Universidad Carlos III de Madrid
Computer Science and Engineering Department

PhD Thesis

Attacks Against Intrusion Detection Networks: Evasion, Reverse Engineering and Optimal Countermeasures

Author:

Sergio Pastrana Portillo

Supervisors:

Dr. Agustín Orfila Díaz-Pabón

Dr. Juan Manuel Estévez Tapiador

Leganés, June 2014

Tesis Doctoral

Attacks Against Intrusion Detection Networks: Evasion, Reverse Engineering and Optimal Countermeasures

Autor: Sergio Pastrana Portillo

Directores: Dr. Agustín Orfila Díaz-Pabón
Dr. Juan Manuel Estévez Tapiador

Firma del Tribunal Calificador:

	Nombre y Apellidos	Firma
Presidente:	Prof. Andrés Marín López	.....
Vocal:	Prof. Sevil Şen	.....
Secretario:	Prof. David Camacho Fernández	.....

Calificación: .....

Leganés, ... de ..... de 2014.

To my family and friends.

Agradecimientos

Son muchas las personas a las que debo mostrar mi gratitud por su apoyo y por la ayuda prestada durante el desarrollo de esta Tesis. Lo primero, y como no podía ser de otra manera, a mis directores y compañeros Juan y Agustín, de quienes he aprendido muchísimas cosas, no solo en lo relativo al tema de Tesis, sino en lo relativo a la investigación, la docencia y la vida en general. Muchas gracias por haberme ayudado tanto, espero poder seguir aprendiendo de vosotros. También a Arturo, por la confianza depositada y el apoyo laboral recibido, que ha permitido que pueda centrarme en labores puramente académicas. Y a mis compañeros de universidad, Benjamín, Pedro, Anabel, Chema, Lorena, Almudena, Esther, José, Israel, etc., por los momentos y las experiencias compartidas, siempre enriquecedoras. Me gustaría hacer una mención aparte a Guillermo y Jorge, quienes además de colegas de trabajo son grandes amigos desde hace ya unos cuantos años. Jorge, tú me animaste a realizar el doctorado, y a ti tengo que agradecerte de forma especial el hecho de que hoy esté escribiendo estas palabras. De verdad que vales millones. Y Guille, compañero de batallas y de despacho, en ti he encontrado un gran apoyo en esos momentos en los que más se necesitan. Gracias.

Por supuesto, nada de esto sería posible sin la ayuda de mis padres y de mi hermano, grandes mecenas y promotores de mi formación y educación. No hace falta que os diga todo lo que agradezco vuestro apoyo y ánimos que me habéis dado siempre, y lo fundamentales que sois en mi vida. A mi abuela, que es una de las personas más maravillosas que existen. A Rebeca, en quien he encontrado una gran cuñada y que espero que lo sea por mucho tiempo. A Marisol, Alfonso, Nerea y Cristian, quienes pronto pasarán a ser oficialmente mi familia, aunque para mi lleváis mucho tiempo siéndolo.

A mis amigos, Álvaro, Charlie, Gulas, Enrique, Vicen, Marcos, Valde, Vanessa, Torri, Tamara, Ana, Marc, Irene, y muchísimos más que aquí no puedo mencionar (vosotros sabéis quienes sois), porque algo básico en el desarrollo de mi Tesis ha sido el poder desconectar de ésta y disfrutar del ocio y el tiempo libre, algo que con

vosotros tengo más que asegurado. Echarse unas risas, tomar unas cervezas (sobre todo si son “birrotes”), salir al monte, darlo todo en conciertos, etc.; sin todos esos momentos no llego aquí, creedme. Y por supuesto a Alex, con quien me gustaría haber compartido muchos más momentos de nuestra etapa como doctorandos, si bien no dejas de ser un gran amigo y una gran persona en mi vida, incluso ahora en la distancia.

Y para finalizar, a Noelia, la persona a quien más tengo que agradecer y que más ha tenido que soportar mis momentos difíciles. Porque nada de lo que estoy cerca de conseguir habría sido posible sin ti. Por hacer que los momentos malos sean buenos, y que los buenos sean mejores. De corazón, gracias por seguir ahí.

A todos los que habéis contribuido de una forma u otra a que un servidor esté cerca de llegar a Doctor, ¡GRACIAS!

Sergio

Abstract

Intrusion Detection Networks (IDNs) constitute a primary element in current cyberdefense systems. IDNs are composed of different nodes distributed among a network infrastructure, performing functions such as local detection–mostly by Intrusion Detection Systems (IDS)–, information sharing with other nodes in the IDN, and aggregation and correlation of data from different sources. Overall, they are able to detect distributed attacks taking place at large scale or in different parts of the network simultaneously.

IDNs have become themselves target of advanced cyberattacks aimed at bypassing the security barrier they offer and thus gaining control of the protected system. In order to guarantee the security and privacy of the systems being protected and the IDN itself, it is required to design resilient architectures for IDNs capable of maintaining a minimum level of functionality even when certain IDN nodes are bypassed, compromised, or rendered unusable. Research in this field has traditionally focused on designing robust detection algorithms for IDS. However, almost no attention has been paid to analyzing the security of the overall IDN and designing robust architectures for them.

This Thesis provides various contributions in the research of resilient IDNs grouped into two main blocks. The first two contributions analyze the security of current proposals for IDS nodes against specific attacks, while the third and fourth contributions provide mechanisms to design IDN architectures that remain resilient in the presence of adversaries.

In the first contribution, we propose evasion and reverse engineering attacks to anomaly detectors that use classification algorithms at the core of the detection engine. These algorithms have been widely studied in the anomaly detection field, as they generally are claimed to be both effective and efficient. However, such anomaly detectors do not consider potential behaviors incurred by adversaries to decrease the effectiveness and efficiency of the detection process. We demonstrate that using well-known classification algorithms for intrusion detection is vulnerable to reverse

engineering and evasion attacks, which makes these algorithms inappropriate for real systems.

The second contribution discusses the security of randomization as a countermeasure to evasion attacks against anomaly detectors. Recent works have proposed the use of secret (random) information to hide the detection surface, thus making evasion harder for an adversary. We propose a reverse engineering attack using a query-response analysis showing that randomization does not provide such security. We demonstrate our attack on Anagram, a popular application-layer anomaly detector based on randomized n-gram analysis. We show how an adversary can first discover the secret information used by the detector by querying it with carefully constructed payloads and then use this information to evade the detector.

The difficulties found to properly address the security of nodes in an IDN motivate our research to protect cyberdefense systems globally, assuming the possibility of attacks against some nodes and devising ways of allocating countermeasures optimally. In order to do so, it is essential to model both IDN nodes and adversarial capabilities. In the third contribution of this Thesis, we provide a conceptual model for IDNs viewed as a network of nodes whose connections and internal components determine the architecture and functionality of the global defense network. Such a model is based on the analysis and abstraction of a number of existing proposals for IDNs. Furthermore, we also develop an adversarial model for IDNs that builds on classical attack capabilities for communication networks and allow to specify complex attacks against IDN nodes.

Finally, the fourth contribution of this Thesis presents DEFIDNET, a framework to assess the vulnerabilities of IDNs, the threats to which they are exposed, and optimal countermeasures to minimize risk considering possible economic and operational constraints. The framework uses the system and adversarial models developed earlier in this Thesis, together with a risk rating procedure that evaluates the propagation of attacks against particular nodes throughout the entire IDN and estimates the impacts of such actions according to different attack strategies. This assessment is then used to search for countermeasures that are both optimal in terms of involved cost and amount of mitigated risk. This is done using multi-objective optimization algorithms, thus offering the analyst sets of solutions that could be applied in different operational scenarios.

Resumen

Las Redes de Detección de Intrusiones (IDNs, por sus siglas en inglés) constituyen un elemento primordial de los actuales sistemas de ciberdefensa. Una IDN está compuesta por diferentes nodos distribuidos a lo largo de una infraestructura de red que realizan funciones de detección de ataques–fundamentalmente a través de Sistemas de Detección de Intrusiones, o IDS–, intercambio de información con otros nodos de la IDN, y agregación y correlación de eventos procedentes de distintas fuentes. En conjunto, una IDN es capaz de detectar ataques distribuidos y de gran escala que se manifiestan en diferentes partes de la red simultáneamente.

Las IDNs se han convertido en objeto de ataques avanzados cuyo fin es evadir las funciones de seguridad que ofrecen y ganar así control sobre los sistemas protegidos. Con objeto de garantizar la seguridad y privacidad de la infraestructura de red y de la IDN, es necesario diseñar arquitecturas resilientes para IDNs que sean capaces de mantener un nivel mínimo de funcionalidad incluso cuando ciertos nodos son evadidos, comprometidos o inutilizados. La investigación en este campo se ha centrado tradicionalmente en el diseño de algoritmos de detección robustos para IDS. Sin embargo, la seguridad global de la IDN ha recibido considerablemente menos atención, lo que ha resultado en una carencia de principios de diseño para arquitecturas de IDN resilientes.

Esta Tesis Doctoral proporciona varias contribuciones en la investigación de IDN resilientes. La investigación aquí presentada se agrupa en dos grandes bloques. Por un lado, las dos primeras contribuciones proporcionan técnicas de análisis de la seguridad de nodos IDS contra ataques deliberados. Por otro lado, las contribuciones tres y cuatro presentan mecanismos de diseño de arquitecturas IDS robustas frente a adversarios.

En la primera contribución se proponen ataques de evasión e ingeniería inversa sobre detectores de anomalías que utilizan algoritmos de clasificación en el motor de detección. Estos algoritmos han sido ampliamente estudiados en el campo de la detección de anomalías y son generalmente considerados efectivos y eficientes. A

pesar de esto, los detectores de anomalías no consideran el papel que un adversario puede desempeñar si persigue activamente decrementar la efectividad o la eficiencia del proceso de detección. En esta Tesis se demuestra que el uso de algoritmos de clasificación simples para la detección de anomalías es, en general, vulnerable a ataques de ingeniería inversa y evasión, lo que convierte a estos algoritmos en inapropiados para sistemas reales.

La segunda contribución analiza la seguridad de la aleatorización como contramedida frente a los ataques de evasión contra detectores de anomalías. Esta contramedida ha sido propuesta recientemente como mecanismo de ocultación de la superficie de decisión, lo que supuestamente dificulta la tarea del adversario. En esta Tesis se propone un ataque de ingeniería inversa basado en un análisis consulta-respuesta que demuestra que, en general, la aleatorización no proporciona un nivel de seguridad sustancialmente superior. El ataque se demuestra contra *Anagram*, un detector de anomalías muy popular basado en el análisis de n-gramas que opera en la capa de aplicación. El ataque permite a un adversario descubrir la información secreta utilizada durante la aleatorización mediante la construcción de paquetes cuidadosamente diseñados. Tras la finalización de este proceso, el adversario se encuentra en disposición de lanzar un ataque de evasión.

Los trabajos descritos anteriormente motivan la investigación de técnicas que permitan proteger sistemas de ciberdefensa tales como una IDN incluso cuando la seguridad de algunos de sus nodos se ve comprometida, así como soluciones para la asignación óptima de contramedidas. Para ello, resulta esencial disponer de modelos tanto de los nodos de una IDN como de las capacidades del adversario. En la tercera contribución de esta Tesis se proporcionan modelos conceptuales para ambos elementos. El modelo de sistema permite representar una IDN como una red de nodos cuyas conexiones y componentes internos determinan la arquitectura y funcionalidad de la red global de defensa. Este modelo se basa en el análisis y abstracción de diferentes arquitecturas para IDNs propuestas en los últimos años. Asimismo, se desarrolla un modelo de adversario para IDNs basado en las capacidades clásicas de un atacante en redes de comunicaciones que permite especificar ataques complejos contra nodos de una IDN.

Finalmente, la cuarta y última contribución de esta Tesis Doctoral describe *DEFIDNET*, un marco que permite evaluar las vulnerabilidades de una IDN, las amenazas a las que están expuestas y las contramedidas que permiten minimizar

el riesgo de manera óptima considerando restricciones de naturaleza económica u operacional. *DEFIDNET* se basa en los modelos de sistema y adversario desarrollados anteriormente en esta Tesis, junto con un procedimiento de evaluación de riesgos que permite calcular la propagación a lo largo de la IDN de ataques contra nodos individuales y estimar el impacto de acuerdo a diversas estrategias de ataque. El resultado del análisis de riesgos es utilizado para determinar contramedidas óptimas tanto en términos de coste involucrado como de cantidad de riesgo mitigado. Este proceso hace uso de algoritmos de optimización multiobjetivo y ofrece al analista varios conjuntos de soluciones que podrían aplicarse en distintos escenarios operacionales.

Contents

List of Figures

List of Tables

1

Introduction

1.1 Context

According to the C.I.A. World Factbook [C.I.A., 2010], more than 2.1 billion people in the world (29.6% of the estimated world population) are connected to Internet, and almost 6 billion people use mobile phones (84% of the world population). Cyberspace plays a key role in modern societies and economies [Patel et al., 2013]. Over the last decade, Internet has changed the way we interact with Public Administrations, has given rise to new business and entertainment models, and has influenced the way we communicate. However, as acknowledged by the Spanish National Security Strategy, cyberspace is nowadays an open and uncontrolled environment. The complexity and globalization of cyberattacks has increased very significantly in the last years, causing an important breach between the capabilities of attackers and defenders.

Internet is rapidly evolving to embrace new computing and communication paradigms, such as cloud computing, wireless networks, and smart mobile platforms. Cloud computing services allow users to externalize computing resources (e.g., networks, servers, storage, applications, and services) to a remotely located site and use them on-demand [Mell and Grance, 2011]. Wireless networks have also gained much popularity in the last decade, particularly those based in the 802.11 (Wi-Fi) family [Crow et al., 1997] both for home access and in urban spaces [Ylipulli et al., 2013]. While this has contributed very significantly to facilitating Internet access from almost everywhere by using mobile devices, it has also created many new vulnerabilities.

Apart from personal users, companies and governments have become highly dependent on the cyberspace for their daily activities. Critical tasks such as communication activities, business and financial transactions, and even the control of critical

infrastructures now depend on the Internet. Until very recently, most of these tasks were carried out in confined and –in many cases– carefully secured platforms. Threats such as espionage and data loss were certainly a concern, but since such systems were rarely accessible from the outside, the spectrum of potential attack avenues was limited. In contrast, most current organizations have their systems permanently connected to Internet, and many of them are offloading parts of their infrastructures to the cloud. This has resulted in a substantial increase of the threats they are exposed to [Santora, 2013], and has given rise to terms such as "cyber-espionage", "cyber-warfare", or "cyber-terrorism".

The profile, intentions, and capabilities of attackers have also changed extraordinarily. Until very recently, they were considered "socially isolated young men" [Jordan and Taylor, 1998] driven by motivations such as gaining notoriety in the hacking underground scene or by cyberactivism campaigns. Although most of them were technically very skilled, they did not have substantial economic resources to develop sophisticated attacks. Nowadays, however, both the sophistication of attacks and the motivations of the adversaries have evolved. For instance, the so-called Advanced Persistent Threats (APTs) [Tankard, 2011] combine sophisticated techniques and exploitation vectors to bypass security mechanisms and remain undetected for prolonged periods of time in attacked systems. Moreover, rather than "isolated obsessed young men", attackers have turned into organized teams with economical or political motivations, targeting high valued infrastructures from governments and big companies. Such teams are often hired or sponsored by rival companies or governments. Stuxnet [Falliere et al., 2011], for example, is an APT—and one of the first cyberweapons—that targeted Industrial Control Systems (ICSs) and was used to attack uranium enrichment plants in Iran. Because of its sophistication and the amount of resources estimated for its development, it is believed that some government was behind it.

The situation discussed above demands more intelligent countermeasures to protect the networks and critical systems. According to the well-known Common Criteria for ICT systems evaluation [Common-Criteria, 2012], properly protecting the assets of an organization requires that countermeasures fulfill two main properties (see Figure 1.1). First, countermeasures should be sufficient, i.e., if the countermeasures do what they claim to do, the threats to the assets are countered. Second, countermeasures must be correct, i.e., they must do what they claim to do.

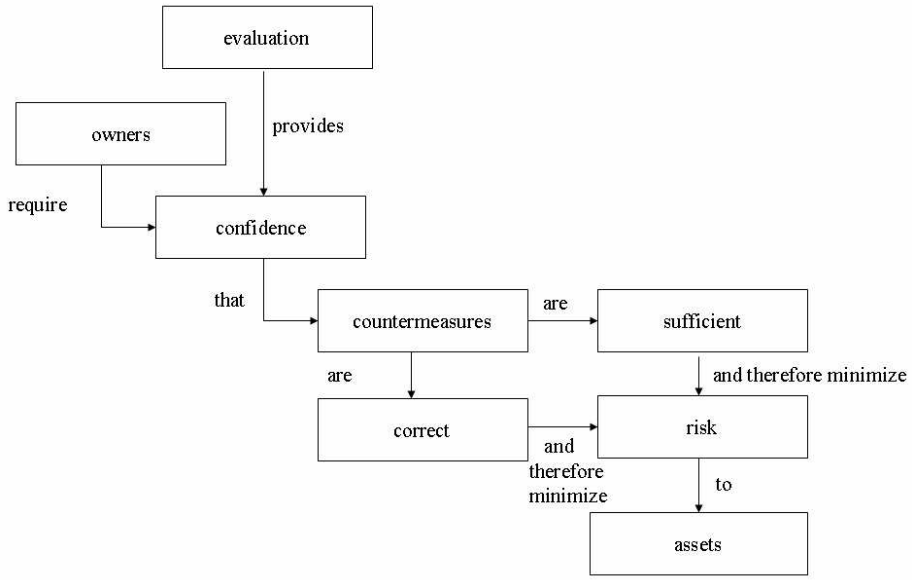


Figure 1.1: Evaluation concepts and relationships established by the Common Criteria.

1.1.1 Cyberdefense Systems

Intrusion Detection Systems (IDS) constitute a primary component for securing computing infrastructures. An IDS monitors activity and seeks to identify evidence of ongoing attacks, intrusion attempts, or violations of the security policies [Scarfone and Mell, 2007]. IDSs have evolved since the first model proposed in the late 1980s [Denning, 1987], and the current threat landscape makes the classical approach for intrusion detection no longer valid. Moreover, intrusion detection must also deal with emerging paradigms in computing and communications. For example, performing detection in wireless nodes such as smartphones [Suarez-Tangil et al., 2013] or wearable sensing devices [Al Ameen et al., 2012], requires lightweight procedures that do not consume much resources like energy or memory.

Detection paradigms and architectures have also evolved to cope with the requirements of complex network infrastructures. Rather than stand-alone components strategically placed to protect a complete network or system, the current trend is to rely on a distributed network of detection nodes. Intrusion Detection Networks (IDN) are composed of different IDS nodes distributed among a network performing local detection and sharing information with other nodes in the IDN. One of the major advantages of IDNs is that, because the detection functions are distributed across different network locations, so is the workload required for each function. Classical

intrusion detection components such as Snort [Roesch, 1999] must be implemented in a single device. Therefore, this host is in charge of gathering the data (monitor the network), pre-process it, running detection algorithms, and generating responses accordingly. This approach is inappropriate both for resource-constrained scenarios and for large networks. The problem becomes even harder if the worst-case scenario for detection is forced by an adversary [Crosby and Wallach, 2003; Smith et al., 2006].

IDNs attempt to solve this problem by distributing the tasks among different nodes. Depending on their role in the network, some nodes gather local data and send it to another node, probably with more resources, who correlates the data and performs actual detection. This separation of duties makes IDNs a suitable solution for distributed systems, including mobile ad hoc networks (MANETs), where there are no central nodes and every host must collaborate to ensure a proper network behavior [Pastrana et al., 2012]. IDNs are also used in networks geographically separated to allow different entities to collaborate and mitigate large scale attacks [Bye et al., 2010]. Current attacks are capable of infecting simultaneously various networks or incorporating evasion techniques to pass undetected [Fogla and Lee, 2006]. Moreover, many zero-day attacks target simultaneously a huge number of systems worldwide [Shin and Gu, 2010], leaving little time to patch other networks. Thus, to prevent threats from propagating through different domains, collaboration between different IDNs is essential.

Since they are key elements of most organizations' cyberdefense systems, IDSs often become themselves the target of attacks aimed at undermining their detection capabilities. This may result in the degradation of the second property evaluated by the Common Criteria, which states that countermeasures must be correct. Actually, when attacking a system, the adversary's first goal is to degrade the effectiveness of the cyberdefenses, thus making the countermeasures inappropriate. In the case of IDNs, attackers may use common attacks for networks to degrade the efficacy of the detection accuracy. Most of the research in this area has focused on attacks against stand-alone IDS boxes and the design of robust detection algorithms [Biggio et al., 2013; Fogla et al., 2006; Kolesnikov and Lee, 2005; Ptacek and Newsham, 1998; Wagner and Soto, 2002]. However, almost no attention has been paid to robust architectures for IDNs, and only a few works have very recently begun to consider attacks against IDNs [Bye et al., 2010; Fung, 2011].

1.1.2 Machine Learning Based Detection

One of the major problems that IDSs must face is adapting to continuous changes both in the networks they defend and the capabilities of the adversaries. The use of Machine Learning (ML) algorithms in the detection process has been extensively studied as a potential solution to this problem. Concretely, classification algorithms are often used to automatically build classifiers that map events as normal or intrusive. This approach has resulted successful in many scenarios. However, the use of ML has been recently questioned because its design does not consider adversaries [Barreno et al., 2006; Biggio et al., 2013; Sommer and Paxson, 2010] and, therefore the resulting detectors can be potentially manipulated. One of the proposed methods to counteract attacks and make resilient ML algorithms is the use of secret or random components [Huang et al., 2011; Mrdovic and Drazenovic, 2010; Wang et al., 2006]. While randomization is extremely useful in many security applications such as cryptography, it is unclear whether it is robust enough in the field of intrusion detection.

