposium on High performance distributed computing*, HPDC '09, pages 101–110, New York, NY, USA, 2009. ACM.

[68] M. E. Locasto, J. J. Parekh, A. D. Keromytis, and S. J. Stolfo. Towards collaborative security and p2p intrusion detection. In *Proc. of the IEEE Information Assurance Workshop*, Maryland, USA, June 2005.

[69] L. Ma, A. Teymorian, and X. Cheng. A hybrid rogue access point protection framework for commodity wi-fi networks. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 1220 –1228, april 2008.

[70] D. J. Malan and M. D. Smith. Host-based detection of worms through peer-to-peer cooperation. In *Proc. of the 2005 ACM workshop on Rapid Malcode (WORM 2005)*, 2005.

[71] M. Marchetti, M. Colajanni, M. Messori, L. Aniello, and Y. Vigfusson. Cyber attacks on financial critical infrastructures. In R. Baldoni and G. Chockler, editors, *Collaborative Financial Infrastructure Protection*, pages 53–82. Springer Berlin Heidelberg, 2012. DOI: 10.1007/978-3-642-20420-3_3.

[72] M. Marchetti, M. Messori, and M. Colajanni. Peer-to-peer architecture for collaborative intrusion and malware detection at a large scale. In *Proc. of the 12th Information Security Conference (ISC 2009)*, Pisa, Italy, September 2009.

[73] J. Mchugh. Intrusion and intrusion detection. *International Journal of Information Security*, 1:14–35, 2001.

[74] M. Meeker, S. Devitt, and L. Wu. Morgan stanley internet trends, 2010. Available online at http://www.morganstanley.com/institutional/techresearch/internet_trends042010.html.

[75] D. Mutz, G. Vigna, and R. Kemmerer. An experience developing an IDS stimulator for the black-box testing of network intrusion detection systems. In *Proceedings of the 2003 Annual Computer Security Applications Conference (ACSAC '03)*, pages 374–383, Las Vegas, Nevada, December 2003.

[76] J. Newsome, B. Karp, and D. Song. Polygraph: automatically generating signatures for polymorphic worms. In *Security and Privacy, 2005 IEEE Symposium on*, pages 226 – 241, may 2005.

[77] S. Patton, W. Yurcik, and D. Doss. An achilles' heel in signature-based ids: Squealing false positives in snort. In *Proc. of the fourth International Symposium on Recent Advances in Intrusion Detection (RAID 2001).*, 2001.

[78] V. Paxson. Defending against network ids evasion. In *Recent Advances in Intrusion Detection*, 1999.

[79] C. E. Perkins. Mobile networking through mobile ip. *IEEE Internet Computing*, 2(1):58–69, January 1998.

[80] C. E. Perkins. Mobile ip. *IEEE Communications Magazine*, 40(5):66–82, May 2002.

[81] C. E. Perkins. Ip mobility support for ipv4, revised. *Request For Comments 5944, Internet Engineering Task Force*, November 2010.

[82] C. E. Perkins, D. Johnson, and J. Arkko. Mobility support in ipv6. RFC 6275 (PROPOSED STANDARD), July 2011.

[83] P. A. Porras and P. G. Neumann. Emerald: Event monitoring enabling responses to anomalous live disturbances. In *In Proceedings of the 20th National Information Systems Security Conference*, pages 353–365, 1997.

[84] Prelude hybrid intrusion detection system.

[85] T. H. Ptacek and T. N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, Inc., Suite 330, 1201 5th Street S.W, Calgary, Alberta, Canada, T2R-0Y6, 1998.

[86] T. R. Gula. Correlating IDS alerts with vulnerability information, available online at http://www.nessus.org/whitepapers/va-ids.pdf, 2002.

[87] D. Ragsdale, C. Carver, J. Humphries, and U. Pooch. Adaptation techniques for intrusion detection and intrusion response systems. In *Proc. of the IEEE International Conference on Systems, Man, and Cybernetics (SMC 2000)*, October 2000.

[88] R. U. Rehman. *Intrusion Detection Systems with Snort: Advanced IDS Techniques with Snort, Apache, MySQL, PHP, and ACID*. Prentice Hall PTR, Upper Saddle River, New Jersey 07458, USA, 2003.

[89] C. Rigney, S. Willens, A. Rubens, and W. Simpson. Remote Authentication Dial In User Service (RADIUS). RFC 2865 (Draft Standard), June 2000. Updated by RFCs 2868, 3575, 5080.

[90] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *Proc. of the eighteenth ACM symposium on Operating systems principles (SOSP 01)*, pages 188–201, New York, NY, USA, 2001. ACM.

[91] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. Scribe: The design of a large-scale event notification infrastructure. In *Proc. of the Third International Workshop on Networked Group Communication, (NGC 2001)*, 2001.

[92] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.

[93] P. Savola. radvd, home page available at http://www.litech.org/radvd/.

[94] SecurityFocus ™. Bugtraq mailing list, 2008. Available at http://www.securityfocus.com/.

[95] U. Shankar and V. Paxson. Active mapping: Resisting nids evasion without altering traffic. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy (SP 03)*, page 44, Washington, DC, USA, 2003. IEEE Computer Society.