Similar to ML algorithms, Evolutionary Computation (EC) is a branch of Artificial Intelligence widely used in the research of intrusion detection. EC involves global optimization methods to solve the problem at stake. In the case of intrusion detection, the problem being solved is the detection of anomalous or intrusive events. One of the key advantages of EC is that resulting detectors are easy to understand and process [Orfila et al., 2009], which is a desired property for IDS. Another advantage of EC methods is that they result rather efficient in challenging scenarios, like Mobile Ad Hoc Networks (MANETs) [Sen and Clark, 2011], where the adversarial model rapidly changes and obtaining responses in real time is critical.

1.2 Motivation and Objectives

IDSs, both working as isolated components or as part of a larger IDN, are designed to provide attack detection capabilities to a protected network. These systems are generally placed in the network perimeter and, consequently, are among the first security barriers that an adversary encounters. Advanced adversaries are increasingly developing attack techniques against IDS nodes with the aim of counteracting their detection capabilities. The Common Criteria [Common-Criteria, 2012] also

establishes that "*owners of assets [...] may choose to increase their confidence in the sufficiency and correctness of some or all of their countermeasures by ordering an evaluation of these countermeasures*". Accordingly, it is critical to provide methods to facilitate such evaluation for IDSs and IDNs.

This Thesis considers the problem of attacks and defenses against IDNs, providing specific attacks to machine learning based anomaly detectors and optimal solutions to make the detection resilient. We next describe the main motivation and objectives of this work. Firstly, we state that the traditional use of classification algorithms for intrusion detection is prone to attacks, and we question the use of randomization as a measure to secure such algorithms. Secondly, we establish the need of systematic approaches for evaluating the security of IDNs in adversarial environments.

1.2.1 Motivation

Problem 1: Vulnerabilities of intrusion detection algorithms based on machine learning

The design of machine learning based algorithms for intrusion detection, particularly for anomaly-based detection, should consider the presence of adversaries that will attack the detection function itself. Unfortunately, classical machine learning algorithms are not designed with this assumption in mind. Attacks against the detector can aim at reverse engineering its inner workings, i.e., learning how the algorithm decides whether an instance belongs to one class or another. This is generally used to devise strategies to evade detection, where the main goal is to carefully modify an instance that would be classified as anomalous so that it becomes normal and still achieves the attacker's goals. Both reverse engineering and evasion attacks deal with one fundamental limitation of classical classification algorithms: they do not attempt to *hide* the decision surface. In an intrusion detection setting, the adversary has the ability to interact with the algorithm, obtain the classification labels for instances of his choice, and use the knowledge gained about how detection works to create, for example, attacks that will pass undetected (false negatives) or normal instances that will raise an alarm (false positives). Such undesirable, induced behaviors—along with some others that will be described in the next chapter— will undermine the confidence put on the detectors.

Some advanced detectors (e.g., KIDS [Mrdovic and Drazenovic, 2010] or Anagram [Wang et al., 2006]) have suggested that *randomization* of the decision surface will make reverse engineering and evasion harder for an attacker. The central idea of such algorithms consists of introducing a random, but controlled, component into the classification process. This is often implemented as a secret key, in such a way that an adversary who is not in possession of the key will not know exactly how the traffic will be processed and, consequently, will not be able to design attacks that thwart detection. The security of randomized classifiers has not been proved yet, and it remains unclear whether it suffices to prevent reverse engineering and evasion attacks, or it just makes them more difficult but still possible and realistic.

Problem 2: Resilient intrusion detection networks

Current cyberdefense systems are composed of a collection of networked components that include data collectors, filters, aggregators, correlators, etc. Detection is thus a complex distributed function that depends on information collected, processed, and exchanged by different subsystems. So far, security analysis about cyberdefense systems has focused almost exclusively on individual detection nodes, neglecting both the threats the overall IDN is exposed to, and robust design principles to make them resilient against them. For instance, many current IDNs follow a hierarchical architecture where security-related events flow from distributed detectors up to aggregation and correlation components located on the top tiers. In this scenario, it is not properly understood the impact that attacks on a subset of all the IDN components would have on the overall detection function, nor what elements (e.g., the roles of the attacked subsystems and how their outputs are processed by subsequent components) are relevant for the attack to succeed. Conversely, it is also unknown how to design architectures that resist, at least to some extent, certain types of deliberate attempts to evade detection or, more generally, subvert the protection offered by the IDN.

1.2.2 Objectives

The main goal of this Thesis is to i**mprove the security of intrusion detection systems and networks operating in adversarial settings by developing**

techniques to analyze their vulnerabilities and countermeasures to increase their resiliency. In particular, we will focus on the following three general objectives:

- **O1**. Study techniques to reverse engineer and evade IDSs based on machine learning algorithms that could be used to assess the security of current detection approaches and to devise countermeasures that make them resistant against adversarial manipulations.
- **O2**. Develop a suitable model for (a) IDNs in adversarial environments that integrates the key features of individual components and existing architectural options; and (b) goals, tactics, and capabilities of adversaries aiming at disrupting the IDN operation.
- **O3**. Based upon a model such as that described in the point above, study techniques to explore the vulnerabilities of IDNs, the threats to which they are exposed, and optimal countermeasures to minimize risk considering possible economic and operational constraints.

1.3 Contributions and Organization

This Thesis provides several contributions in the field of resilient IDNs along the lines discussed in the main objectives above. These contributions are grouped in the following four points:

1. **A general technique to explore reverse engineering and evasion attacks against machine learning based IDS**. Chapter 3 describes these attacks and provides experimental results against IDS using detectors based on classifiers.
2. **A reverse engineering attack against randomized classifiers**. One of the mechanisms proposed to increase the security of anomaly detectors is the use of randomization to hide the decision surface from adversaries. Chapter 4 analyzes this solution and provides a sophisticated attack against Anagram, a popular randomized anomaly detector. It is also proved that discovering

the secret key makes evasion easier than in the non-randomized setting, as it provides further capabilities to an adversary.

3. **A model for IDNs and adversarial capabilities**. Chapter 5 provides a general model for IDNs operating in adversarial settings. The model is illustrated with two common scenarios for IDNs, providing examples of such adversarial capabilities.

4. **A framework for the optimal allocation of countermeasures in IDNs**. Chapter 6 presents a framework that allows to automatically determine the overall risk an IDN is subject to as a consequence of potential attacks against individual components. Furthermore, the chapter discusses the application of multi-objective optimization algorithms to search for countermeasures that are optimal both in terms of risk and cost.

Finally, Chapter 7 presents the main conclusions, analyzes the contributions of this Thesis and the published results, and discusses open research problems and future work.

2

Intrusion Detection Networks in Adversarial Environments

This chapter analyzes the state of the art in the area of attacks against intrusion detection solutions, including stand-alone IDSs and IDNs. First, Section 2.1 presents an overview of IDNs, including common detection approaches and classical IDSs. Then, Section 2.2 reviews the state of the art of attacks against IDSs and IDNs and presents a recent taxonomy for attacks which is adopted in this Thesis. Finally, Section 2.3 summarizes the chapter and discusses the main problems in the field.

2.1 Intrusion Detection Networks

Intrusion detection is the process of identifying and/or blocking any attempt to bypass the security of a system [Bace and Mell, 2001]. Due to the continuous evolution of Information and Communication Technology (ICT), intrusion detection is an open and volatile research field. The seminal work of Denning in 1987 [Denning, 1987] proposed a general model for intrusion detection which is independent of specific technologies or implementations. Denning stated that "*exploitation of a system's vulnerabilities involves abnormal use of the system; therefore, security violations could be detected from abnormal patterns of system usage*". Nowadays, almost 30 years after the work of Denning, this statement is still valid. However, as the complexity of network and information systems increase, so does the capabilities of adversaries, which continuously forces the intrusion detection research to adapt to new adversaries.

Intrusion Detection Systems have evolved since the work of Denning. Due to the increasingly sophisticated threat of current attackers, the classical approach for

intrusion detection is no longer valid. Moreover, intrusion detection must also face the challenge posed by new paradigms in computing and communications, including cloud computing and the new generation of wireless technologies.

This section first reviews the classical approach of IDS which work stand-alone, i.e., without sharing information with other IDSs. Then, the main detection approaches are presented, with special emphasis in the machine learning algorithms which have been widely used for anomaly detection and are subject of study in parts of this Thesis. Finally, we discuss the need of IDNs to address the deficiencies of classical IDSs.

2.1.1 Intrusion Detection Systems

An IDS is a system that analyzes data to detect malicious activity, reporting an alert if such an activity is found. IDSs are normally formed from several components. In the most classical architecture, IDSs consists of 4 components [Corona et al., 2013] (see Figure 2.1), namely the decoder, the preprocessor (or set of preprocessors), the detection engine and the alert module. The way in which these components work is described next:

1. The **decoder** receives pieces of raw audit data from the audit data collectors and transforms each of these pieces into data that the preprocessor can handle.

2. The **preprocessor** extracts features from the raw data. It receives the pieces of data transformed by the decoder, analyzes them to determine which pieces depend on each other and treats dependent pieces in such a way that they can be later scrutinized by the detection engine. A typical preprocessor widely used in network-based IDSs is the TCP preprocessor, whose main task is to compose session flows from a given set of TCP segments (reordering fragments, assembling them, etc). Currently, sophisticated preprocessors are able to perform detection tasks supplementing those performed by the detection engine.

3. The **detection engine** receives the data treated by the preprocessor and examines it searching for intrusions. If an intrusion is found, the detection engine requests the alert module to raise an alert.

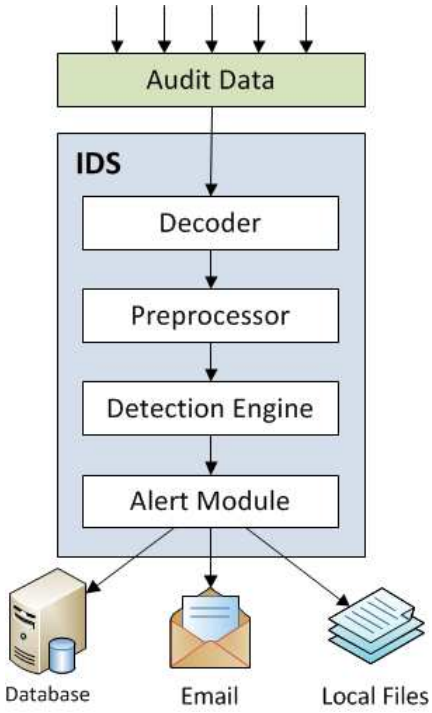


Figure 2.1: Architecture of a classical IDS.

4. The **alert module** is in charge of raising the alerts requested by the detection engine. Raising an alert can range from logging the alert in a local file to emailing the alert to the system administrator.

There exist many different taxonomies to classify IDSs, depending on the corresponding component of the IDS [Patel et al., 2013]:

1. Regarding the source of the audit data, an IDS can be network based or host based:

   (a) **Network IDSs (NIDSs)**: they analyze network traffic. The level of detection may vary from one NIDS to another, but most of them have modules in charge of analyzing traffic from the network, transport, and application layers in the OSI model. For instance, Snort [Roesch, 1999], one of the most used open source IDSs, has a preprocessor specialized in HTTP data, another one for TCP data and the same for the other protocols and layers in the OSI model. NIDSs are normally placed outside the system being monitored but in the same network segment, thus enabling them to monitor a complete LAN.

   (b) **Host IDSs (HIDSs)**: they analyze local data of the devices. Most of them analyze the sequence of system calls of the programs running in the device. Within these sequences, optimal HIDS analyze system call arguments, memory registers, stack states, system logs, user behaviors, etc.

2. Regarding the model used to detect malicious activity, an IDS can be **misuse-based**, **anomaly-based** or **hybrid**. In next section we analyze in detail these approaches.

3. Regarding the type of action triggered when a malicious behavior is detected, an IDS can be active or passive:

   (a) **Passive IDS**: when a malicious behavior is detected, an alert is raised and no further action is taken.

   (b) **Active IDS**: apart from raising an alert, the IDS tries to neutralize the malicious data by executing a predefined action. Some authors refer to active IDSs as Intrusion Prevention System (IPS).

There are many other possible classifications. For example, in [Amer and Hamilton, 2010], a taxonomy based on the following characteristics is presented:

1. Regarding the **technology**, IDSs may be *wired* or *wireless*. Furthermore, wireless IDSs can be further classified as fixed or mobile.

2. Regarding the **data processing method** and the arrangement of its components, IDSs can be *centralized* or *distributed*.

3. Regarding the **timing** of the detection process, IDSs can be *real time* or *non-real time*.

4. Regarding the detection technique, IDSs can be *state-based* or *transition-based*.

In order to evaluate the effectiveness of IDSs, two important measures are mainly used: the hit rate and the false positive rate. The hit rate (denoted H or true positive rate) measures the effectiveness of an IDS by indicating the percentage of intrusions that it detects (see Equation 2.1). The false positive rate (denoted F)

Table 2.1: Contingency matrix for binary classification problems: true negatives (TN), false positives (FP), false negatives (FN) and true positives (TP).

		Detection	
		Negative	Positive
Real	Negative	TN	FP
	Positive	FN	TP

measures the accuracy of an IDS by indicating the percentage of false alarms that it raises (see Equation 2.2). In order to calculate these two statistics, the following four values are necessary (see contingency matrix in Table 2.1): the number of real intrusions detected (true positives or TP), the number of real intrusions undetected (false negatives or FN), the number of alarms raised without any real intrusion taking place (false positives or FP), and the number of normal events considered normal (true negatives or TN). As a binary classification problem, IDSs are often evaluated using metrics from the information retrieval field such as the precision and recall. Considering an intrusion as the relevant information to be retrieved from the data, the recall indicates the fraction of relevant instances retrieved, i.e., it coincides with the hit rate. The precision indicates the fraction of retrieved instances that are relevant; i.e., given an alarm by the IDS, the precision indicates the likelihood that this alarm actually represents and intrusion (see Equation 2.3).

$$H = \frac{TP}{TP + FN} \tag{2.1}$$

$$F = \frac{FP}{FP + TN} \tag{2.2}$$

$$Precision = \frac{TP}{FP + TP} \tag{2.3}$$

Another important statistic that has recently become relatively popular in the field of IDSs evaluation is the Intrusion Detection Capability index [Gu et al., 2006] (denoted C_{ID}). C_{ID} is the ratio of the reduction of uncertainty of the IDS input, given the IDS output. It is formally defined as:

$$C_{ID} = \frac{I(X;Y)}{H(X)} \tag{2.4}$$

Where $I(X;Y)$ is the mutual information between the set of inputs to the IDS (X) and their corresponding outputs (Y); and $H(X)$ is the entropy of the input data. The higher the C_{ID} is, the better capacity the IDS has to classify the input properly. Besides the hit rate (H) and the false positive rate (F), the C_{ID} takes into account the prevalence B of attacks in the dataset, defined as:

$$B = \frac{FN + TP}{TP + FP + TN + FN} \tag{2.5}$$

Since not all systems have the same probability of being attacked, the C_{ID} index provides a more accurate measure than the hit rate and the false positive rate. Moreover, evaluating an IDS using H and F is somehow difficult, as various points of operation (i.e., pair H, F) are considered, and it must be defined an optimal tradeoff between the two measures. As the C_{ID} considers the two measures along with the prevalence of attacks, it can be used as a single scalar measure to evaluate the IDS. Using the metrics H, F and B, extracted from the contingency matrix (Table 2.1), the C_{ID} can be computed as shown in Equation 2.6. Further details about this measure can be found in [Gu et al., 2006].

$$\begin{aligned} C_{ID} = & -BH \log \frac{BH}{BH + HF} - \\ & - B(1-H) \log \frac{B(1-H)}{B(1-H) + (1-B)(1-F)} - \\ & - (1-B)(1-F) \log \frac{(1-B)(1-F)}{(1-B)(1-F) + B(1-H)} - \\ & - (1-B)F \log \frac{(1-B)F}{(1-B)F + BH} \end{aligned} \tag{2.6}$$

2.1.2 Detection Approaches

There are many approaches proposed in the literature to detect intrusions. They can be classified in three main categories: misuse, anomaly, or hybrid detection. Each of these detection approaches, together with the machine learning techniques used for anomaly detection, are next presented.

2.1.2.1 Misuse Detection

Misuse detection looks for intrusive evidence in the monitored events using previous knowledge from known attacks and malicious activity. The most common approach for misuse detection is to compare the monitored events with intrusive patterns stored in a database. These stored patterns are called signatures, and misuse detection is often called *signature-based* detection. For example, Snort [Roesch, 1999] is a NIDS which contains a huge number of publicly available signatures. The signatures follow a specific format, and allow for a deep inspection of the network packets at network (IP protocol), transport (TCP and UDP protocols) and application layer (protocols such as HTTP, FTP, SMTP, etc.).

Although signature-based is the most common approach for misuse detection, there are additional methods to represent knowledge of known attacks. *Attack path analysis* [Chen et al., 2007; Guzzo et al., 2014], for example, models the actions provoked by a potential attack in the system using several attack paths. If a monitored event follows any attack path from the beginning to the end, then it is considered intrusive.

Misuse detection works well for known vulnerabilities and attacks. Indeed, they have low false positive rates because if an activity matches a signature or follows a known attack path, then it is very likely that this activity actually has malicious intentions. However, misuse detection is not able to detect zero-day attacks. These attacks do not have an associated signature in the IDS, either because they have been discovered recently and the signatures have not been published yet [Gascon et al., 2011], or because the IDS have not been updated with the new required signatures. For example, Snort offers a set of signatures for free, but these signatures are at least one month old. Thus, free versions of Snort do not protect against potential threats appearing in the last month.

2.1.2.2 Anomaly Detection

Anomaly detectors compare monitored activity with a predefined model of normality to detect intrusions. These systems compute the model of normality by a learning process that is usually done off-line, i.e., before deployment, although recent approaches suggest the use of online training to update the model as new normal activity is

observed [Lee et al., 2011]. The monitored activity can be either network flows, service requests, packet headers, data payloads, etc. During the learning process, the system analyzes a set of normal data and computes the normal model. Afterwards, any activity that does not fit in the normal model is considered a potential intrusion. Several approaches have been proposed so far to compute the model from network data [Liao et al., 2013].

- *Statistic-based approaches* [Ariu et al., 2011] define the normal model as the probabilities of appearance of certain patterns in the training data, using thresholds and basic statistical operators such as the standard deviation, mean, co-variance, etc. In detection time, any activity that considerably differs from the learned probabilities is considered malicious. Here, the term "considerably" depends on the thresholds established, which also determines the tradeoff between false positive and detection rates.

- *Specification-based approaches* are built by experts who know how the system monitored should behave. Any activity that does not fit this behavior is considered anomalous. For example, Sekar et al. [Sekar et al., 2002] uses state-machine specifications of network protocols. The anomalies are detected whenever the state-machine does not ends the execution in a valid final state.

- *Heuristic-based approaches* automatically generate the model of normal behavior using different approaches such as machine learning algorithms [Pastrana et al., 2012], evolutionary systems [Aziz et al., 2012] or other artificial intelligence methods [Kumar et al., 2010]. This approach is probably the most extended in the research community because it provides lightweight solutions offering good results [Pastrana et al., 2012]. A more detailed explanation of machine learning for intrusion detection is given below.

- *Payload-based detectors* analyze application layer data to look for attacks [Perdisci et al., 2009; Wang, 2007]. One of the problems of using anomaly-detection for detecting malicious payloads is the difficulty of deriving features from the monitored data. A common approach is to extract n-grams from payloads to compute the model and detect anomalies [Wang et al., 2006]. An n-gram is a sequence of n consecutive bytes obtained from a longer string. The use of n-grams has been widely explored in the intrusion detection area, although

it presents some limitations too [Hadziosmanovic et al., 2012]. Moreover, the size of the vectors increases exponentially with n, which makes this method useless in some restricted scenarios.

One potential problem of anomaly-based IDSs is the need to periodically re-train the model as network traffic evolves. Online training solves this problem, but also opens the door to new threats as we discuss later. Another problem is that they still present some limitations that make them useless in real world scenarios [Sommer and Paxson, 2010], including the huge amount of false positives they produce or the difficulty to faithfully compute a model of normality. As a consequence of this, few commercial systems actually use anomaly-based approaches.

2.1.2.3 Hybrid Detection

Anomaly based detectors produce a huge amount of false positives if the model of normality is not generic enough. The alternative are misuse-based detectors, which however are unable to detect zero-day attacks and are vulnerable to polymorphism [Song et al., 2007]. In order to properly detect real-world intrusions, a combination of both techniques is necessary. Hybrid IDSs combines both misuse detection and anomaly detection. For example, in Snort [Roesch, 1999], the data preprocessors performs anomaly-based detection while decoding and generating the events, and the detection engine performs the signature matching.

2.1.2.4 Artificial Intelligence and Machine Learning

Artificial Intelligence (AI) looks for methods and procedures to provide computers with human-like intelligence. In the case of intrusion detection, because of the huge amount of data being processed in the cyberspace, it is required to use automatic tools that detect intrusions without little human intervention.

Machine Learning (ML) is a branch of AI which provides such methods. ML algorithms automatically build detection engines from a set of events performing a training process. These models are then used to detect intrusions in real time. There are two classical approaches to train the system: supervised and unsupervised. In a supervised setting, the training dataset is labeled, and the learning algorithm knows to which class each trace belongs to. Examples of supervised learning algorithms are

Decision Trees, Artificial Neural Networks (ANNs) and Support Vector Machines (SVM). An unsupervised algorithm obtains a program that is able to separate traces from different classes without knowing which the exact class of each trace is. Clustering and Correlation-based algorithms are good examples of unsupervised ML. ML techniques offer the benefit that they can detect novel differences in traffic (which presumably represent attacks) by being trained on normal (known good) and attack (known bad) traffic [Huang and Lee, 2004].

Classification algorithms build classifiers from a training data set that are used to classify events in detection time. Given a set of n samples $X = X_1, ..., X_n$ where each sample X_i is composed of j features $(F_1, ..., F_j)$, a classification algorithm generates a classifier that, for each new trace provided, returns its estimated class C_i from the set of classes $C = C_1, ..., C_k$.

Nowadays, many intrusion detection techniques proposed by the research community use ML and classification algorithms to discern between normal and intrusive data [Tsai et al., 2009]. For example, [Pastrana et al., 2012] presents a comparison of six different classification algorithms for intrusion detection in the domain of Mobile Ad-hoc Networks (MANETs). The experiments are performed using simulated network traffic, under different scenarios and conditions. Results showed that Support Vector Machines along with Genetic Programming achieve better detection accuracy than the remaining algorithms studied (Naïve Bayes, Multi-layer perceptron, Linear model and Gaussian Mixture Model).

The use of Evolutionary Algorithms (EAs) in intrusion detection aims at automatically evolving solutions that will be applied in the detection process [Sen and Clark, 2011]. EAs maintain a population of individuals, where each individual is a particular solution to the given problem. The population evolves during various generations following various procedures inspired by the laws of natural selection. Concretely, the evolution selects the best individuals to reproduce and compose subsequent generations. The selection is done regarding a fitness function, which measures how well each individual in the population performs. The selected individuals are crossed over or mutated to provide variability in the offspring individuals. After a given number of generations, or else when an optimal solution is achieved, the algorithm stops and the best individual of the last generation is given as solution. A particular case of EA is Genetic Programming (GP) [Koza, 1992] , where the individuals are programs with a tree-like shape, and the output of these programs is

used to determine the fitness function of the individuals.

In intrusion detection, whenever an alert is raised it is important to understand which event has triggered such alarm. Thus, it is easier for a security operator to determine the impact of the intrusion or to decide if it is a false alarm. Anomaly detection systems face the challenge of transferring alarms into understandable reports for the security operator [Sommer and Paxson, 2010]. Regarding this challenge, one of the main advantages of GP against other classical ML algorithms is that the programs evolved could be easy to understand and readable [Orfila et al., 2009]. Accordingly, it is easier to know how the detection is done.

2.1.3 Networks and Architectures

A large-scale coordinated attack targets or utilizes a large number of hosts that are distributed over different administrative domains, and probably in different geographical areas [Zhou et al., 2010]. These attacks have the property of targeting multiple networks or sites simultaneously, and may use evasion techniques to stealthy compromise each single network. For example, an attacker may slow down the scan in one single host by increasing the frequency of packets sent to this host. Meanwhile, it can use the time between packets to scan hosts from other networks. The main characteristic of large-scale attacks is that they usually target multiple hosts from either a single host or from many hosts. That is, the attack is distributed among various hosts.

IDNs are used in many scenarios, from collaborative domains, where different entities share information to detect global attacks [Zhou et al., 2010], to local wireless network composed by a network of sensors, like for example Mobile Ad-hoc Network (MANET) [Xenakis et al., 2011]. In both cases, the IDN is composed of multiple nodes distributed over the network where each node communicates with one or many other nodes [Patel et al., 2013]. Depending on how nodes are connected, and which are their responsibilities or roles within the network, the architecture of an IDN can be either centralized, hierarchical, or distributed. We next explain such architectures. Figure 2.2 shows an scheme of such architectures, and Table 2.2 summarizes the description of each architecture and provides examples from the literature.

In a **centralized** architecture, there is a central node gathering data from the remaining nodes in the network [Snapp et al., 1991]. The central node correlates

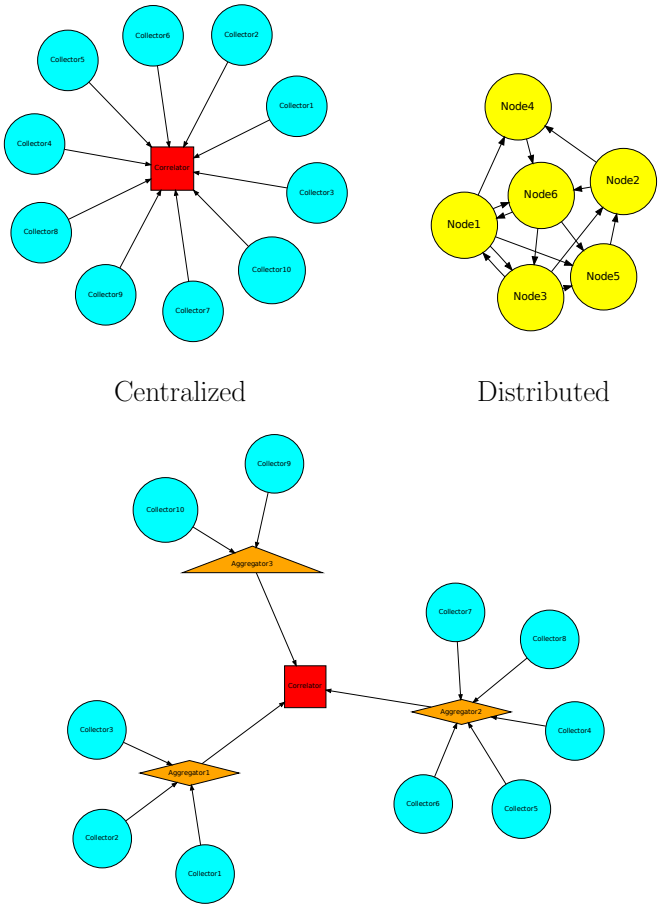


Figure 2.2: IDN architectures.

the data and emit responses. The main problem of this approach is that the central node becomes a critical point, and if it falls down (for example, due to an attack or bandwidth bottlenecks), the entire IDN falls. Moreover, the central node requires much more processing and communication capabilities, which makes this architecture useless for constrained networks like MANETs. DShield [Ullrich, 2000] is a cooperative, web-based project, where a central server receives data from multiple sources and generates security reports, such as the most trending attacks or recently discovered vulnerabilities. These reports are accessible through Internet. DShield works in a client-server model, and users can upload their logs using a web interface.

In a **hierarchical** architecture the network is organized into different levels of detection and nodes have different roles depending on their responsibilities within

the hierarchy. Each level of the hierarchy is divided into zones or clusters. In each cluster, cluster-members gather local data and provide these data to the cluster-head, and this aggregated data is then transmitted to a higher level node, who correlates. This way, a tree-based hierarchical architecture is established to cover all the network. For example, DSOC [Karim-Ganame et al., 2008] is a hierarchical IDN for protecting different networks through the Internet. DSOC considers four roles of IDN nodes: data collectors, remote correlators, local analyzers and global analyzer.

In a **distributed** architecture, the nodes share responsibilities and there are no central, critical nodes. Nodes have two main functions. First, they detect intrusions locally using monitored events within their sites. Second, nodes share data with other nodes to correlate with their local detection and thus obtain a global awareness of the network. Information sharing can be done in different ways, following a Peer-to-Peer model [Ghosh and Sen, 2005], a subscribe-publish behavior [Fung and Boutaba, 2013], etc. DOMINO [Yegneswaran et al., 2004] is a complex cooperative network that connects nodes through Internet. The nodes are connected following a distributed architecture, although each of them performs detection in local networks using local hierarchies.

There are many works in the research literature facing the problems of trust in IDNs [Gil-Pérez et al., 2013], correlation methods, detection algorithms, etc. According to a recent survey by Patel et al. [Patel et al., 2013], most of the distributed detection approaches are proposed for wireless networks such as MANETs, and few works address the problem of collaboration among entities from different administrative domains. However, this is nowadays a hot research topic which is gaining interest by the research community [Fung and Boutaba, 2013; Gil-Pérez et al., 2014; Patel et al., 2013].

2.1.3.1 Intrusion Detection Networks in MANETS

A Mobile Ad-hoc Network (MANET) is a network of mobile wireless nodes. Nodes can communicate with every other node located within a specific distance, called transmission range. When a node wants to send a packet to another node that does not belong in its one-hop neighborhood then it has to rely to intermediate nodes to forward the packets to the final destination. Thus, efficient routing protocols are required in order to optimize the communication paths. MANETs do not use a

Architecture	Characteristics	Examples from the literature
Centralized	• Several nodes are connected to a central node. • External nodes collect data, and the central node aggregates and correlates it. • The collectors send information to the central node. • The central node makes the final decision and emit responses, if needed.	• DShield [Ullrich, 2000] • DIDS [Snapp et al., 1991]
Hierarchical	• The nodes in the IDN are clustered in different levels. • Nodes in the lowest levels collect data. • Intermediate levels aggregate data from lower levels. • The highest level contains a single node who correlates data from lower levels and makes the final decision.	• DSOC [Karim-Ganame et al., 2008] • Hierarchical IDN for MANETs [Huang and Lee, 2004] • Zone-based IDN for MANETs [Sun et al., 2006]
Distributed	• Each node in the IDN is connected to one or various nodes. • There is not a single node making decisions, i.e., the detection is distributed. • There are different approaches to share information: P2P, publish-subscribe, etc.	• Peer-to-Peer model [Ghosh and Sen, 2005] • DOMINO [Yegneswaran et al., 2004] • Distributed IDN for MANETs [Zhang et al., 2009]

Table 2.2: Description of the architectures of IDNs and proposed works.

fixed infrastructure and all the nodes belonging to the network may be mobile. In MANETs there is no central node acting as an access point, and mobile nodes share the responsibility of the proper functionality of the network, since a collaborative behavior is required.

Several IDNs have been proposed to be used in mobile networks. Marti et al.

[Marti et al., 2000] presented in 2000 an IDS approach for MANETs that implements two techniques, named *Watchdog* and *Pathrater*. These are used to detect and prevent nodes performing packet dropping in the routing protocol. Every node in the network is provided with an IDN node to perform local detection and response. *Watchdog* nodes monitor their neighbors to identify malicious behavior: if the node sends a packet to a neighbor which is addressed to another node, *Watchdog* is used to observe whether the neighbor actually forwards the packet. It manages a table of failure scores for each of its neighbors, and when it observes that a packet is not forwarded by a neighbor, the score is increased. When the score of some neighbor exceeds a certain threshold, then it is considered malicious. Afterwards, whenever a new route has to be selected to send a packet to any destination, *Pathrater* uses the information from *Watchdog* to avoid routes with malicious nodes. Though the proposal of Marti et al. does not consider cooperation nor distributed detection (i.e., each node detects intrusions locally), we include it in our analysis since many distributed approaches implement *Watchdog* as basic detection function in nodes.

Zhang et al., initially in 2000 [Zhang and Lee, 2000], and later in 2003 [Zhang et al., 2003], proposed a distributed and collaborative detection architecture. Every node in the network monitors their local neighbors, locally and independently, to detect any sign of intrusion. The key idea is that they may share information to perform this search for intrusion. Each IDN node is structured in several pieces or modules. Initially, a data collection module gathers audit traces and activity logs. Then, a local detection engine analyzes the data to look for local anomalies. Two modules are responsible for performing the response actions: the local and global response modules. To share information, an extra secure communication module is used to provide trusted communications. In these approaches Zhang et al. use classifiers to detect anomalies. They use entropy and conditional entropy to describe the characteristics of "normal" traffic and classification algorithms to build models of "normal" behavior. Therefore, classifiers are trained using "normal" data, to predict what is normally the next event given the previous n events. If a detector node monitors an event which is not what the classifier has predicted, an alarm is triggered.

Huang and Lee [Huang and Lee, 2003] presented a cluster-based IDN, in order to combat the resource constraints of MANETs. The authors use a set of statistical features obtained from routing tables and apply a decision tree algorithm, C4.5, in

order to discriminate "anomalous" and "normal" traffic. This approach allows the identification of the source of the attack, if the attack occurs within one-hop. Later, in 2004 [Huang and Lee, 2004] they proposed a hybrid system where they use both specification-based and anomaly-based detection, by using a taxonomy of anomalous activities and a finite state machine, which represents the correct behavior of the Ad-hoc On Demand Distance Vector (AODV) [Perkins, 2003] protocol.

In 2003, Karchirski and Guha [Kachirski and Guha, 2003] proposed the use of multiple collaborative sensors, where each sensor acts as a lightweight mobile IDN node. Each node has a different role: network monitoring, host monitoring, decision-making and action-taking. The nodes are divided into clusters, and each cluster has a head node which monitors packets. Nodes vote to select their cluster head, based on the connectivity data received after a broadcast step. Karchirski and Guha focus on minimizing the use of resources by the IDN nodes. However, they do not give details about how the detection process is performed.

Sun et al. [Sun et al., 2003] presented a model for IDN nodes in 2003. The model is used in a collaborative approach, due to the Global Aggregation and Correlation (GACE) component. GACE components are responsible for the communication to share detection events between IDN nodes. Later, Sun et al. [Sun et al., 2006] have also dealt with the problem of cooperativeness between nodes and presented a non-overlapping zone-based IDN. In their approach, the nodes of the IDN are grouped into zones, such that some of the internal nodes of a zone act as gateways to other zones. The nodes use Markov Chains to detect intrusions and they send alarms to their corresponding gateway when they detect some abnormal activity, using the proposed MANET Intrusion Detection Message Exchange Format (MIDMEF).

Kurosawa et al. proposed in [Kurosawa et al., 2007] an approach for intrusion detection for the AODV routing protocol. Each IDN node monitors its neighbors using a vector of 3 features: the number of RREQ and RREP messages sent and received, respectively, and the average of the variations of the destination sequence numbers between each RREQ with its corresponding RREP packets. They use an adaptive anomaly-detection approach. First, the average of all vectors of a training set is computed, which constitutes the normal model. Each input sample is compared with this average using an euclidean distance. If this distance exceeds a certain threshold, the packet is considered malicious; otherwise, the sample is included in the training dataset to recompute the average (normal model) in a next slot. Thus,

in each slot the detection is adapted to possible changes in the topology. In this work, no response action is described.

Authors in [Zhang et al., 2009] propose a method where every node in the IDN cooperate to detect Black Hole attacks in the route discovery phase of the AODV protocol. The main idea is to compare the Sequence Number (SN) of the response with the SN originally posted by the receiver to detect nodes attempting a Black Hole attack. For such a purpose, they define a new packet, the SREQ, which queries for the destination of the SN original. This way, the sender can compare the SN original with the SN of the RREP received. If it does not coincide, then the intermediate node sending the fake RREP is considered malicious. The intermediate node is responsible for asking the destination for the SN, sending an SREQ packet. Due to its malicious behavior, it may not cooperate in this step to avoid being detected, by not sending the SREQ or by modifying the response. Therefore, cooperativeness between IDN nodes is needed, because neighbors of the malicious node can detect if the attacker has forged the SREP or if it has dropped the SREQ.

Sen et al. [Sen and Clark, 2009] presented different evolutive approaches to detect intrusions. More precisely, the authors use simulated networks to obtain the data they use to evolve the programs, implementing different attacks. First, in [Sen and Clark, 2009] a grammatical approach is used to detect Packet Dropping, Flooding, and Route Disruption attacks. They achieve good detection rates for the three types of attacks, but with a rather high false positive rate in the Packet Dropping and Flooding attack. They argue that this is due to packet losses that usually occur in these networks, and differentiating packet losses from malicious droppings is not an easy task. Secondly, in [Sen et al., 2010] they use Genetic Programming with a multiobjective approach to obtain programs that maximize the detection rate and minimize both the false positive rate and the energy consumption, which is one of the main constraints in MANETs. They evaluate their approach for two types of attacks, the Flooding attack and the Route Disruption attack. In both works, different intrusion detection approaches are employed for each kind of attack and, again, almost all the attacks are detected.

In 2011, Su [Su, 2011] proposed a IDN where IDN nodes monitor their neighbors in order to detect packets that are suspicious of being part of a Black Hole attack in the AODV protocol. If an IDN node observes that a node is responding (i.e., sending a RREP) to a RREQ which it has not previously forwarded, then its malicious score

is increased by 1. When this score exceeds a predefined threshold, a block message (response) is broadcast to the nodes in the neighborhood, and the malicious node is blocked by these nodes. This block message is firstly authenticated with the ID of the detector node, and carries information indicating that the packets sent by the malicious node should be ignored. Thus, authors assume that there is some authentication mechanisms to ensure that the ID of a node cannot be forged and that block messages cannot be modified or counterfeited.

Li et al. [Li et al., 2012] recently proposed a cooperative scheme where IDN nodes share their observations with their neighbors. It uses a reputation scheme. Every node in the network runs an IDN node which manages a local view of its neighborhood implemented as a list of trustworthiness about each neighbor. Agents perform local detection to observe misbehaving neighbors (for example, using the *Watchdog* technique from [Marti et al., 2000]). Then, detection events are exchanged between neighbors, and based on the received data from other IDN nodes, the local views of of the nodes are updated. Only information received from trusted nodes is considered. The main innovative idea from other similar approaches is that authors propose a multidimensional level of trust. Instead of a single value, the trust in neighboring nodes depends on three values: the collaboration trust, which measures how collaborative is a peer; the behavioral trust, obtained from the observed anomalous behavior; and the reference trust, which depends on the correctness of the reports given by the neighbor.

2.2 Intrusion Detection in Adversarial Settings

In this section, we first review the main proposals to attack IDSs. Second, we make a special emphasis on attacks focused on specific ML algorithms, as they have been widely used in the literature for intrusion detection. Finally, we review the most relevant proposals that address attacks to IDNs.

2.2.1 Early Attacks to Intrusion Detection Systems

The research on attacks against IDSs gained research attention in the late nineties, when IDSs were so sophisticated that adversaries were forced to consider them while targeting the endpoints.

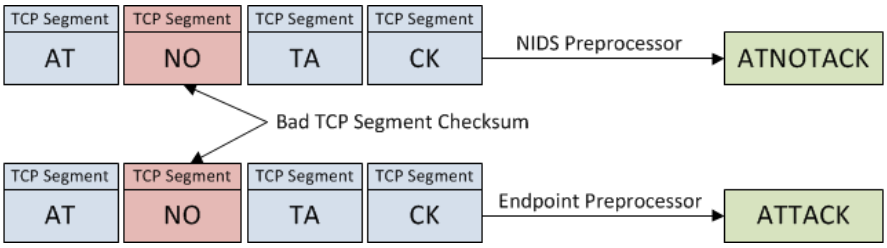


Figure 2.3: Packet insertion attack. A packet is processed by the NIDS, whereas it is not actually reaching the endpoint being monitored.

2.2.1.1 Packet Insertion and Evasion

Evasions against NIDS were first proposed by Ptacek and Newsham in 1998 [Ptacek and Newsham, 1998]. In this seminal paper, the authors highlighted the existence of some ambiguities in the TCP and IP protocols that allow different systems to implement the protocols differently. An evasion succeeds when the NIDS ignores packets which are going to be processed on the endpoints (packet evasion) or when it accepts and processes a packet which is not processed by the endpoint system (packet insertion). Packet insertion and evasion lead to different data being processed at the endpoints and NIDS, which can be used by an adversary, for example, to evade a signature matching as shown in Figure 2.3. In [Pastrana et al., 2013] we provided a detailed description of the methods that the adversary may use to evade a NIDS using these protocol ambiguities as well as the solutions proposed to mitigate the problem. These solutions mainly rely on normalizing the traffic before it reaches the NIDS [Antichi et al., 2009; Varghese et al., 2006; Vutukuru et al., 2008; Watson et al., 2004], or to configure the NIDS specifically for each endpoint operating system [Shankar, 2003] (the last solution is implemented in the popular IDS Snort [Roesch, 1999]). These solutions solve the problem of ambiguous traffic, and are rather efficient in current networks. Thus, research on attacks to IDS have turned to higher layers of the detection.

2.2.1.2 Polymorphic Worms and Mutant Exploits

The most explored technique to evade IDS is probably the modification of intrusion patterns to avoid signature matching. The first approach considered was implemented by polymorphic worms. The main characteristic of a worm is the self-replicating capability among different victims. A polymorphic worm changes its appearance

each time it propagates from one infected host to another. Indeed, many automated tools are publicly available, such as CLET, a polymorphic shellcode engine published in Phrack (a hacking community journal) [Detristan et al., 2003]; or ADMutate [Macaulay, 2007]. These polymorphic worms can effectively evade detection by signature-based IDSs [Perdisci et al., 2006]. However, polymorphic worms still contain invariant and structural similarities between different instantiations. This invariant parts are used by automatic signature generators like Paragraph [Newsome et al., 2005]. Moreover, statistical analysis of the mutated worms also allows for its identification [Kruegel et al., 2006].

In 2004, Vigna et al. presented a method to evaluate the response of different signature-based NIDSs against evasion attacks [Vigna et al., 2004]. The authors proposed an automated mechanism to generate variations of a given exploit by applying mutant operators to a predefined exploit template. As the modifications could make the exploit to become ineffective, they proposed the use of a system (an *oracle*, according to the authors) to monitor the quality of the exploit. Figure 2.4 shows the schema of the proposed framework. Using a series of mutation mechanisms and a set of exploit templates, the framework combines both sets to deterministically generate a set of mutant exploits. Then, these mutant exploits are analyzed by the external oracle in order to verify that the changes are valid (such a verification was made by checking that the application of the exploits to the target applications were successful). Moreover, the mutant exploits are presented to the analyzed NIDSs (authors experiment with Snort and RealSecure), to verify whether the mutant exploits could actually evade the detection or not.

Regarding the set of mutation mechanisms included, they used *transport layer* mechanisms, *application layer* mechanisms, and *mutation layer* mechanisms. Regarding the transport layer, they used some of the techniques presented by Ptacek and Newsham in 1998, like IP fragmentation, along with new ones, like using IPv6 instead of IPv4. They also proposed application layer mutations. Concretely, they modify FTP traffic, by inserting telnet commands in the FTP flow; HTTP traffic, generating malformed headers; and SSH traffic, inserting *NULL* records in the negotiation of the master key. Finally, as part of the so-called mutation layer, they used polymorphic shellcode and alternate encodings to directly modify the semantics of the exploits. As for the results, they were quite promising, as 6 out of 10 exploits were evaded in Snort and 9 out of 10 were evaded in RealSecure.