[96] S. Siddharth. Evading nids, revisited. available online at http://www.symantec.com/connect/articles/evading-nids-revisited, 2005.

[97] R. Smith, C. Estan, and S. Jha. Backtracking algorithmic complexity attacks against a nids. Dec. 2006.

[98] S. R. Snapp, J. Brentano, G. V. Dias, T. L. Goan, L. T. Heberlein, C.-L. Ho, K. N. Levitt, B. Mukherjee, S. E. Smaha, T. Grance, D. M. Teal, and D. Mansur. *Internet besieged: countering cyberspace scofflaws*, chapter DIDS (distributed intrusion detection system)—motivation, architecture, and an early prototype, pages 211–227. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, May 1988.

[99] S. R. Snapp, J. Brentano, G. V. Dias, T. L. Goan, L. T. Heberlein, C.-L. Ho, K. N. Levitt, B. Mukherjee, S. E. Smaha, T. Grance, D. M. Teal, and D. Mansur. Dids (distributed intrusion detection system) motivation, architecture, and an early prototype. *Internet besieged: countering cyberspace scofflaws*, pages 211–227, 1998.

[100] Snort home page.

[101] R. Sommer and V. Paxson. Exploiting independent state for network intrusion detection. In *Proc. of the 21st Annual Computer Security Applications Conference*, Tucson, AZ, USA, December 2005.

[102] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. Grids - a graph-based intrusion detection system for large networks. In *Proc. of the 19th National Information Systems Security Conference*, October 1996.

[103] G. Thamilarasu, A. Balasubramanian, S. Mishra, and R. Sridhar. A cross-layer based intrusion detection approach for wireless ad hoc networks. In *Mobile Adhoc and Sensor Systems Conference, 2005. IEEE International Conference on*, pages 7 pp. –861, nov. 2005.

[104] F. Valeur, G. Vigna, C. Kruegel, and R. A. Kemmerer. A comprehensive approach to intrusion detection alert correlation. In *IEEE Transactions on Dependable and Secure Computing*, 2004.

[105] M. Vallentin, R. Sommer, J. Lee, C. Leres, V. Paxson, and B. Tierney. The nids cluster: Scalable, stateful network intrusion detection on commodity hardware. In *RAID*, pages 107–126, 2007.

[106] G. Vigna and R. Kemmerer. NetSTAT: A network-based intrusion detection system. *Journal of Computer Security*, 7(1):37–71, 1999.

[107] Y. Wang, H. Yang, X. Wang, and R. Zhang. Distributed intrusion detection system based on data fusion method. In *Proc. of the Fifth World Congress on Intelligent Control and Automation (WCICA 2004)*, June 2004.

[108] L. Watkins, R. Beyah, and C. Corbett. A passive approach to rogue access point detection. In *Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE*, pages 355 –360, nov. 2007.

[109] D. Winer. XMLRPC home page, available online at http://www.xmlrpc.com/.

[110] Y.-S. Wu, B. Foo, Y. Mei, and S. Bagchi. Collaborative intrusion detection system (cids): A framework for accurate and efficient ids. In *Proc. of the 19th Annual Computer Security Applications Conference*, December 2003.

[111] Q. Xue, J. Sun, and Z. Wei. Tjids: an intrusion detection architecture for distributed network. In *Proc. ot the 2003 IEEE Canadian Conference on Electrical and Computer Engineering CCECE 2003*, May 2003.

[112] V. Yegneswaran, P. Barford, and S. Jha. Global intrusion detection in the domino overlay system. In *Proc. of the ISOC Symposium on Network and Distributed Systems Security*, Feb. 2004.

[113] Y. Yoon, J. Y. Oh, and Y. M. Yoon. Nids evasion method named SeolMa. *Phrack Magazine*, 0x0b(0x39), June 2001.

[114] Y. Zhang, W. Lee, and Y.-A. Huang. Intrusion detection techniques for mobile wireless networks. *Wireless Networks*, 9:545–556, 2003. 10.1023/A:1024600519144.

[115] Y.-F. Zhang, Z.-Y. Xiong, and X.-Q. Wang. Distributed intrusion detection based on clustering. In *Proc. of 2005 International Conference on Machine Learning and Cybernetics*, Aug. 2005.

[116] Z. Zhang, J. Li, C. N. Manikopulos, J. Jorgenson, and J. Ucles. HIDE: a hierarchical network intrusion detection system using statistical preprocessing and neural

network classification. In *Proc. of the 2001 IEEE Workshop on Information Assurance and Security*, June 2001.

[117] Z. Zhang, J. Li, C. N. Manikopulos, J. Jorgenson, and J. Ucles. A hierarchical anomaly network intrusion detection system using neural network classification. In *Proc. of 2001 WSES Conferemce on Neural Networks and Applications (NNA '01)*, Feb. 2001.

[118] C. V. Zhou, S. Karunasekera, and C. Leckie. A peer-to-peer collaborative intrusion detection system. In *Proc. of the 13th IEEE International Conference on Networks*, 2005.