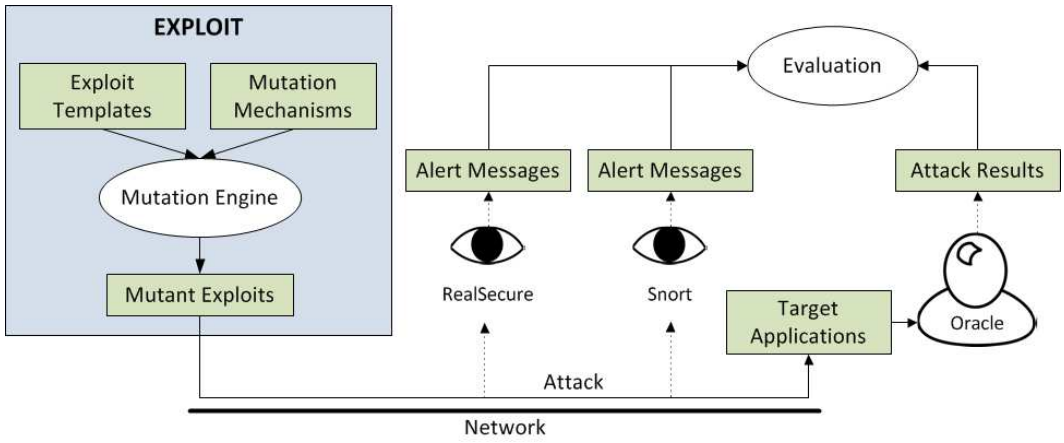


Figure 2.4: Framework for creating and testing mutant exploits [Vigna et al., 2004]

2.2.1.3 Mimicry and Blending Attacks

A polymorphic worm changes its appearance every time it is instantiated. These types of worms can effectively evade the detection of signature-based NIDS, as it is not feasible for a NIDS to manage all the different signatures of all the possible instances of a worm, even with automatic signature generators, because the complexity of these detectors is rather high. However, polymorphic worms are not classified as normal behavior, and therefore, they cannot evade anomaly-based NIDS. Next, we present the mimicry and polymorphic blending attacks, which are attacks whose aim is to appear as normal events. These attacks have been designed to evade both HIDS and NIDS.

In 2002, Tan et al. presented a novel idea to evade anomaly-based HIDS that do not take into account the arguments of system calls [Tan et al., 2002]. In particular, they showed how to evade the Stide HIDS [Forrest et al., 1996]. Authors realized that Stide looked for anomalies using a detection window size, i.e., it only detected anomalies if the number of involved system calls was smaller than the window size. Since Stide uses a detection window size, if the attacks are modified so that the abnormal sequence of system calls is larger that the window size, the attacks may succeed and still remain undetected. Authors proposed to automatically look for all the allowed sequences that, if correctly tuned, do nothing to the system and make malicious sequences larger.

Wagner et al. presented a study in 2002 to evade anomaly-based HIDSs [Wagner and Soto, 2002]. All the methods studied were related to the fact that many host-

based anomaly detectors in 2002 monitored systems call sequences. Thus, an attack can remain undetected if no system calls are made, although the damage that such an attack can cause is quite limited. Moreover, just changing the parameters of a valid system call may allow an attack to be executed undetected. This situation happened because most HIDS in 2002 did not examine these parameters when searching for anomalies.

In 2005, anomaly-based HIDSs were improved in such a way that they no longer examined only the sequence of system calls of a program, but also additional information such as the values stored in the stack, the origin of the system calls, the information about the call stack, etc. Kruegel et al. [Kruegel et al., 2005] showed that if a legitimate program containing malicious code is able to modify some memory segments, it can manage to control the flow of the program and execute pieces of the malicious code at memory locations where detection can be evaded. In particular, the authors claimed that such an operation can be achieved by directly modifying the register, the stack and the heap areas. In order for an attacker to be able to launch the aforementioned attack, she must first find a vulnerability in the code of the program to be infected (e.g., using symbolic execution) and then, find a sequence of system calls that can be executed in the program without raising suspicion. In their research, Kruegel et al. focused on the Intel X86 architecture and managed to infect a vulnerable program written in C as well to bypass the detection of two different HIDSs.

In 2005, Kolesnikov et al. [Kolesnikov and Lee, 2005] extended the idea of polymorphic worms and proposed the use of Polymorphic Blending Attacks (PBAs) against NIDS, which was later refined by Fogla et al. in [Fogla et al., 2006]. A PBA is a technique that aims to change the appearance of an attack in such a way that it blends in with the normal behavior of a network. It is therefore aimed to evade both signature-based and anomaly-based NIDS. A PBA is composed of three parts (see Figure 2.5):

- The *attack vector*, used to exploit a vulnerability of the target system successfully and thus penetrate in the target host.
- The *attack body*, which represents the core of the attack performing the malicious actions inside the victim, for example, a shellcode. It is encrypted with some simple reversible substitution algorithm using as key the substitution table.

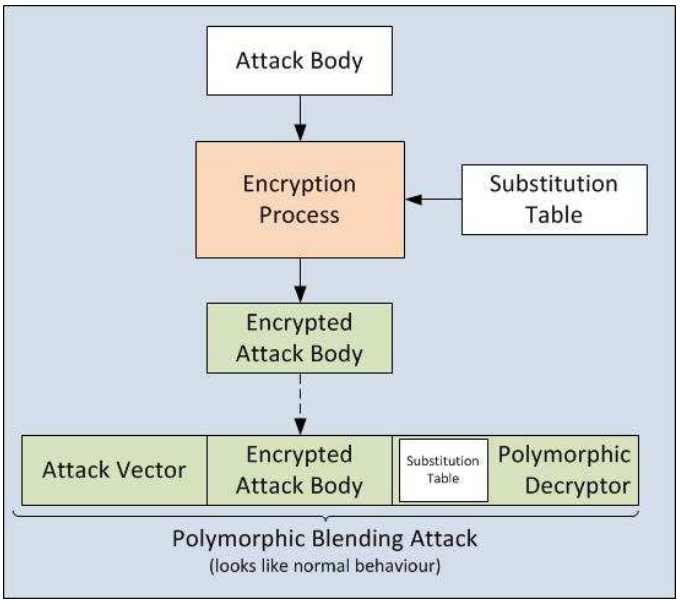


Figure 2.5: General structure of a Polymorphic Blending Attack (PBA).

- The *polymorphic decryptor*, which has the substitution table to decrypt the attack body and then transfers the control to it

The main steps involved in the generation of a PBA are described next.

1. *Learning the normal profile of the NIDS*: the authors assume that the attacker has complete knowledge of the anomaly-based NIDS to be evaded. With such knowledge, the adversary can use the NIDS learning algorithm and a set of normal traffic in order to construct a statistical normal profile similar to the one used by the NIDS.

2. *Encrypting the attack body*: in order to generate polymorphic instances of an attack vector, the attack body (i.e., the malicious code) is encrypted using a simple reversible substitution algorithm, where each character in the attack body is substituted according to a particular substitution table. The objective of such a substitution is to masquerade the attack body as normal behavior, guaranteeing that the statistical properties specified in the normal profile are satisfied (note that finding an optimal substitution table is a very complex task).

3. *Generating the polymorphic decryptor*: when the PBA reaches the victim host, the attack body must be decrypted and executed. In order to do that, a

polymorphic decryptor is required. Such a decryptor consists of three parts: the code implementing the decryption algorithm, the substitution table necessary to perform the decryption process and the code in charge of transferring the control to the attack body.

In [Fogla and Lee, 2006], the authors extended their work and proposed a formal framework to automatically generate PBAs against any statistical anomaly-based NIDS. They stated (and proved) that such NIDSs could be represented as a Finite State Automata (FSA), either deterministic (FSA) or stochastic (sFSA). As a consequence, the problem of finding a PBA for a NIDS became equivalent to finding a PBA for an sFSA. However, finding such a PBA was, as proved by the authors, an NP-complete problem. For this reason, they proposed a method to reduce the NP-complete problem to a satisfiability (SAT) problem (if the NIDS had been represented as an FSA) or Integer Linear Programming (ILP) problem (if the NIDS had been represented as an sFSA), two types of problems for which polynomial-time algorithms already existed.

In 2011, Kayacik et al. proposed a method to generate exploit mutations with which to evade any anomaly-based HIDS capable of outputting anomaly and delay rates [Kayacik et al., 2011]. In order to generate such mutations, Genetic Programming (GP) was used. GP individuals were represented by ordered sets of system calls with their arguments (if any) and evolved according to a fitness function with three objectives: increasing the attack success, minimizing the anomaly rate and minimizing the delay. Concretely, they studied an attack composed of a sequence of 3 defined system calls and arguments: *open('/etc/passwd'), write('toor::0:0:root:/root:/bin/bash')* and *close('/etc/passwd')*. If this sequence was found in the set of system calls of the individual, its fitness was increased.

They compared two approaches, a "black-box" approach, where the attacker only knows the outputs that the IDS produces from a particular set of inputs, and a "white-box" approach, where the attacker has knowledge of the internal behavior of the IDS. The results of such a comparison show that, although using the white-box approach produces exploits with lower anomaly score, the black-box technique may achieve similar rates if the exploit length is increased, even being a more difficult problem. Therefore, they concluded that in certain cases no internal knowledge of an IDS is necessary to evade it. A complementary study made in that work was to

determine the effectiveness of the approach when generating exploit mutations for a particular IDS and using the resulting mutations in another IDS. This could be the case where an attacker possesses an IDS but wants to evade another one.

2.2.2 Machine Learning Algorithms in the Presence of Adversaries

ML algorithms build classifiers from a training data set and are used to classify events in detection time. Nowadays, many intrusion detection techniques in the research community use ML and classification algorithms to discern between normal and intrusive data [Lee et al., 1999; Rieck et al., 2011; Tsai et al., 2009].

The benefits of ML are manifold. First, they are relatively easy to use and do not require much understanding about what the insights of the algorithms are. Tools such as Rapid Miner [Land and Fischer, 2012] and WEKA [Hall et al., 2009] permit users to set-up the algorithms in a black-box fashion by just providing the input dataset. Second, ML are fast and provide good results in terms of efficiency. The detection is often very efficient and consumes little amount of resources. This is a rather important aspect to detect intrusions in real time, mostly in constrained scenarios such as MANETs. Third, ML algorithms have been widely studied in the field of intrusion detection, and provide good results in terms of detection and false positive rates. At a first sight, these strengths makes ML a suitable and helpful solution for intrusion detection. However, as we discuss below, the use of ML for intrusion detection has been recently criticized by the research community and several problems regarding its use have arisen.

Dalvi et al. in 2004 [Dalvi et al., 2004] stated for the first time the problem of using ML algorithms in adversarial environments. The authors stated that the design of robust classifiers requires continuous ad-hoc reconfiguration in order to face the adversarial actions. This problem can be viewed as a "game" between the adversary and the classifier design, and thus, authors proposed a mathematical model in terms of the optimal strategy used by the adversary. Dalvi et al. assumed that an adversary has perfect knowledge about the classifier, which is a strong assumption. Lowd and Meek in 2005 [Lowd and Meek, 2005] presented a reverse engineering attack to quantify the maximal cost required by the adversary to gain enough information about the classifier in order to evade it.

A classification of attacks against ML was first proposed by Barreno et al. [Barreno et al., 2006] in 2006 and later updated by Huang et al. [Huang et al., 2011] in 2011. This taxonomy classifies the attacks regarding three aspects: the Influence, the Specificity, and the Security Violation.

1. **Influence**. Depending on the process of ML that the attack affects, it can be either *causative*, if they have influence over the training data, or *exploratory*, if it can just interact with the classifier in detection time. Causative attacks are mostly efficient if they target ML using online learning, where the classifier adapts to changing conditions through continuously retraining in detection time. For example, Biggio et al. [Biggio et al., 2012] have proposed a poisoning attack against the SVM algorithm by injecting data in the training set of an incremental solution.

2. **Specificity**. The attack can be *targeted* if it focuses on particular, small set of points, or *indiscriminate* if the adversary seeks to disturb any point from the distribution.

3. **Security Violation**. Depending on the result of attacks, these can be either *integrity* attacks, which results in false negatives (i.e., attacks which evade the classifier), or *availability* attacks, aiming to generate so many false positives that the classifier becomes unusable. Huang et al. in 2011 [Huang et al., 2011] proposed a new Security Violation class which is the *privacy* attacks. In these attacks, the adversary aims to reveal any information related to the classifier, such as the ML algorithm used, the data distribution, etc.

In 2010, Sommer and Paxon presented a paper where they discussed the use of ML in the field of anomaly detection of network intrusions [Sommer and Paxson, 2010]. They claimed that detecting real-world attacks in real scenarios is such a complex task that cannot be performed automatically with ML algorithms. They also stated that using ML for intrusion detection should be done carefully, as there are several differences with some other ML applications, like product recommendation, spam detection or natural language translation. Authors analyze five main problems that system designers must take into account:

1. **Outlier detection**, i.e., the lack of "intrusive" examples in the training phase. Training a system with ML requires data with high representation of all classes.

2. **High cost of errors**, i.e., the need of achieving a high detection rate while having a low false alarm rate. In other areas, an error may comprise an spam arriving to the client email account or missing a potential client. However, a successful attack in a system may have tragic effects.

3. **Semantic gap**, i.e., the problem of providing security administrators with a good understanding of the alarms. ML algorithms are able to discern between classes. However, classical algorithms cannot explain why a given instance has been classified as its related class. Thus, a system administrator who wants to know what happened when analyzing an alert should not have extra information, which is usually needed.

4. **Diversity of network traffic**, i.e., the problem of faithfully representing the real world in the training phase. Due to the complexity and variety of current networks, even with a huge training dataset it is not possible to assure that the system has dealt with all the possible scenarios.

5. **Difficulties with evaluation**, i.e., the lack of publicly available datasets to experiment. System designers often use simulated traffic which do not correspond with real scenarios. Additionally, using real data recorded in some institution or network can reveal sensitive data, leading to privacy concerns.

In 2011, Nelson et al. quantified the effort required for an adversary to find near-optimal evasions [Nelson et al., 2011]. A near-optimal evasion is the attack performed by the adversary against an ML algorithm which require the lowest number of queries. The quantification of such near-optimal evasion provides the complexity required by an adversary to evade a classifier, which can be used to design robust classifiers.

Recently, Biggio et al. [Biggio et al., 2013] have proposed a framework for empirical assessment of the robustness of classifiers to attacks. The framework requires the definition of four different components:

1. **Adversarial model**. It defines the goals, capabilities and strategies assumed for the adversary. The goal is defined in terms of the taxonomy proposed in [Barreno et al., 2006; Huang et al., 2011]. These capabilities include the attacker knowledge about the classifier -concretely about the training data-

the feature set, the learning algorithm, the decision function, and the feedback from the classifier. The capabilities also include the skills of the adversary, i.e., the control that it has over the training and test datasets. Finally, the attack strategy refers to quantification of the optimal evasions, e.g. how features are modified from the vector space, how many samples are modified, etc.

2. **Data model**. Depending on the assumptions given for the adversary, the distribution of the data may change differently. The data model establishes the degree of change in the distribution of both the training and detection data used by the classifier due to the adversarial activity.

3. **Generation of training and testing sets**. This component proposes an algorithm to generate training and testing datasets that uses the data distribution derived from the previous component. These datasets are then used to train and test the classifier to perform the evaluation under attack.

Regarding the countermeasures to attacks against ML algorithms, Barreno et al. [Barreno et al., 2006] provided three mechanisms that could be used to counteract adversaries. First, *regularization* allows to train classifiers that have some prior knowledge, thus reducing the complexity of the classifiers and encode expert knowledge which may be useful for the detection. Second, *disinformation* and *information hiding* prevents the adversary to deducing which the decision boundary of the classifier is. For example, by do not providing feedback about the result of the detection, the learner prevents from probing attacks. Third, *randomization* of some components of the detection also makes harder the task for adversaries, as it does not know which the exact boundaries of the detection are.

Another method proposed to harden the task to adversaries is the use of multiple classifiers [Biggio et al., 2010; Giacinto et al., 2003]. The main idea is to combine several classifiers for the detection, even applied to different set of features. The combination can be done in several ways. For example, the IDS McPAD [Perdisci et al., 2009] uses different classifiers based on Support Vector Machines but which construct features differently. McPAD combines the classifiers randomly choosing a subset of them. Given the output of these selected subset, authors propose four different combination methods to get a final anomaly threshold: calculating the average, calculating the product, choosing the maximum or choosing the minimum.

The analysis of the state of the art in the adversarial machine learning turns into the following conclusions:

- The use of ML for intrusion detection must consider the adversarial setting where IDS are placed. Indeed, IDS normally are the first barrier that attackers find, and thus it is likely that they will try to attack them.
- The classical design of ML algorithms has focused on efficiency and efficacy of the results, and it has considered a common data distribution in both training and testing phases. However, this is clearly erroneous in the case of intrusion detection.
- The taxonomy of attacks presented in [Barreno et al., 2006; Huang et al., 2011] and the framework to evaluate classifiers presented in [Biggio et al., 2013] open new research horizons in the field of robust design of ML algorithms, that can be generalized to the intrusion detection area.
- It is rather impossible to consider a perfect classifier providing good detection performance while being robust against attacks. Thus, it is necessary to understand the adversarial model and consider what, how and where the attacks are being happening in order to set additional countermeasures and maintain a certain level of security in the protected systems.

2.2.3 Attacks to Intrusion Detection Networks

IDNs are complex defense mechanisms that detect and counteract distributed, sophisticated attacks against distributed organizations or entities. This makes them an attractive target for attackers. Thus, besides performance requirements such as accuracy and efficiency, features such as resilience against attacks are becoming increasingly critical in order to maintain an acceptable level of security even in the presence of adversaries. Few works have dealt with the problem of adversarial capabilities in IDNs.

As stated by Bye et al. [Bye et al., 2010], research on IDNs can be divided into two main components: the *detection aspect* and the *framework aspect*. The first includes the internal procedures implemented in the nodes to process the data, such as detection functions, correlation algorithms, data aggregation, trust management,

etc. The attacks presented in the previous section for IDS are applicable to the *detection aspect* stated by Bye et al., though little attention has been paid to the *framework aspect*. Bye et al. define five building blocks that must be considered for the framework of IDNs:

1. **Communication Scheme**. It indicates how nodes communicates between them. This scheme defines the architecture of the network.

2. **Group Formation**. How nodes are aggregated in the network. Depending on the network, creating "teams" intended to accomplish specific missions is useful to divide tasks.

3. **Organizational Structure**. It determines whether the nodes have the same responsibility, or if there are nodes having more competences than others.

4. **Information Sharing**. It defines the format and contents of messages interchanged. Nodes may exchange local data collected by sensors or knowledge about intrusion events detected.

5. **System Security**. This block considers the security of the IDN itself. Concretely, three factors are considered: trust management, which is defined to deal with malicious insiders; access control (P2P, publish/subscribe, central authorities, etc.); and availability, to define continuity plans even in presence of attacks such as distributed denial of service (DDoS).

From the five blocks above, only the last considers robustness of the network itself against adversaries. In their work, Bye et al. define an adversarial model with adversaries only capable of performing query/response analysis. This is a rather useful attack for an adversary, as it is used to gain information about the functions (as the attack presented in Chapter 4) and locations of the nodes. Accordingly, authors propose a robust design for resilient IDNs dealing with privacy localization attacks. However, authors do not consider other adversarial capabilities, such as modification of data or packet insertion. Thus, the adversarial model presented in [Bye et al., 2010] is incomplete.

In 2011, Fung [Fung, 2011] defined a set of "insider attacks", i.e., attacks where the adversary has gained access into the IDN. Fung introduces four attacks. First,

the *sybil attack*, where the adversary uses fake identities to make false reports and gain influence in the IDN. Second, the *newcomer* attack, where the adversary changes its identification information once it is detected. Third, the *betrayal* attack, where a benign node turns into malicious and sends false information attacking other nodes. Four, the *collusion* attack, where different malicious nodes collaborate to attack the IDN. Regarding these attacks, Fung provides a review of the main proposals for IDNs, analyzing the robustness of each proposal against the presented attacks. The work by Fung provides an overview of specific attacks, and the likelihood of happening in the architectures proposed in the literature. However, it does not take into account general adversarial capabilities.

Xenakis et al. [Xenakis et al., 2011] surveyed the weaknesses of IDNs architectures for Mobile Ad Hoc Networks (MANETs). The main purpose of authors is to point out flaws in the design of architectures in many aspects, including the ratio of false positives, communication overhead incurred in the network, processing requirements for the nodes performing detection, etc. However, they do not expose weaknesses derived from specific attacks targeted against the different proposals.

2.2.4 Taxonomy of Attacks

A recent survey by Corona et al. [Corona et al., 2013] identifies six general categories of attacks against classical IDS (see Table 2.3). In this Thesis we consider this taxonomy to categorize the attacks. The attacks are classified regarding the goal of the adversary, which results in different consequences:

1. **Evasion**, where an attack is carefully modified so that the IDS would not be able to detect it. These are the most common attacks studied in the literature. For example, blending and mimicry techniques are examples of evasion.

2. **Overstimulation**, where the IDS is fed with a large number of attack patterns to overwhelm analysts and security operators. For example, *Mucus* is an IDS stimulation tool by Mutz et al. [Mutz et al., 2003] that generates packets that purposely matches the signatures of Snort [Roesch, 1999] to generate a large number of detection alerts.

3. **Poisoning**, where misleading patterns are injected in the data used to train or construct the detection function. This attack is applicable to IDS that use

retraining, i.e., that modify the detection function in detection time [Rubinstein et al., 2009]. An example of such attacks are the Allergy Attacks [Chung and Mok, 2006, 2007], which targets automatic signature generators such as Polygraph [Newsome et al., 2005]. These attacks insert noisy data into the generation process to generate signatures in the IDS that filter out normal requests.

4. **Denial-of-Service (DoS)**, where the detection function is disabled or severely damaged. Algorithmic complexity attacks [Crosby and Wallach, 2003] are examples of such attacks. These attacks force the IDS to perform the worst case scenario, for example by generating packets that make the signature matching to generate the highest number of matches [Smith et al., 2006].

5. **Response Hijacking**, where carefully constructed patterns produce incorrect alerts so as to induce a desired response. This attack directly targets the response module of a system. For example, in a MANET, several colluding malicious nodes may send false reports indicating bad behavior of a benign node [Fung and Boutaba, 2013]. An IDN node then may block or ban such benign node from the network.

6. **Reverse Engineering**, where an adversary gathers information about the internals of the IDS by stimulating it with chosen input patterns and observing the response. The common approach is to perform query-response analysis, for example to discover signatures used by IDS [Mutz et al., 2005]. As part of the contributions of this Thesis are the reverse engineering attacks presented in Chapters 3 and 4.

Corona et al. focus their work on IDS working in a stand-alone fashion, i.e., without communicating with other systems. These systems have a common structure, depicted in Figure 2.1, i.e., decoders, preprocessors, detection engine, and response module. However, the taxonomy does not provide general mechanisms an adversary can use to achieve her goals. Moreover, the survey focuses on stand-alone IDS, not considering specific attacks to IDNs. Typically, these specific attacks are focused on the communications between nodes and the propagation of the information in the network.

Attack	Goal	Examples from the literature
Evasion	To bypass the detection of the IDS	• Polymorphic Blending Attacks [Fogla et al., 2006; Kolesnikov and Lee, 2005] • Packet Insertion [Ptacek and Newsham, 1998] • Mutant Exploits [Vigna et al., 2004]
Overstimulation	To stimulate the IDS and force it to generate a huge amount of alarms	• IDS stimulator [Mutz et al., 2003]
Poisoning	To modify the detection function or any component in the IDS	• Attacks to signature generators [Newsome et al., 2006; Perdisci et al., 2006; Rubinstein et al., 2008] • Allergy Attacks [Chung and Mok, 2006, 2007]
Denial of Service	To make the IDS unavailable or useless	• Algorithmic complexity attacks [Crosby and Wallach, 2003; Smith et al., 2006]
Response Hijacking	To force the IDS to generate specific responses useful for the adversary	• IDS stimulator [Mutz et al., 2003] • Sybil and Betrayal attacks [Fung, 2011]
Reverse Engineering	To obtain information about the functioning or internal components of the IDS	• Reverse Engineering of Network Signatures [Mutz et al., 2005] • Traffic analysis [Bye et al., 2010] • Query-response analysis [Pastrana et al., 2014]

Table 2.3: Taxonomy of attacks proposed in [Corona et al., 2013].

2.3 Discussion

In this chapter, we have analyzed the state of the art in the area of attacks against intrusion detection solutions, including stand-alone IDSs and IDNs. Misuse detection has proven inappropriate to detect sophisticated threats such as polymorphic attacks, which are included in many existing APTs and other malware distribution mechanisms. Research in the IDS field has been trying to move towards anomaly detection approaches for more than a decade. However, anomaly detection has also many drawbacks, including the difficulty to generate appropriate models of normal behavior. Machine learning is currently the best available technique to face this problem, and a very significant number of works have explored the use of many machine learning algorithms for intrusion detection.

The research community has recently questioned the security of machine learning algorithms when adversarial environments are considered. The design of resilient machine learning algorithms is an active research topic in which little progress has been made over the last few years. One key limitation is that most proposals are discussed using artificial datasets, and it is unclear that they will work in real scenarios. A recent framework by Biggio et al. [Biggio et al., 2013] suggests that, in order to test the resilience of classifiers against attacks, the artificial datasets used for training and testing should include potential modifications performed by an adversary. Still, there is a substantial lack of experimental work exploring the problems derived from an attacker who can modify instances at will to subvert the detection function.

In order to secure IDSs against attacks, many state of the art solutions have proposed using secret random components in the detection process [Wang et al., 2006][Perdisci et al., 2009][Huang et al., 2011]. For example, Anagram [Wang et al., 2006] randomly partitions each incoming payload into a number of non-overlapping subsets using a random mask. Thus, the authors state that a potential adversary could not know which parts of the payloads are being processed and, therefore, will be unable to perform the attack. Despite such claims, it is still unclear whether these solutions improve the security against current attacks.

More generally, a prudent practice in security engineering is to design systems that remain secure—or, at least, whose security degrades smoothly—even if some integrating parts are compromised. IDNs are key components of current cyberdefenses

and should be designed according to this principle. Thus, even if components such as local sensors (IDSs) are attacked, the overall detection network should still perform well. While strong analysis models for attacks on individual IDN nodes have been explored, almost no research works have focused on the study of resilient IDNs in the face of adversaries.

3

Attacks on Machine Learning Based IDSs

3.1 Introduction

Machine Learning (ML) algorithms have been largely used in the intrusion detection field [Pastrana et al., 2012; Song et al., 2013; Tsai et al., 2009], particularly for anomaly-based detectors. Basically, these algorithms learn a model from a training set composed of attack-free samples. The obtained model is later used during the detection process to classify activities as normal or malicious.

Most ML algorithms require a representation of the data in the form of a vector of nominal or numerical features. In network intrusion detection, feature construction is the process that converts raw traffic data into feature vectors. Feature construction at the network layer may use session attributes (e.g., number of packets sent and received, duration of the session, etc.) or fields extracted from packet headers (e.g. sequence numbers, windows sizes, etc.). Moreover, feature construction at application layer can be also performed using session attributes. For example, a recent work by Goseva et al. [Goseva-Popstojanova et al., 2014] use session-based features to characterize web traffic. Authors compare C4.5, CART and SVM to distinguish between vulnerability scans and actual attacks to web servers. However, when applying ML to detect malicious payloads at the application layer, the feature construction process is not trivial. A common approach is to use text processing methods, like n-grams [Torrano-Gimenez et al., 2012; Wang et al., 2006; Wang and Stolfo, 2004], which obtains all the words of size n from the payload. To reduce the dimensionality of the data space and obtain more lightweight classifiers, some proposals set n to 1.

ML algorithms consider a similar distribution of the training and test datasets. Most frequently, the main goal for IDSs is to maximize the detection rate while minimizing the false positive rate. Accordingly, in intrusion detection ML algorithms have been applied with the main purpose of optimizing these rates, and not much attention has been paid to actually protecting these systems against adversaries interested in manipulating the classification process. Only recently, the research community has focused on designing robust ML algorithms [Biggio et al., 2013; Huang et al., 2011] for adversarial environments, but still many current proposals do not consider such settings. However, since the deployment of IDS is done in adversarial scenarios, when designing an IDS it is critical to evaluate not only its effectiveness, but also the security and robustness of the detection. Sommer and Paxson discussed the appropriateness of using ML for IDS [Sommer and Paxson, 2010]. One of the problems there stated is that network traffic exhibits a great variability even within a single network, which is wider at the application layer where the payloads are encapsulated. Thus, it is unfeasible to get a dataset that properly represents all the network data space to train the IDS.

In this chapter we discuss and analyze the use of four classical ML algorithms for malicious payload detection in the presence of adversaries. We provide experimental results that confirm the problem of using an artificial dataset for training an IDS, and show that ML-based IDS that are trained with non-representative datasets are vulnerable to attacks. Specifically, it is shown that the models learned work well for the specific distribution of the training dataset (i.e., the IDS is efficient). However, they may also learn specific rules or pattern of this distribution. An adversary who infers these irregularities will be able to attack the system by properly modifying some features (i.e., the IDS is not robust).

Even when the adversary gets feature vectors that evade the system, she still has to build raw payloads from these vectors to obtain real world evasions. This requires to invert the feature construction process, which is only possible if the feature construction has an inverse function. This challenge has been recently stated by Huang et al. [Huang et al., 2011], who questioned "*how the feature mapping can be inverted to design real world instances*". In this work, we show that if the feature construction is simple and lightweight, then it is easy for an adversary to obtain its inverse function. Thus, the process of finding real world evasions from the feature space is easier. For example, if the construction from payloads uses 1-grams, the

adversary only needs to include or remove single bytes wherever she chooses into the payload.

The experiments discussed in this chapter were conducted using the CSIC 2010 Dataset [Torrano-Gimenez et al., 2010], which is composed of real HTTP traffic data with both normal and anomalous requests. A reverse engineering attack is first presented, where the adversary discovers the main features used in the detection process and how they are used. The attack uses Genetic Programming [Koza, 1992] (GP) to generate models over the traffic distribution that resemble the behavior of IDS classifiers built from training datasets following a similar distribution. Subsequent analysis of the obtained models suggests mechanisms to evade these IDSs. The process is illustrated with four representative IDSs implementing different classification algorithms, concretely two decision trees (C4.5 and CART), Support Vector Machines (SVM) and Multilayer Perceptron (MLP). These algorithms have been widely used in the field of intrusion detection (see, e.g., [Goseva-Popstojanova et al., 2014; Pastrana et al., 2012; Torrano-Gimenez et al., 2012; Tsai et al., 2009]).

The remaining of this chapter is organized as follows. Section 3.2 describes the adversarial model assumed for the attacks presented in this chapter. In Section 3.3 we present the experimental framework used to demonstrate the attacks, which are then described in Section 3.4. The main experimental results are analyzed in Section 3.5, and Section 3.6 discusses potential countermeasures and open problems. Finally, Section 3.7 concludes the chapter.

3.2 Adversarial Model

In the analysis of attacks and countermeasures against a system, it is important to establish the capabilities assumed for an adversary. Indeed, depending on these capabilities, different procedures are established in the design of countermeasures, which is critical in order to avoid spending unnecessary resources. Since intrusion detection systems have only been analyzed in adversarial environments very recently, there is a lack of widely accepted adversarial models. Despite this, most works in this area assume an adversary with, at least, the capabilities described next [Biggio et al., 2013]. The attacks presented in this work assume that the adversary has knowledge about the following information:

1. *The distribution of the training data used by the IDS.* This does not mean that the adversary has the same training dataset, but she must know the distribution and characteristics like the protocol used, type of traffic, normal contents, common patterns, etc.

2. *The Feature Construction method (FC).* We assume that the adversary knows the algorithm used to generate feature vectors from the raw payloads. Thus, the adversary knows how the payloads are mapped into the classifier's feature space.

Both the distribution and feature construction method may be kept secret in many cases. In this work we do not consider the problem of how to get this information. However, from the security point of view, this possibility cannot be underestimated, and the security of the system should not reside in the obscurity of its implementation. Indeed, many authors have assumed that this information is known by an adversary [Biggio et al., 2013]. In fact, in [Huang et al., 2011] authors stated that *"specialized features constructed for a specific learning problem [...] may not be known or inferable by the adversary"* but *"knowledge about the training and evaluation data used by the algorithm [...] may be available to the adversary because of actions the adversary makes outside the system [...] or because the adversary is an insider"*. Moreover, recent works (see, e.g., [Ateniese et al., 2013]) have proposed reverse engineering attacks that disclose statistical properties and information about the training set of a classifier.

3.3 Experimental Setup and Base Classifiers

This section describes the setup and assumptions made in our experimental work. First, Section 3.3.1 explains the data used, its content, and the feature construction method used. Next, Section 3.3.2 details the classification algorithms studied, how we have constructed the experimental IDSs and measured their performance in terms of detection and false alarm rates.

3.3.1 Dataset

A major issue in intrusion detection is the selection of an adequate dataset to evaluate proposals. For example, the DARPA dataset [Lippmann et al., 2000] has been widely used for intrusion detection. However, it has been criticized by the research community [McHugh, 2000], mainly because it is outdated and does not include many of the current attacks, what makes it inappropriate for the detection of web attacks. In this work, we use the CSIC 2010 HTTP dataset [Torrano-Gimenez et al., 2010]. This dataset has been successfully used for web detection in previous works, such as for example [Nguyen et al., 2013; Torrano-Gimenez et al., 2009].

The dataset contains traffic targeted to a realistic web application developed for this purpose. It consists of an e-commerce web application running on Apache Tomcat, and it is composed of several web pages that allow users to do actions such as buying items with a shopping cart or registering by providing their personal data. Some of the web pages require certain parameters, for example the user name and address for the registration process or the name of the product that the user wants to buy.

The traffic contained in the dataset was automatically generated and contains normal and anomalous requests targeted to all the web pages of the application, using different values for those web pages that accept different parameters. In total, 36,000 normal requests and more than 25,000 anomalous requests are included in the dataset. All the requests are labeled either as normal or as anomalous. The dataset includes modern web attacks such as SQL injection, buffer overflow, information gathering, CRLF injection, Cross Site Scripting (XSS), server side include and parameter tampering. Further details about the dataset can be found in [Torrano-Gimenez et al., 2010].

The complete dataset is composed of 61,065 traces. We randomly divide it into three subsets. The first subset is used to train and build the classifier at the core of the IDS using the corresponding algorithm (details of the construction process are given below). The second subset is used to test each IDS and to train the GP models as part of the reverse engineering attack. These are independent processes and therefore we use the same subset. Finally, the third subset is used to test the GP models.

The dataset uses two FC methods: 1-grams and expert knowledge. Although

Table 3.1: The 89 non-null 1-grams from the HTTP dataset CSIC 2010.

Features extracted with N-grams					
!	”	#	$	LF	Space
%	:	&	’	(	)
*	+	,	-	.	/
0	1	2	3	4	5
6	7	8	9	;	<
=	>	?	@	A	B
C	D	E	F	G	H
I	J	K	L	M	N
O	P	Q	R	S	T
U	V	W	X	Y	Z
_	a	b	c	d	e
f	g	h	i	j	k
l	m	n	o	p	q
r	s	t	u	v	w
x	y	z	—	~	

the dataset uses these two FC methods, the evasion attacks studied here focus on the features extracted with the 1-grams method, in order to show its weaknesses. Torrano et al. [Torrano-Gimenez et al., 2012] show that using a combination of these two methods increases the effectiveness of the detection. However, in this work we study if this combination improves the robustness of the IDS against adversarial attacks.

The FC method based on the 1-gram method for intrusion detection works as follows: every HTTP request p is associated with a feature vector

$$x_p = (x_1, x_2, \ldots, x_{256})$$

where x_i contains the number of occurrences of the i-th 1-gram in the method, path, and arguments of p.

An analysis of the 1-grams extracted from the dataset indicates that from the 256 possible features (i.e., the total number of ASCII characters), only 89 (34.77%) appear one or more times in the HTTP requests. These 89 characters are listed in Table 3.1.

The FC process based on expert knowledge considers 28 features relevant for

Table 3.2: The 28 features constructed using expert knowledge from the HTTP dataset CSIC 2010.

Features extracted with expert knowledge	
Length of the request	Length of the path
Length of the arguments	Length of the header "Accept"
Length of the header "Accept-Encoding"	Length of the header "Accept-Charset"
Length of the header "Accept-Language"	Length of the header "Cookie"
Length of the header "Content-Length"	Length of the header "Content-Type"
Length of the Host	Length of the header "Referer"
Length of the header "User-Agent"	Method identifier
Number of arguments	Number of letters in the arguments
Number of digits in the arguments	Number of 'special' char in the arguments
Number of other char in the arguments	Number of letters char in the path
Number of digits in the path	Number of 'special' char in the path
Number of other char in path	Number of cookies
Number of distinct bytes	Entropy
Number of keywords in the path	Number of keywords in the arguments

the detection process. These are listed in Table 3.2. Some of these 28 features refer to the length of different parts of the request, as length is important for detecting attacks such as buffer overflows. Non-alphanumeric characters are present in many injection attacks. Therefore, four types of characters are considered: letters, digits, non-alphanumeric characters that have a special meaning in a set of programming languages (referred in Table 3.2 as "special" char), and other characters. Several features are created by analyzing their occurrences in both the path and the argument values. Another feature is built by studying the entropy of the bytes in the request. Additionally, the construction by expert knowledge also considers keywords of several programming languages that are often used in injection attacks, and count the number of occurrences in the path and the arguments of the request.

Overall, the 89 features obtained from the 1-gram method and the 28 features from the expert knowledge are combined for each payload, resulting in vectors with 117 features. Since in this work our aim is to analyze the weaknesses of the 1-gram construction process, the evasion attack focuses only on the 89 features constructed using 1-grams, although the reverse engineering attack focuses on the overall set of

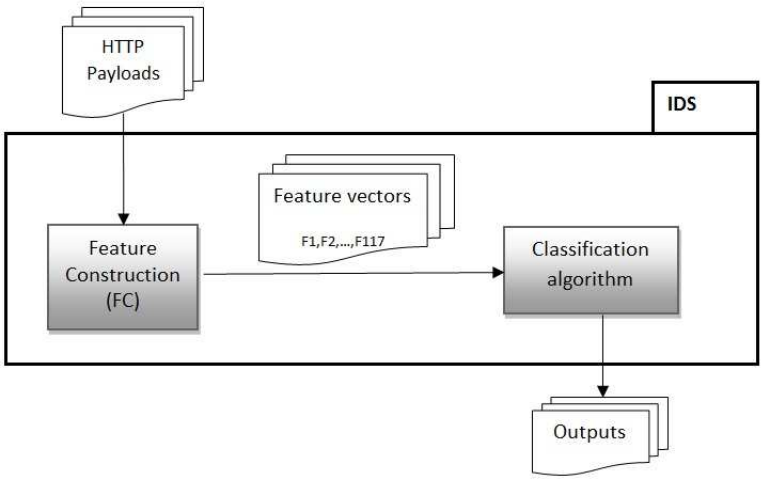


Figure 3.1: Structure of the IDS studied. It is composed of a FC module and the classification algorithm at the core of the detection engine.

features. We explain these attacks in detail in Sections 3.4.1 and 3.4.2.

3.3.2 Classification Algorithms

Figure 3.1 shows the structure of the IDS built to demonstrate our attacks. First, the HTTP traffic is preprocessed as explained in the previous section in order to extract a feature vector for each HTTP payload. Four classification algorithms are used as detection engines: C4.5 and CART decision trees, SVM, and MLP. The IDS is trained to classify HTTP packets using labeled data with both normal and intrusive packets, using the first part of the dataset.

Each classification algorithm is tested with the second part of the dataset. As output of the testing process, the classification algorithm indicates for each test request whether it has been correctly classified by the detector or not. In order to evaluate the effectiveness of the IDS, we first define the measures used. On the one hand, a real attack can be detected, obtaining a True Positive (TP), or not detected, being a False Negative (FN). On the other hand, a normal event (non-malicious) can be signaled as normal by the detector, being a True Negative (TN), or can be classified as an attack, resulting in a False Positive (FP). Given these definitions, several metrics are defined, as explained in Chapter 2:

- Hit rate (H). It is the ratio of attacks that are properly detected by the system.

Table 3.3: Detection rate (H), false alarm rate (F) and C_{ID} index of the classification algorithms studied.

	C4.5	CART	SVM	MLP
H	0.96	0.97	0.95	0.96
F	0.04	0.07	0.06	0.12
Cid	0.73	0.72	0.67	0.62

It should be maximized.

- False positive rate (F). It is the ratio of normal events wrongly classified as intrusions. It should be minimized.
- C_{ID} index [Gu et al., 2006]. As explained in Chapter 2, the C_{ID} measures the amount of uncertainty of the input resolved once the IDS output is obtained, and provides a single scalar to assess the efficacy of the IDSs.

Table 3.3 shows the effectiveness of the different IDS classification algorithms studied over test data. In the table, bigger H and C_{ID} values and lower F value means better effectiveness of the classifier. As it can be observed, the best classifier is the C4.5, with a 96% of Detection Rate (H) and a 4% of False Positives Rate (F). In general, it can be observed that the classifiers obtain high detection rates with acceptable false alarm rates.

3.4 Attacks

This section first describes a general technique to conduct reverse engineering attacks against IDS based on classifiers such as those described above. The scheme is based on the idea of obtaining an approximation of the inner workings of the classifier (i.e., of its decision surface). Such an approximation is then used to search for strategies that evade it, which are then mapped into real world payloads.

Figure 3.2 shows the overall process of the attacks. The top of the image corresponds to the behavior of the IDS. First, the IDS gets some training data with a given data distribution. The feature construction (FC) obtains the feature vectors from the raw data, which are then used to train the classifier using the corresponding ML algorithm. The classifier is used in detection mode to evaluate new events and outputs a response (normal or intrusion).

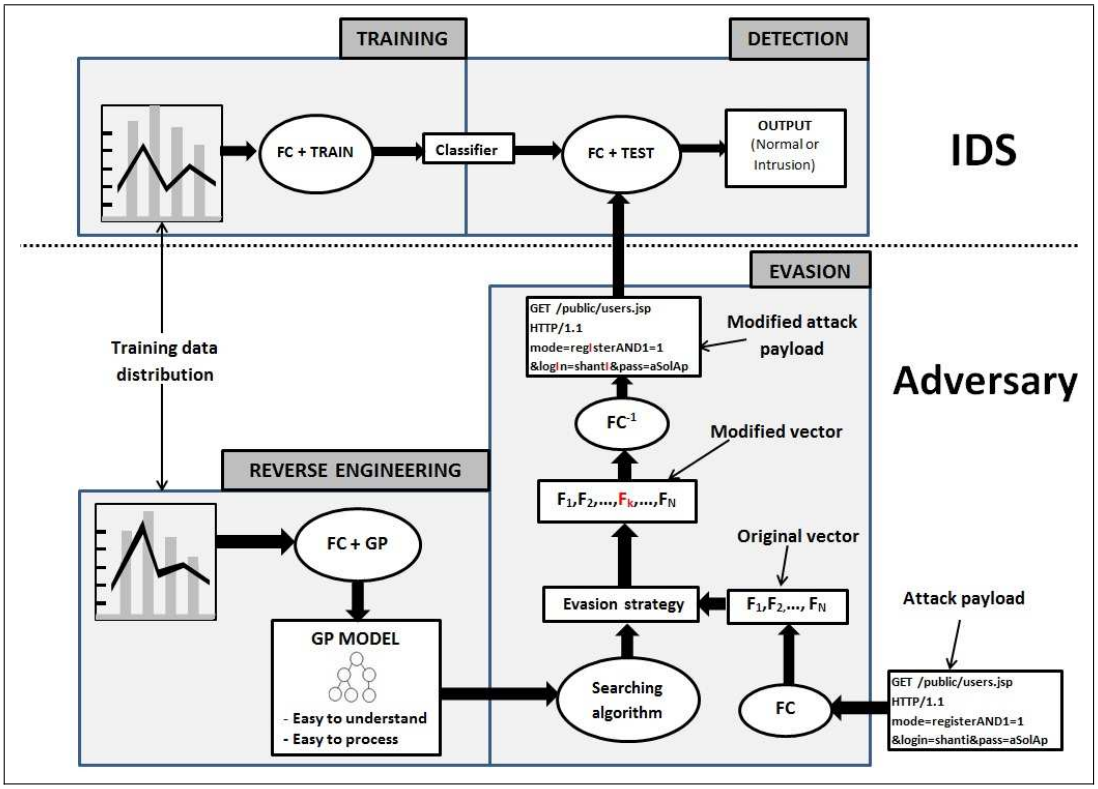


Figure 3.2: Illustration of the attack performed by an adversary to evade IDS that use an invertible FC method and ML algorithms.

The bottom part of Figure 3.2 represents the actions performed by the adversary in each attack. Given an attack payload, the adversary first performs a reverse engineering attack (bottom-left part of the figure), which obtains a GP model. Then, this model is processed to search for evasion strategies (bottom-right part of the figure). In the following sections we detail the reverse engineering and the evasion attacks .

3.4.1 Reverse Engineering Attack

We next present an attack aimed at reverse engineering the inner workings of the classifier at the core of the IDS. This is done using the data distribution used for training the IDS that, as explained before, is known by the adversary. Recall that we also assume that the adversary knows the feature construction algorithm used by the IDS. Our main hypothesis is that, under these circumstances, the adversary can generate training samples and build another classifier that is a good approximation,

in terms of the decision surface, to the actual IDS classifier; that is, the model built by the adversary classifies instances similarly to how the IDS does it. Such a model is subsequently used to perform evasion attacks. We conduct our experiments over the IDS created using 1-grams with the four different classification algorithms explained above (C4.5, CART, SVM and MLP). This attack is represented in the bottom-left part of Figure 3.2.

In this work, we use Genetic Programming (GP) [Koza, 1992] to obtain an approximation of the decision surface of the actual detection model at the core of the IDS. We choose GP because it outputs tree-based expressions that can be simply evaluated with a recursive function. Moreover, these models are easy to understand and readable [Orfila et al., 2009]. However, the presented reverse engineering attack works with any algorithm whose output is readable and could be processed by a posterior searching algorithm.

Given a search problem over a large solution space, GP performs a heuristic search to obtain a locally optimal solution. GP is a technique that keeps a set of programs (also called the population of individuals), randomly initialized, which are evolved according to various procedures inspired by the laws of natural selection. In our scheme, each program (individual) has a tree-like structure where the root and intermediate nodes are mathematical and logic functions, and the leaves are terminal features (see the example in Figure 3.3). Each generation is obtained by selecting the best individuals from the previous one. Some individuals are mutated (changing an internal subtree by another) or subject to crossover (exchanging subtrees from two different individuals), according to a set of parameters. After a given number of generations, or else when an optimal solution is achieved, the algorithm stops and the best individual of the last generation is given as solution. Table 3.4 shows the values used for the GP parameters in our experimentation. These values have been obtained using 10-fold cross-validation and using the combination of parameters that performs best in terms of accuracy.

The operators used in the GP algorithm are shown in Table 3.5. Most of them output a binary value. Therefore, the final output of each individual is binary, indicating whether the input trace is considered an attack or not. If the final output is not binary (i.e., the root of the tree is a non-binary function), then an output value different from zero is interpreted as an attack as well. The fitness function is the operator measuring how well each individual in the population performs. We use the

Table 3.4: GP parameters used in the experiments.

Name	Value
Number of generations	300
Maximal tree depth	[2-10]
Population size	1000
Tournament size	8
Crossover rate	90%
Mutation rate	10%

fitness shown in Equation (3.1), which considers both the classification error, defined as the number of incorrectly classified events divided by the total events, and the C_{ID}. The greater the C_{ID}, the better. Because the algorithm we use aims at minimizing the fitness, we use $(1 - C_{ID})$ in the calculation. The values α and β are the weight given for each metric, and they must add up to one. In our experiments, we use 0.5 for both values (i.e., the C_{ID} and the classification error have equal weight).

$$fitness = \alpha \cdot E\_class + \beta \cdot (1 - C_{ID}) \tag{3.1}$$

A critical parameter here is the maximum tree depth. Since one goal of our reverse engineering attack is to generate an understandable model of the classification algorithm, it is important to avoid the bloating of the individuals when crossing and mutating the trees. The value of the maximum tree depth avoids bloating. On the one hand, setting a bigger value causes the GP algorithm to evolve more complex programs, which are likely to be difficult to understand. On the other hand, simpler models generally obtain worse results efficacy-wise, but they facilitate analysis for the evasion attack, which is the actual goal of the adversary. In Section 3.5.1, we provide an analysis of the impact of this parameter in the efficacy and effectiveness of the attacks.

Figure 3.3 shows an example of a model obtained by following this procedure. This program applies the operators to the features in the leafs following a tree-based scheme. In the example, we can observe that the model outputs 1 (anomaly) if both the left side (OR operator) and the right side (ADD operator) are true (i.e., non-zero values). The goal is that the programs evolve in such a way that they classify each instance properly, in a supervised setting using the second part of the dataset as training instances. Finally, programs are tested using the last part of the dataset to

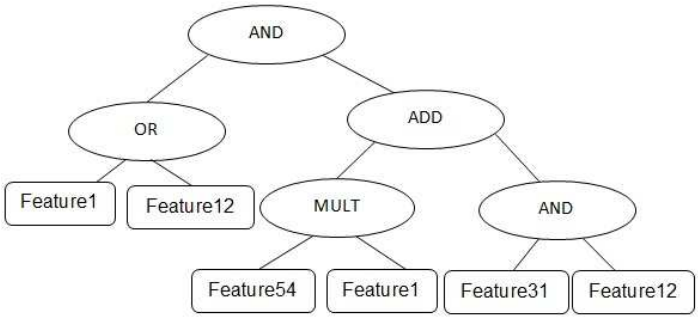


Figure 3.3: A sample of a GP model.

Table 3.5: List of operators used by GP.

Name	Description
ADD	Addition of two numbers
MULT	Multiplication of two numbers
DIV	Division between two numbers. Returns 0 if the denominator is 0
AND	Boolean 'and' operator between two numbers
OR	Boolean 'or' operator between 2 numbers
NOT	Boolean 'not' operator between 2 numbers
GREATER	Returns 1 if the first argument is greater than the second, and 0 otherwise
LEAST	Returns 1 if the first argument is lower than the second, and 0 otherwise
MAX	Returns the maximum of two values
MIN	Returns the minimum of two values

measure the effectiveness of the evolved models in terms of classification.

The training and testing phases are repeated several times in order to obtain a statistically significant measure that does not depend on the initial random seed. Using different random seeds covers a larger portion of the search space. We then take the results obtained with the best individual, i.e., the one that has produced the best test results, to perform the evasion attack.

Finally, it can be observed that in this setting the reverse engineering attack does not require the adversary to interact with the IDS, because it only uses a dataset with a similar distribution to the one used during the IDS training.

3.4.2 Evasion Attack

The reverse engineering attack explained above provides the adversary with a model of the way the IDS works that facilitates the construction of evasion attacks. Recall that the main idea of an evasion is to transform a instance that would be classified as a true positive by the IDS into one that would result in a false negative, i.e., performing attacks without generating alarms. The bottom-left part of Figure 3.2 shows a graphical explanation of this evasion attack.

Key information about how to find such evasions can be extracted from the evolved models using a search algorithm. In our experiments, the models are GP programs that have a tree shape, with a root node, several internal nodes distributed in different levels, and a final level with the leaves. The root and each internal node in a model is an operator and the leaves represent features extracted from the input data. These are the features that an IDS trained with a similar data distribution may use in the detection. Therefore, the evasion search is done by analyzing these features and operators from the models to look for potential vectors that evade the classifier. Then, these vectors are mapped into real payloads to evade the system using the inverse of the feature construction (FC^{-1}).

We perform the evasion search by using a special top-down tree-traversal searching algorithm over each candidate model. The final goal is to make the root node change the output (from 1, meaning attack, to 0, meaning normal). To this end, we allow the search to modify the value of its children nodes as follows. On the one hand, if any of the children is a leaf node, the corresponding feature is set to zero. On the other hand, if both children are intermediate nodes (operators), the algorithm requests them being zero, and the process is recursively repeated. In Section 3.5.2 we provide an example of how this search is done over a model.

The search over the tree models provides the adversary with a set of evasion strategies, which indicate which features should be modified in order to evade the classifier. The adversary generates a new dataset by implementing the modifications of the features suggested by the search. Accordingly, the adversary obtains a set of modified vectors from the input data that cause evasions. This modified dataset is given as input to the studied classifier to check whether new false negative appears. Each modified attack that is not detected by the IDS is considered as an evasion candidate. We use the term “candidate” because, as mentioned above, before

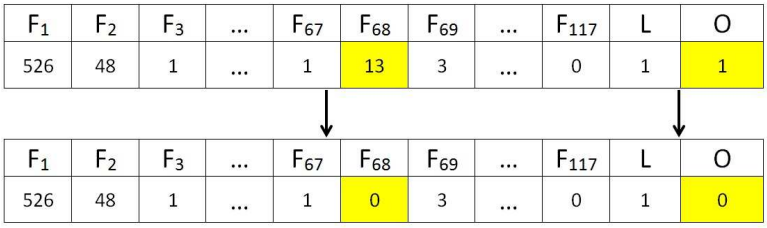


Figure 3.4: A candidate evasion strategy which modifies the feature F_{68} to evade detection.

considering it as a real world evasion we have to check that two requirements are met:

1. The payload obtained after the modification must represent valid HTTP payload. For example, the word GET cannot be removed from a HTTP request.

2. The attack still works after the modification. For example, removing the word INSERT in an SQL Injection translates into a useless payload for the adversary.

Figure 3.4 shows an example of a candidate evasion strategy. The row shows the 117 features constructed from the payload along with its label (L), indicating whether is a normal (i.e., $L = 0$) or an intrusive (i.e., $L = 1$) instance, and the output given by the IDS (O). The figure shows an attack trace where, when modifying the feature 68 from 13 to 0, the output (O) of the IDS changes from 1 (intrusion) to 0 (normal).

For each modified vector that potentially evades the IDS, it is required to get the corresponding payload to perform real world evasions. This is done by inverting the feature construction algorithm (FC^{-1}). In the concrete example of 1-grams, each feature in the vector specifies the number of 1-grams in the payload (see Table 3.1). Accordingly, the adversary only has to remove or insert 1-grams (bytes) in the payload to get the desired numbers as indicated by the features of the candidate vectors.

3.5 Results

This section presents the results obtained when attacking the four IDSs created using C4.5, CART, SVM and MLP. First, in Section 3.5.1 we present the models obtained

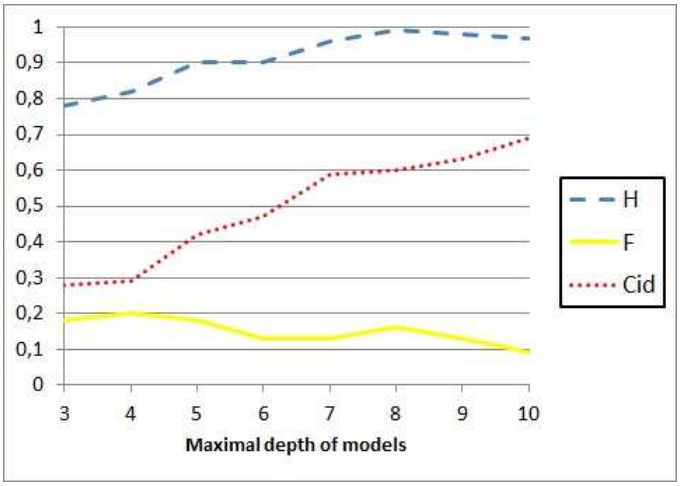


Figure 3.5: Efficacy vs Complexity of models.

through the reverse engineering procedure described above. Since the models are built using a dataset with a similar traffic distribution than the training dataset used by the classifiers, the results of the reverse engineering attack are generic for the four IDSs. Then, in Section 3.5.2 we explain how these models are used to conduct an evasion search and discuss one example of a malicious payload which is properly modified to evade the four IDSs studied.

3.5.1 Reverse Engineering

As explained in Section 3.4.1, we have experimented with different values of the parameter "maximum tree depth". Figure 3.5 shows the effectiveness in terms of H, F and C_{ID} of the best individuals obtained for each value of the parameter. Recall that bigger H and C_{ID} values and lower F values mean a better effectiveness of the classifier. It can be observed that, as the maximum allowed depth increases, so does the accuracy of the obtained models. However, bigger models are more complex and require bigger efforts to perform the evasion attack. Though the false positive rate of the models is rather high (above 10% in all the cases), we do not care about improving this because the final goal is to perform evasion. Accordingly, it is preferable that the models detect the attacks properly (i.e. getting a high detection rate), because this provides further information to perform the evasion search.

Figure 3.6 shows the best individual obtained with a maximum depth of 4. In the next section, we use this individual to illustrate how the search of evasion attacks is

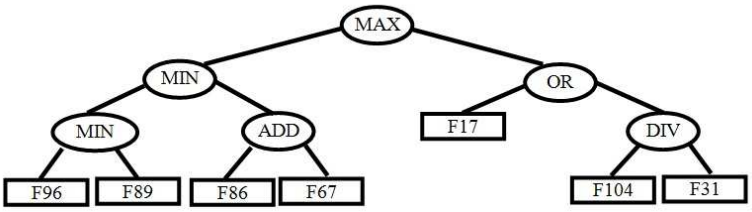


Figure 3.6: GP model obtained with the reverse engineering attack.

done. In this individual, the relevant features appearing in the leaves of the tree are:

- F96: Number of 'l'
- F89: Number of 'd'
- F86: Number of 'A'
- F67: Number of '-'
- F17: Number of special characters in the arguments
- F104: Number of 'U'
- F31: Number of '1'[1]

It can be observed that, with the exception of the feature F17, all the features are extracted with 1-grams. Modifying such features to get real world evasions is straightforward. Thus, the model presented provides the adversary with high chances to look for evasions. In the next section, we explain how this is carried out.

3.5.2 Evasion

In order to explain how the tree-traversal algorithm works, next we describe the search over the model shown in Figure 3.6. In order to succeed in evading the IDS, the output of the tree must be zero. The root is a MAX operator, which requires that its two children are zero. In what follows we describe how the algorithm works in each subtree to accomplish this goal:

[1]Feature 96 is the lower-case of L while Feature 31 is the character of the number one

Table 3.6: Example of a rule suggested after performing the evasion attack

[F89=0 OR F96=0 OR (F86=0 AND F67=0)] AND (F17=0 OR F104=0 OR F31=0)

1. The left child is a MIN operator, and requires one of its two children being zero.

   (a) The left child is a MIN, and it is zero if either the feature F89 (number of 'd') or the feature F96 (number of 'l') are zero.

   (b) The right child is an ADD, and requires setting to zero both the feature F86 (number of 'A') and the feature F67 (number of '-').

2. The second child of the root is an OR operator. It requires that either the feature F17 (number of special characters in the arguments) is zero or the right child is zero.

   (a) The right child is a DIV operator. It is zero if either the feature F104 (number of 'U') or the feature F31 (number of '1') is zero (in order to avoid inconsistent operations, if the divisor is zero, the operator DIV returns zero).

This search suggests various possible modifications of features according to the analysis above. The modifications are given in the form of rules. Table 3.6 shows an example of a rule obtained from analyzing the model shown in Figure 3.6. This rule shows that, for example, if the features F89 and F17 are set to zero, it is possible evade the tree model. However, performing such modifications to get real world evasions is not an easy task for an adversary. Indeed, F17 is the "Number of special characters in the argument's values", and in many attacks it cannot be set to zero because some special characters are required (like, for example, the '>' symbol for an XSS attack). Thus, this modification may not be useful in real attacks. In general, when evaluating the evasion attack the adversary should focus on modifications that are semantically valid, as explained in Section 3.4.2. As we show below, focusing on the features extracted by 1- grams simplifies the search of real world evasions, because inverting the FC is straightforward for the adversary.

We have analyzed the models generated with different tree depths and obtained all the output rules. Without loss of generality, we provide an example of how an

```
mode=register&login=mya&password=rencor&name=Juan&surname=Perez&email=jperez@fig
hting-machines.ukAND 1=1&id=05736398Z&adress=Victoria Kents 46&town=San Miguel
de Serrezuela&cp=18330&province=Girona&ntc=2610269003230246&B1=Register');
```

```
mode=regIster&logIn=mya&password=rencor&name=Juan&surname=Perez&emaIl=jperez@fIg
htIng-machInes.ukAND 1=1&Id=05736398Z&adress=VIctorIa Kents 46&town=San MIguel
de Serrezuela&cp=18330&provInce=GIrona&ntc=2610269003230246&B1=RegIster');
```

Figure 3.7: Original (above) and modified (below) malicious payloads, classified as intrusion and normal respectively by the four IDSs studied.

evasion could succeed by following this procedure. One of the modifications suggested in these rules is to set the number of 'i' characters to zero. Accordingly, we replace the character 'i' by its upper-case representation ('I') in all the attack instances initially detected by each classifier, thus generating a payload free of 'i' characters. Figure 3.7 shows an example of such a payload, which originally was detected by all the IDSs studied, i.e., using the C4.5, CART, SVM and MLP classification algorithms. Then, when such a modification is carried out, it evades detection of the IDSs. This payload is a typical example of an SQL Injection attack against the web server. It can be observed that it is a real world evasion because the HTTP protocol is not case-sensitive, so replacing a capital letter by its corresponding lower-case letter maintains the functionality of the protocol and thus the attack still succeeds. However, from the ML perspective, this small change in a feature suffices to alter the classification output. A prior normalization of the traffic exposed to the IDS, though, would counteract this evasion.

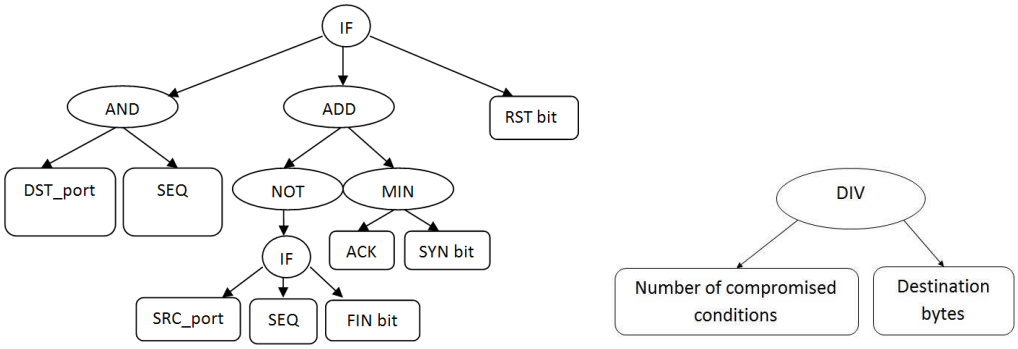
Another evasion strategy suggested by the rules consists of removing the hyphens ('-') characters from the arguments in the URLs. This could be done by changing these characters by the underscore('_') in the names or surnames of people. For example, in Figure 3.7, the argument "email" has the value "*jperez@fighting-machines.log*". If the domain of this email is changed to "*fighting_machines.log*", the evasion succeeds. However, the HTTP request has a different semantic, i.e., the domain of the email may not exist, and the response to this request may lead to some error message, like "invalid email". Nonetheless, the evasion attack is harder to counteract by the IDSs, as it is not enough to normalize the traffic, but also it would be required to remove invalid email domains (which in turn requires to manage a whitelist of these domains).

The aforementioned evasion attack may seem simple, as we are only changing a lower-case letter by its corresponding upper-case character, or hyphen by underscore characters. It can be observed that an adversary can easily compose a malicious payload that evades the IDSs. Somehow during training, the classifiers learn that the presence or absence of these characters can be used to tell apart normal from anomalous payloads. As shown in Table 3.3, the detectors obtain a rather good performance in terms of effectiveness. This happens because the ML algorithms are capable of processing a training dataset and, when a similar testing data is presented, they classify these data properly. However, ML algorithms do not have the domain-specific intelligence required to know whether the classification makes sense from the application at hand –intrusion detection in this case. Accordingly, as we have demonstrated, they are weak and vulnerable to specific targeted modifications. Once the adversary discovers this vulnerability through the reverse engineering attack, she just have to take care of setting properly the number of characters (1-grams) in the attack payload and thus the IDS will be evaded.

3.5.3 Additional Experimentation

The attacks discussed above were also tested against network-based detectors [Pastrana et al., 2010, 2011]. In this case, we used the LBNL dataset [Nechaev et al., 2010], which provides both normal traffic and traffic corresponding to a port-scanning attack, and the well-known KDD Dataset, derived from raw traffic captured during MIT/LL 1998 evaluation [Lippmann et al., 2000]. Different from the work presented in this chapter, in our previous works we only experimented with the C4.5 algorithm. At a first sight, the results demonstrated that an adversary could evade a C4.5-based NIDS with SYN flooding attack and port scanning attacks. However, we found that obtaining real world evasions in those cases were highly difficult for the adversary, because inverting the FC used in that works is not easy.

The GP models obtained in [Pastrana et al., 2010, 2011] are showed in Figure 3.8. As it can be observed, these models are rather simple and easy to understand, which was one of the objectives proposed for that works. This implies that the reverse engineering attacks succeeded. Regarding the evasion attack, though, the suggested evasion strategy from the models forces the attacker to perform hard modifications. Concretely, the strategy requires to modify either a TCP header field (the window



LBNL dataset KDD dataset

Figure 3.8: GP models obtained in previous experimentation with network traffic.

size) in the case of the LBNL dataset (left figure), or the "destination bytes" field of the KDD (right figure), which is the number of data bytes from destination to source. Consequently, obtaining real world evasions in this case is not easy, or even impossible in the case of the destination bytes, since it depends on the destination (victim) and thus the source (attacker) has not control over the field.

3.6 Discussion

We identify three main points that lead to the success of the attacks presented in this chapter. We next analyze them and discuss possible countermeasures.

1. *IDSs are trained with a dataset that does not represent properly the complete space of HTTP payloads.* This is one of the major problems in the application of machine learning to intrusion detection [Sommer and Paxson, 2010], and the situation is even worse in the case of payload-based detection. A solution would be to constantly retrain the detector, in the hope that feeding the classifier with newly observed payloads will result in more representativeness. However, these solutions entails new problems. For example, it must be ensured that the new payloads are attack-free, as otherwise the detector would incorporate malicious behaviors into its notion of normality. Moreover, the re-training solution must deal with poisoning attacks, in which the adversary carefully inserts data in order to progressively move the detection surface to a desired point. This attack is further explained in a paper by Biggio et al. [Biggio et al., 2012] with Support Vector Machines.

2. *The feature construction method is simple and it is easy to invert the process to get real world evasions.* The use of 1-grams to extract features is useful for efficiency purposes, but it allows an adversary to easily find real world evasions. Ideally, in adversarial environments, the feature construction should be a one-way function, i.e., whose invert function was computationally hard to perform. Thus, regarding the feature construction by 1-grams, it is required to use higher-order n-grams. For example, using values of n greater than 2, the Anagram IDS solves the problems identified in PAYL. In the next chapter, we provide further details about this.

3. *The adversary knows the distribution of the training dataset and the feature construction algorithm.* A naïve solution to hide this information to the adversary would be not publish the dataset used for training by the IDS. However, a query-response analysis would allow to obtain such information, as proposed by Ateniese et al. [Ateniese et al., 2013]. In general, securing a system by hiding its functionality is against the Kerckhoff principle which states that "*the security of the system must depend solely on the key material, and not on the design of the system*". Another solution, adopted by others IDS like KIDS [Mrdovic and Drazenovic, 2010] or Anagram [Wang et al., 2006] is to use secret information in the detection function, much in the way encryption functions use secret keys. However, the secrecy of KIDS has been broken in [Estévez-Tapiador et al., 2013]. Regarding the solution proposed by Anagram, in the next chapter we show that the secrecy of the detection is broken using a sophisticated reverse engineering attack. Indeed, as we discuss in the next chapter, randomization of IDS does not necessarily increment the security of the detection. However, this is still an active research area with more open problems than solutions.

3.7 Conclusions

In this chapter we have presented reverse engineering and evasion attacks against payload-based intrusion detection systems based on conventional machine learning classifiers. The attack is conducted in two phases. First, a reverse engineering on the IDS aims to discover which properties have the detectors learned from the training

dataset. Second, by analyzing these properties, the payload is carefully modified to get an evasion attack. Furthermore, when such IDS rely on weak feature construction algorithms, the evasion strategies suggested from analyzing the reverse engineered models are easy to translate into usable real world payloads. The reverse engineering process suggested here relies on Genetic Programming to derive tree-based models. The analysis of these models allows the adversary to find evasions over the IDS, by carefully modifying the payloads. Our experiments show that an adversary can find evasions in well-known classification algorithms such as C4.5, CART, SVM and MLP.

We have shown how an adversary can evade the detection if the training process makes the IDS to learn models that are specific to the training dataset used. In such a case, an adversary can infer such models (or approximations to them) and use them to evade the system. The premises for the attack are realistic, as both the traffic distribution of the protected network and the feature construction method are generally either public or easy to infer. In particular, the weaknesses of 1-grams are related to the fact that they are easily manipulable by an adversary, which in turn makes it easy to get real world evasions. This is a fundamental difference with other works focused on IDS analyzing network level traffic. For these, obtaining such real evasions requires additional capabilities for the adversary, such as for example being able to modify connection parameters.

In our approach, the adversary do not require to know the training algorithm, nor has to query the detector to build models that behave somehow similar to the target IDS. This avoids dealing with the near-optimal evasion problem stated in [Nelson et al., 2011], where authors discuss the complexity of a query-response analysis in terms of the number of queries required by an adversary.

4

Reverse Engineering Attacks on Randomized Classifiers: The Case of Anagram

4.1 Introduction

Reverse engineering attacks often seek to acquire knowledge that is essential to subsequently attain other attack goals. One clear example is evasion, as the attacker generally does not possess full details about the detection function and, therefore, potential ways of evading it. For example, in anomaly-based IDS the detection function is commonly built from a set of "normal" (and attack-free) events, such as network traffic or service requests, using machine learning algorithms [Hu and Shen, 2012; Pastrana et al., 2012; Satpute et al., 2013; Song et al., 2013]. The resulting model is then used to spot anomalous activities, assuming that anything deviating from normality is suspicious. A direct consequence of this operation principle is that any network packet that fits the normality model will not raise any alarm. An advanced attacker could try to modify his original attack payload so that it blends in with the normal behaviour of a network, thus evading detection. These strategies were termed Polymorphic Blending Attacks (PBA) by Fogla et al. in [Fogla et al., 2006], and were also demonstrated by Kolesnikov et al. [Kolesnikov and Lee, 2005] to evade PAYL [Wang and Stolfo, 2004], an anomaly detector based on n-grams (with $n = 1$ or $n = 2$).

The threat posed by evasion attacks such as those discussed in the previous chapter has forced some schemes to incorporate defenses to thwart them. Nearly all schemes proposed so far rely on the idea of depriving the adversary of some

critical knowledge about how the payload will be processed. This can be achieved in a number of ways. For example, McPAD [Perdisci et al., 2009] generates various different models of normality, each one based on a distinct set of features, and uses all (or some) of them to seek anomalies. In doing so, it forces the adversary to craft a payload that looks normal to all models. A similar idea was explored by Biggio et al. in [Biggio et al., 2010] by using multiple classifiers and randomly assigning weights to each of them in the final decision. Other detectors such as KIDS [Mrdovic and Drazenovic, 2010] draw some inspiration from cryptography and propose a scheme where the normality model depends upon some secret material (the key). In KIDS the key determines how the classification features are extracted from the payload. The security argument here is simple: even though the learning and detection algorithms are public, an adversary who is not in possession of the key will not know exactly how a request will be processed and, consequently, will not be able to design attacks that thwart detection.

While most such schemes successfully provide a rationale of their strength against evasion, almost none of them include in their security analysis arguments about an adversary who first reverse engineers the detection function and then uses the acquired knowledge to evade detection. Thus, for example, in PAYL an adversary can try to infer, either completely or approximately, which specific n-grams are not recognized as anomalous and then modify the attack (e.g., by adding carefully constructed padding as in [Kolesnikov and Lee, 2005]) so that it matches one of those n-grams. One debatable issue about reverse engineering attacks is precisely their viability and/or practical relevance. To begin with, it is assumed that the attacker can somehow observe the responses induced by inputs of his choosing, and also that he can query the IDS with a potentially large number of payloads. Shedding doubts about the feasibility of doing this is reasonable, but from a security perspective it would be unsafe to assume that it is not possible, even if it seems hard to figure out realistic scenarios where the attacker has such a capability at his disposal.

In this chapter we discuss reverse engineering attacks against randomized classifiers. We focus on Anagram [Wang et al., 2006], a well-known anomaly detector that models n-grams (with $n > 1$) observed in normal and attack traffic. Anagram, which can be seen as an evolution of PAYL [Wang and Stolfo, 2004] to resist evasion by polymorphic mimicry attacks, also introduces a new strategy to hinder evasion called randomization. Roughly speaking, each detector uses a secret and random mask

(the key) to partition packets into several (and possibly interleaved) chunks. These chunks are reordered according to the secret random mask to produce new inputs to the same normality model; the maximum anomaly score among them is assigned to the packet. Thus, in randomized Anagram the secrecy of the mask prevents an attacker from knowing where and how to modify the original attack so as each chunk will look normal.

We analyze the strength of randomized strategies such as that incorporated into Anagram against key-recovery attacks and the security consequences of an adversary being able to recover the secret mask. In particular:

- We discuss adversarial settings where an attacker is given the opportunity to interact with the detector by providing carefully chosen payloads and analyzing the binary response (normal/anomalous).

- We provide an efficient algorithm to recover the secret mask using a bounded amount of queries to Anagram and discuss the experimental results obtained with a prototype implementation.

- We show that knowledge of such a secret mask, even if it is just approximate, could actually make the randomized version of Anagram weaker than the non-randomized one against evasion attacks. We present an example of how an adversary may use it to carefully distribute the attack code along the payload to bypass detection.

The rest of this chapter is organized as follows. Section 4.2 describes the evasion problem present in PAYL and the design of Anagram, including the randomized variants. Subsequently in Section 4.3 we introduce an algorithm to recover the key used in randomized Anagram together with the associated adversarial model. In Section 4.4 we discuss the experimental results obtained with a prototype implementation, including an example of cross-site scripting (XSS) attack appropriately modified to evade detection. Finally, in Section 4.5 we summarize the main contributions of this chapter and draw conclusions.

4.2 The Anagram Detector

Anagram is a network IDS (NIDS) proposed by Wang et al. in 2006 [Wang et al., 2006]. It improves PAYL, a former version presented two years before by the same authors [Wang and Stolfo, 2004]. In 2005, Kolesnikov et al. [Kolesnikov and Lee, 2005] showed a method to evade PAYL using Polymorphic Blending Attacks [Fogla et al., 2006]. This section analyzes this evasion method and presents the mechanisms present in Anagram to counteract it.

4.2.1 Evasion Attacks on PAYL

In 2004, Wang et al. presented an anomaly-based NIDS called PAYL (Payload Anomaly Detection) [Wang and Stolfo, 2004]. For each payload length observed in the training data, PAYL obtains a normality model by storing every n-gram (with $n = 1$ or $n = 2$) from all available attack-free payloads. In detection mode, PAYL first extracts the n-grams of the packet being analyzed. Each n-gram that is not present in the normal model increments the anomaly score of the packet. If the final anomaly score exceeds a predefined threshold, then the packet is tagged as anomalous. Kolesnikov et al. described an efficient method to evade PAYL using Polymorphic Blending Attacks (PBAs) [Kolesnikov and Lee, 2005]. See Chapter 2 for a detailed description of PBAs.

In order to generate a PBA against a NIDS, the attacker first learns the normality model used by the NIDS, which in the case of PAYL consists of guessing the distribution of 1-grams (bytes) that normal payloads follow. The attacker then encrypts the attack body using a simple reversible substitution algorithm, where each unaccepted byte in the attack body (i.e., one that is not included in the normal model) is substituted with an accepted byte according to a particular substitution table. The objective of such a substitution is to masquerade the attack body as normal behavior, guaranteeing that it statistically fits the normal profile. However, as the attack body is sent within the polymorphic decryptor, an optimal substitution table should itself satisfy the normal model as well. Finding such an optimal substitution table turns out to be an NP-complete problem. In [Fogla and Lee, 2006], Fogla and Lee reduced the problem to a Satisfiability or an ILP (Integer Linear Programming) problem, which can be efficiently solved for the problem sizes involved. Finally, the

polymorphic decryptor is generated. As the anomaly score of PAYL is the percentage of previously unseen n-grams, the PBA adds some extra normal bytes as padding to make the score lower. The attack is spread over different packets, according to the Maximum Transmission Unit (MTU) of the system, in order to make it easier for the attack to pass undetected.

4.2.2 Anagram

In 2006, Wang et al. proposed Anagram [Wang et al., 2006] to overcome PAYL's vulnerability to PBA. Anagram uses a more complex model, and also randomizes the detection process. We next provide a brief overview of its functioning.

4.2.2.1 Higher Order n-grams

Anagram builds a model of normal behavior by considering all the n-grams (for a given, fixed value of n) that appear in normal traffic payloads. Unlike PAYL, Anagram uses higher order n-grams (i.e, $n > 2$), so instead of recording single bytes or pairs of consecutive bytes, it records strings of size n. This obviously increments the complexity of the normal model and, therefore, requires more computational resources. Anagram uses Bloom Filters [Bloom, 1970] to reduce the memory needed to store the model and the time to process packets. Anagram also uses a model of bad content consisting of n-grams obtained from a set of Snort signatures and a pool of virus payloads. This procedure is called by the authors semi-supervised learning. In detection mode, each n-gram that does not appear in the normal profile increments the anomaly score by 1, except if such an n-gram is also present in the bad content model, in which case the anomaly score is incremented by 5. The final anomaly score of a packet is obtained by dividing the final count by the total number of n-grams processed. Note that the use of bad-content models makes it possible for the anomaly scores to be greater than 1. With this semi-supervised procedure, the already known attacks are taken into account, making Anagram more efficient. More details about the implementation and how Anagram works can be found in [Wang et al., 2006].

Evading PAYL requires to learn the normal profile used by the detector. This can be done stealthily by sniffing traffic destined to the victim's network from an

external system. To be successful, the PBA needs to add normal n-grams (with $n = 1$ or $n = 2$) in order to decrease the relative frequency of unseen n-grams in the payload [Wang et al., 2006]. However, by using higher order n-grams, Anagram makes it harder to effectively reduce this frequency. The attacker has to prepare and execute the complex task of spreading the entire attack among multiple packets so as to reduce the number of unseen n-grams in each packet. However, this does not completely prevents Anagram from being evaded. In fact, the problem of finding a PBA against Anagram can be reduced using the techniques presented by Fogla and Lee in [Fogla and Lee, 2006]. For this reason, Anagram does not only focus on higher order n-grams but also introduces the concept of randomized testing.

4.2.2.2 Randomized Anagram

A PBA always contains some number q of n-grams that cannot be encrypted, such as the attack vector or the polymorphic decryptor. Therefore, to achieve a statistically significant percentage of valid n-grams, the attacker must add an extra amount p of padding, with $p >> q$. In [Wang et al., 2006], a technique called *randomized testing* that aims at thwarting PBA against Anagram is described. By using such a randomization, the attacker will not know exactly how each packet will be processed and, therefore, where to put the padding to guarantee that evasion is successful.

Figure 4.1 graphically shows how randomized testing works, assuming that a random mask with 3 sets is used. In this case, incoming packets are partitioned into 3 chunks by applying a randomly generated mask. Such a mask consists of contiguous strings of 0s, 1s or 2s. Anagram establishes that each string must be at least 10 bytes long in order to keep the n-gram structure of the packets (see [Wang et al., 2006] for a detailed description of the random mask generation). The mask is applied to the payload of a packet to assign each block to one of the three possible sets. Each resulting set is considered by Anagram as an independent packet formed by the concatenation of individual blocks, and are tested separately, thus obtaining different anomaly scores. The higher of these scores is the one given as anomaly score of the original packet. If such an anomaly score exceeds a predefined threshold T, then the packet is tagged as “anomalous”; otherwise it is considered “normal”.

The random mask applied in the detection process is kept secret. Consequently, an attacker does not know how the different parts of a packet will be processed in

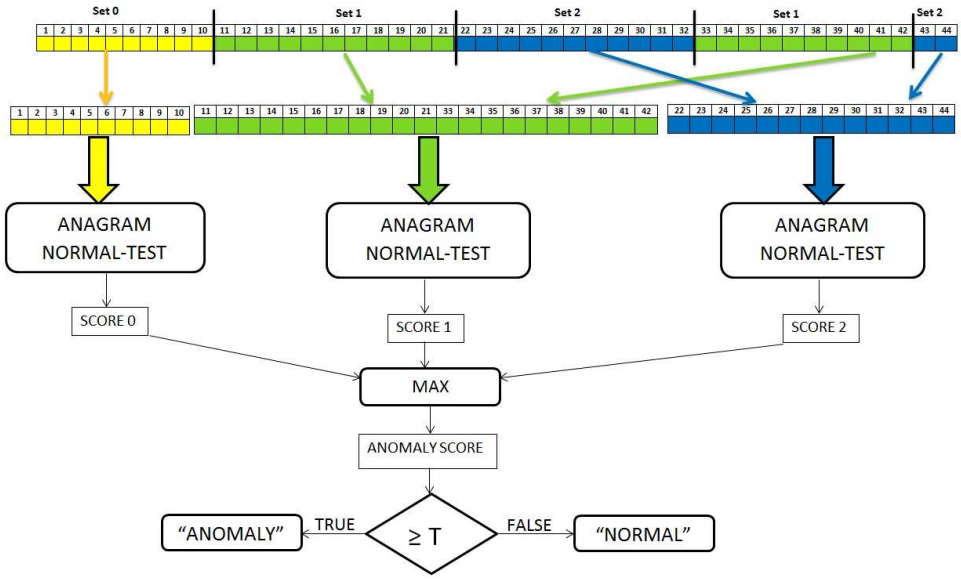


Figure 4.1: Computation of the anomaly score in randomized Anagram.

the detection process and, therefore, does not know where normal padding should be added in order to achieve an acceptable ratio of unseen n-grams. Thus, the first goal of an attacker pursuing to evade Anagram should be to find out the random mask used. Once this is achieved, the techniques presented in [Fogla and Lee, 2006] to perform an ordinary PBA could be applied. Moreover, if the adversary is able to estimate the random mask used in Anagram, this can be used to easily evade the system it other ways. We describe this in detail and present an example in Section 4.4.4. Next section describes our algorithm to reveal the secret random mask of Anagram along with the adversarial model that we consider.

4.3 Reverse Engineering Attacks on Randomized Anagram

In this section, we describe a reverse engineering attack against randomized Anagram. We first introduce the adversarial model required for this attack to work, including what capabilities the attacker must possess. We subsequently discuss the algorithm to recover the random mask. For simplicity, full details about the attack are provided in Appendix A.

4.3.1 Adversarial Model

In a reverse engineering attack, the attacker must possess the ability to interact with the system being attacked, often in ways that differ quite significantly from what may be regarded as normal (e.g., by providing malformed inputs or an unusually large number of them). In some cases, the ability to do so is close to the bare minimum required to learn something useful about the system's inner workings.

In the field of Secure Machine Learning, in particular when assessing the security of systems such as Anagram, one major problem comes from the absence of widely accepted adversarial models giving a precise and motivated description of the attacker's capabilities. Barreno et al. [Barreno et al., 2010, 2006] have recently introduced one such adversarial model and discussed various general attack categories. Our work does not fit well within this model because our main goal is not force the algorithm to misclassify an instance, but to recover one piece of secret information used during operation. Our reverse-engineering scenario is far more similar to that of Lowd and Meek [Lowd and Meek, 2005], where the focus is on the role of active experimentation with a classifier. In this case, as emphasized in [Lowd and Meek, 2005] it is absolutely essential for the attacker to be able to: (1) send queries to the classifier; and (2) get some feedback about properties of the query as processed by the system.

In this work, we use the adversarial model introduced in [Lowd and Meek, 2005] to analyze the security of Anagram against reverse engineering attacks. In particular, we assume that the adversary can query Anagram with specific inputs of his choosing and analyze the corresponding responses, i.e., the adversary can:

1. Prepare a payload p.

2. Query Anagram with p.

3. Obtain the classification of p as *normal* or *anomalous*.

Our emphasis in this work is on what can be attained by assuming an attacker with such capabilities. Consequently, we do not make any claims about the feasibility of the proposed attacks in real-world scenarios. However, in practical terms we identify two different settings where this adversarial model *might* materialize:

- *Offline setting.* In this case, the attacker is given full access to a trained but non-operational Anagram for a limited period of time. The attacker can freely query the system and observe the outputs at will and without raising suspicions. For example, this situation may occur during an outsourced system auditing, in which the consultant may ask the security administrator to take full control of the NIDS for a short period of time in order to carry out some stress testing. Among the battery of tests used, he might include those queries required by the attack.

- *Online setting.* Even if the NIDS is operational, it is reasonable to assume that an attacker can send queries to the NIDS, as the ability to feed the NIDS with inputs is available to everyone who can access the service being protected. Thus for example, such queries would be arbitrarily chosen payloads sent to an HTTP, FTP, SQL, etc. server. Two difficulties arise here. First, getting feedback from the NIDS (point 3 above) seems more problematic. In order for the attacker to determine whether an alarm has been generated or not, he would need to exploit an already compromised internal resource, such as for example an employee or device that provides him with this information. Alternatively, side channels may also be a source of valuable information (in particular, timing channels [Brumley and Boneh, 2005; Chen et al., 2010]), for example if it takes a different amount of time to classify a normal and an anomalous request, and this can be remotely determined. The second difficulty has to do with the fact that during the attack Anagram receives a large amount of queries, many of which will be tagged as anomalous. As this might certainly raise some suspicions, the attacker could spread them over a much larger period of time.

4.3.2 Mask Recovering Algorithm

We next describe an algorithm that recovers the secret mask applied by randomized Anagram. For reasons that will be clearer later, in some cases our attack could fail to locate exactly the borders between sets. Fortunately, such errors can only occur in the proximity of the borders and, therefore, the majority of the recovered mask is correct. Furthermore, the masks thus recovered are still extremely useful for an adversary to launch an evasion attack, as they point out which parts of the payload

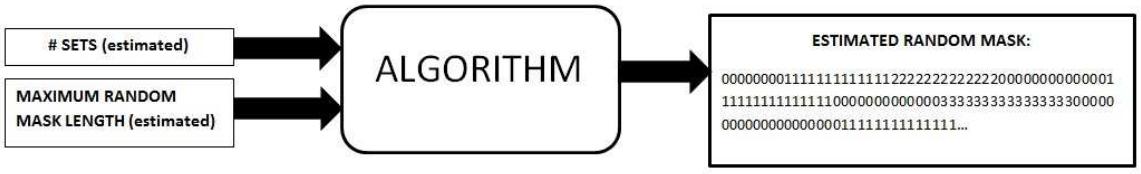


Figure 4.2: Input-Output view of the mask recovering attack.

will be grouped together for analysis, even if there is some uncertainty about a few bytes at the beginning and end of each block.

For readability, in this section we present a high-level description of the attack. The pseudo-code of all involved procedures along with a detailed explanation can be found in Appendix A.

As described above, Anagram's masks are formed by concatenating runs of length at least 10 of natural numbers from the set $[0, K]$. As shown in Figure 4.2, our attack requires two inputs: (1) the maximum estimated size of the mask; and (2) the maximum estimated number K of sets. The attack would be successful if both parameters are greater than or equal to the actual ones in the mask. However, these inputs have a direct influence on the execution time of the attack, in such a way that a more conservative (or resourceful) adversary could just use sufficiently high values to guarantee that the recovered mask is correct. Alternatively, it is possible to launch several attack instances, each one with a progressively higher value, until the result does not change.

The attack returns a vector with the estimated random mask, each position indicating the estimated set number. We will use the term *delimiter* to designate those mask positions where the mask changes from one set to another; in particular, if $m_i m_i \cdots m_i m_j m_j \cdots m_j$ are two consecutive blocks in the mask, with $m_i, m_j \in [0, K]$ and $m_i \neq m_j$, then the delimiter is the first m_j.

The algorithm is iterative. At each iteration, it identifies a new set $S_{current}$. The attack stops either when all the mask positions are filled with a set number, or when the maximum number of sets is reached. Each iteration of the algorithm is composed of two phases, which are explained in next sections. Each phase uses as input an initial starting point (I_s), which corresponds to the first position of the delimiter in $S_{current}$. In the first iteration, I_s is set to 1. In subsequent iterations, I_s is the first delimiter found in the previous iteration such that no set has yet been assigned to it.

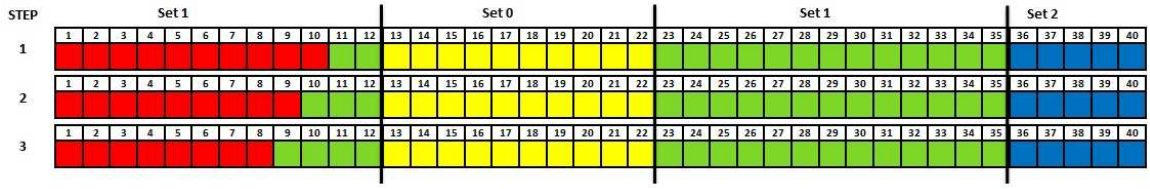


Figure 4.3: Phase I of the attack: obtaining an "nearly anomalous" payload.

4.3.2.1 Phase I

The main goal of this phase is to construct a payload that is "nearly anomalous". Such a payload is one that is classified as normal by Anagram, but such that if one single byte is replaced by an "anomalous" one μ (which causes changes in n consecutive n-grams), forces Anagram to classify it as anomalous[1]. The key observation here is that such a substitution will cause the payload to become anomalous if and only if μ is put in a position that belongs to the set with the maximum anomaly score (S_{max} for convenience), and that is not close to the border with the next set so that most affected n-grams fall in S_{max}.

The payload is built up as represented in Figure 4.3. The algorithm starts with a normal payload and I_s is set to 1. Then, μ is inserted into the 10 positions that follow I_s (red cells in Figure 4.3). Next, the algorithm checks whether the payload becomes anomalous (i.e. whether $S_{current}$, which is set 1 in Figure 4.3, is S_{max}). If so, the algorithm removes one by one the μ bytes starting at the end (steps 2 and 3 in Figure 4.3), until the payload becomes normal again (step 3 in the Figure 4.3). The resulting payload is such that, while it is classified as normal, a single addition of μ in any of the positions associated with S_{max} would cause it to become anomalous. Note that after Phase I, we guarantee that $S_{current} = S_{max}$.

4.3.2.2 Phase II

By using the payload obtained in Phase I, Phase II exploits the fact that the addition of a single μ into the sets of S_{max} would cause the entire payload to become anomalous, whereas if it is inserted in any other set the payload remains normal. Phase I obtains

[1]In our experiments, we have used as μ a byte with very low probability of occurrence in normal payloads, such as for example control characters (low ASCII numbers) or letters belonging to an alphabet different from the one used in the service being protected by the NIDS (e.g., ñ, accents, etc. if the system serves English web pages).

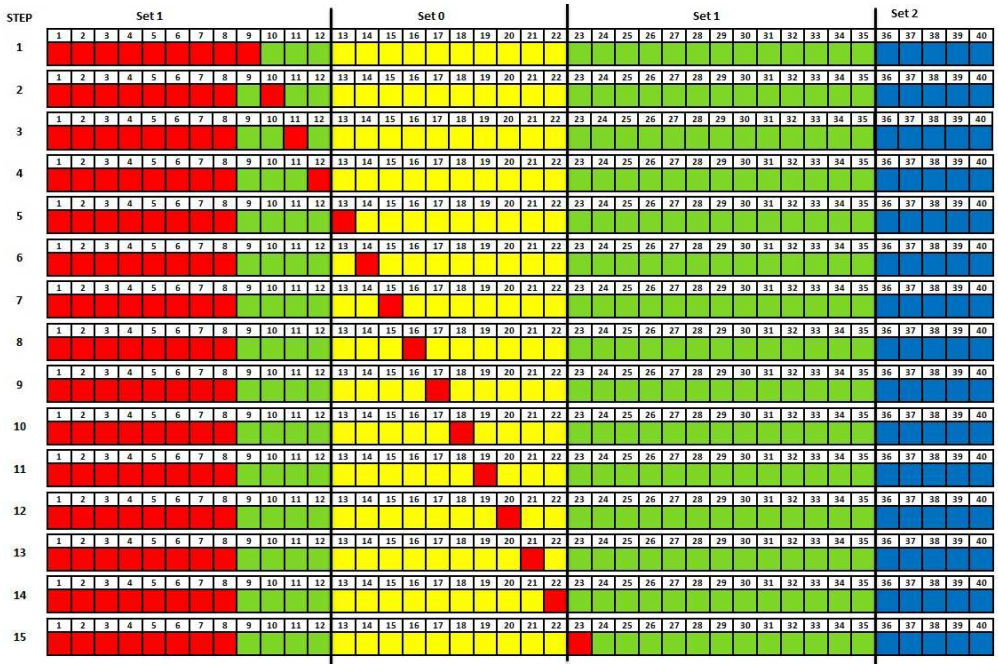


Figure 4.4: Phase II of the attack to discover the delimiters of the random mask. The anomalous character μ is moved throughout the payload to detect changes in Anagram's output. Each change corresponds to a delimiter of the set being processed. In the figure, the algorithm would detect the delimiters of Set 1, located at positions 13, 23 and 36.

payloads such that $S_{current}$ is S_{max}. Now Phase II moves one μ over the entire packet, as shown in Figure 4.4. Whenever such a moving μ is inserted within the limits of S_{max}, the payload becomes anomalous; otherwise it remains normal. This allows us to identify all those positions of set $S_{current}$ in the mask. In addition, any position where the output changes from normal to anomalous, or vice versa, is considered a delimiter between set $S_{current}$ and any other set. Thus, the next starting point I_s will be the first delimiter belonging to a set that has not yet been assigned.

There are some border cases where the procedure described above may fail. These are next discussed, together with a simple but effective countermeasure consisting of running Phase II multiple times and carrying out a majority voting step.

4.3.2.3 Majority Voting

In the description of Phase II above it was assumed that the set obtained in Phase I, $S_{current}$, is S_{max}. Thus, μ is moved throughout the entire payload to detect where the output changes, therefore detecting the delimiters of the current set. However, the

algorithm fails if the positions close to the delimiters contain an already anomalous n-gram. The problem is that the payloads obtained in Phase I are "nearly anomalous", meaning that one single μ within the limits of S_{max} usually induces a change in the output. The anomaly score, as explained in Section 4.2.2.2, is obtained by dividing the number of unseen n-grams by the total number of n-grams. Consequently, the output changes when the number of unseen n-grams increases. However, during Phase II, if μ is inserted within an already unseen n-gram, then the number of unseen n-grams remains constant, the output does not change and, therefore, the delimiter is not detected. This situation, which decreases the effectiveness of the algorithm, can also be exploited to evade Anagram, as we discuss in Section 4.4.4.

In order to increase the robustness of our attack, we introduce a majority voting scheme. Instead of simply recording the results for a single payload obtained in Phase I, we use several of them. The algorithm records all the positions indicated by each payload (votes) and, if some position has at least one half of the votes, then it is considered a delimiter. Even in those cases where it is unclear where the delimiter is, an analysis of the number of votes on each position will allow the adversary to estimate zones where the delimiters are supposed to be, which is enough to evade the system. We show this fact in Sections 4.4.2 and 4.4.4.

4.4 Experimental Setup and Results

We have implemented Anagram using the pseudo-code available in [Wang, 2007]. Both our attack and Anagram's implementation have been written in Java. Experiments have been run in a dual-core machine with 4GB of RAM. We have trained and tested Anagram using the same HTTP datasets used by McPAD [Perdisci et al., 2009], another application-layer anomaly detector, which are freely available[2]. A summary of the number of payloads and the partition into training, validation and test sets is given in Table 4.1. To generate the bad-content model, we use the web-based signatures of Snort [Roesch, 1999], as done originally in Anagram[3]. Furthermore, to avoid inserting normal n-grams into the bad-content model, we filter out the data using the validation set from the McPAD dataset and a list of known words of the

[2]See `http://roberto.perdisci.com/projects/mcpad`

[3]We do not use any virus signatures, as Anagram's authors do not provide information about what kind of virus database they used.

Table 4.1: Description of the dataset

	Training	Validation	Test	Total
Normal payloads	102157	1521	1050	104728
Attack payloads	–	–	1050	1050
Total	102157	1521	2100	105778

HTTP protocol.

We performed the experiments using 3 different n-gram sizes: $n = 5$, $n = 6$ and $n = 7$. Both in the original Anagram paper and in our experiments these values translate into the best detection quality. The experimental results presented next are grouped into three sections. In Section 4.4.1 we assess the performance of randomized Anagram as suggested in the original paper and discuss its limitations in terms of detection quality. Subsequently, in Sections 4.4.2 and 4.4.3 we report on the accuracy and efficiency (in terms of queries and CPU time) of the attack, respectively.

4.4.1 Detection Accuracy

In this first experiment, we assess the detection accuracy of randomized Anagram and compare it with the non-randomized version of the detector. Wang et al. [Wang et al., 2006] reported results on randomized Anagram using a binary mask (i.e., with just 2 sets). As our attack is designed to estimate random masks composed of any number of sets, we also explored the performance of randomized Anagram using a number of sets greater than 2, with different mask lengths. Specifically, we have experimented with 2, 3, 4, 5, 6 and 7 sets, and mask lengths of 128, 160, 200 and 256 bytes. In this section, we first present a ROC analysis of the randomized detectors. We next analyze the effect of introducing randomization on the anomaly score distribution, showing that the anomaly threshold in the case of randomized detection has to be increased in order to maintain a low false alarm rate.

4.4.1.1 ROC Analysis

Figure 4.5 shows the ROC curves of the detectors using n-grams of size 5, 6 and 7. Each plot contains a ROC curve for different number of sets, which has been obtained by averaging the results of different mask lengths. These curves are similar to those

originally presented by Wang et al. in [Wang et al., 2006]. It can be observed that for any value of n, as the number of sets increases, so it does the false alarm rate. However, all detectors achieve a 100% of detection rate with a false alarm rate lower than 1%. A false alarm rate greater than 1% has traditionally being considered unmanageable for a human operator at a large installation [Champion and Durst, 1999]. Accordingly, a typical design goal is to develop NIDS that can operate at points with a false alarm rate under 1%.

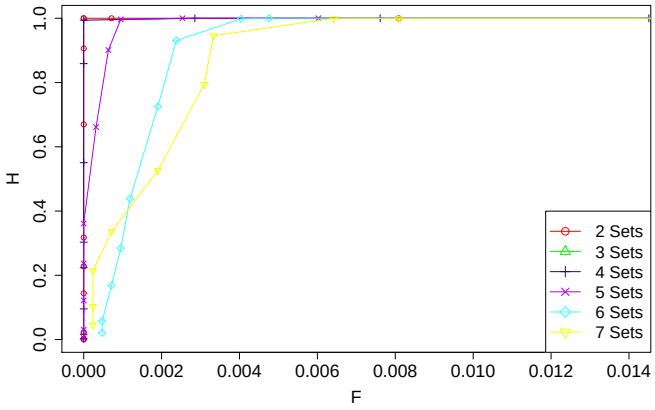
4.4.1.2 Anomaly Score Analysis

Figure 4.6 shows the distribution of anomaly scores obtained during test using various n-gram sizes for attack-free packets (FREE, in red) and packets containing Polymorphic Blending Attack (PBA, in blue). Each plot shows the anomaly score in the x-axis and the number of payloads having such an anomaly score in the y-axis. Dotted lines represent the results of the randomized detector using a mask of 128 bytes with 2 sets, while solid lines show the results using a normal, non-randomized version of Anagram.

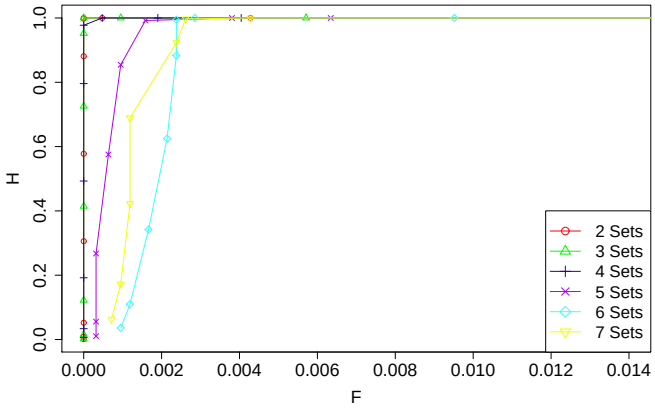
As it can be observed, the randomized detector considerably increases the anomaly scores for both FREE and PBA packets. In our experience, this increment is even greater for attack-free packets. Although both distributions remain reasonably distinguishable, the detection threshold in the case of randomized testing is significantly higher. As we explain in Section 4.4.4, this property makes Anagram less robust against an adversary who has discovered the random mask.

4.4.2 Effectiveness of the Attack

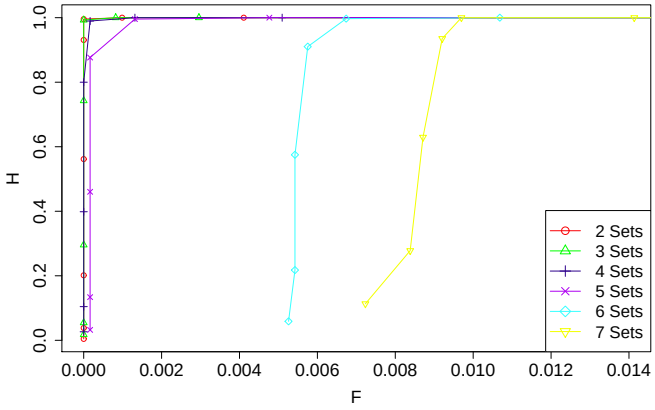
Figure 4.7 shows an example of the execution of the attack using 3 sets and masks of 160 bytes. The figure is partitioned into three vertical blocks that should be viewed as concatenated. The actual mask content is shown in the first row (with the tag M). We run the attack using a maximum number of 8 sets and a maximum random mask length of 300 bytes. Each set is represented with a different color. The state of the estimated random mask after each iteration is shown in a different line. Thus, the second row corresponds to the initial state, where all the positions are set to -1 (in red). In the first iteration of the attack, the algorithm estimates the positions of the



(a) N=5



(b) N=6



(b) N=7

Figure 4.5: ROC curves obtained by randomized Anagram for different number of sets and different n-gram sizes.

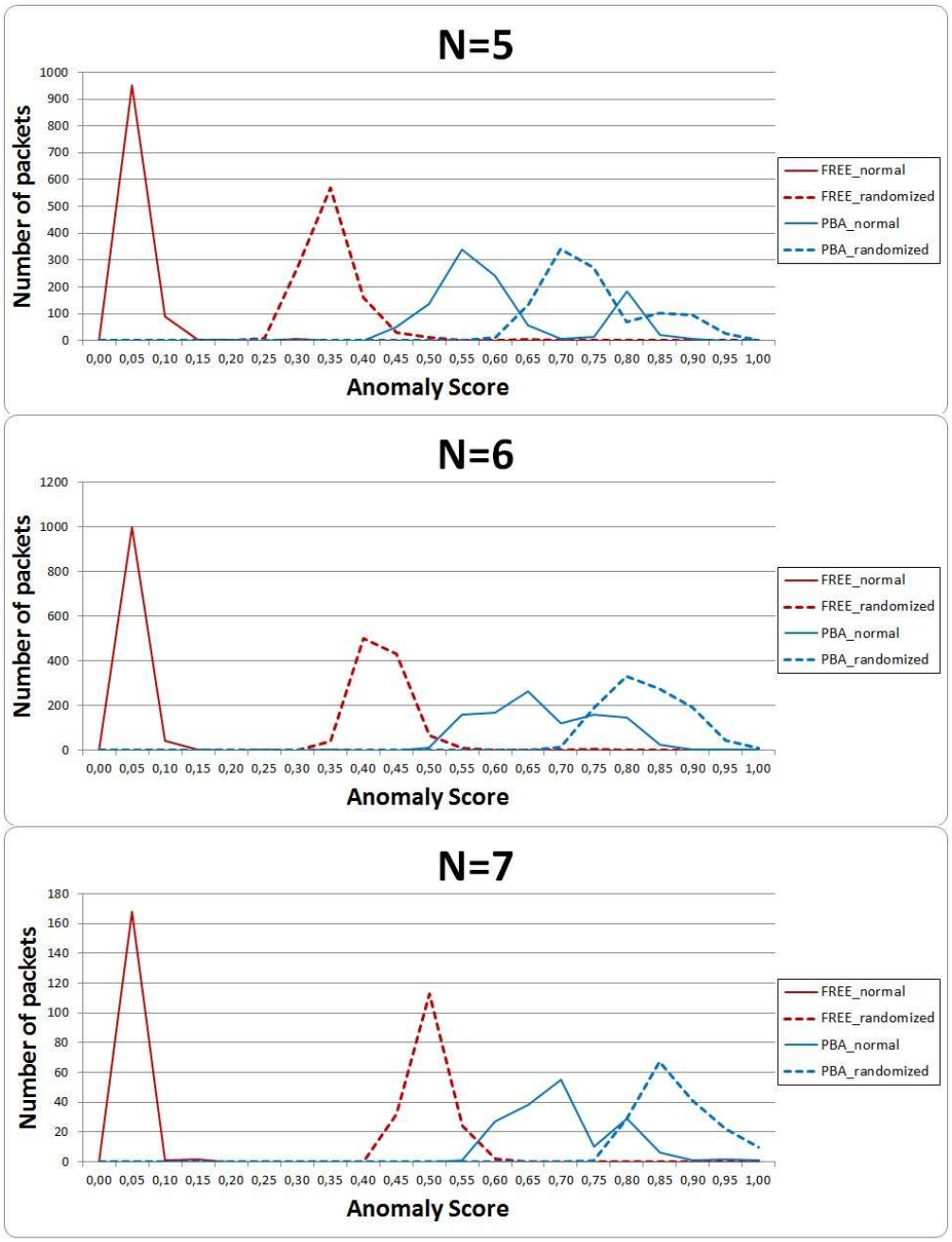


Figure 4.6: Distribution of the anomaly scores using 5-, 6- and 7-grams in non-randomized (plain lines) and randomized (dotted lines) Anagram.

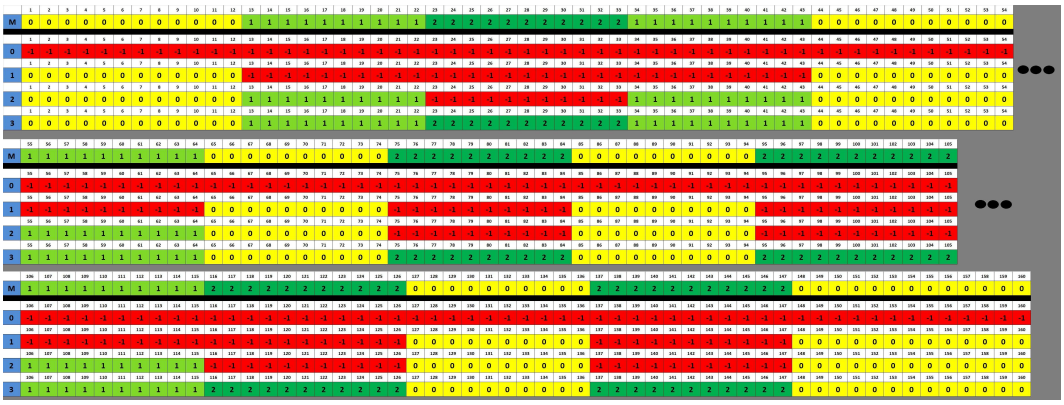


Figure 4.7: Evolution of the estimated random mask through consecutive iterations of the attack. The first line shows the actual mask with 3 different sets. The second line shows the initial state with all the positions set to -1, while each remaining line corresponds to an iteration of the algorithm.

first set (0, in yellow). In the second iteration, the positions of set 1 are estimated (light green in the figure), while in the third step the algorithm finds the positions of set 2 (dark green). After step 3, there is a positive value in every position of the estimated random mask, so the algorithm stops and returns the estimated mask which, in this case, perfectly matches the original.

In the case shown in Figure 4.7, the attack recovers exactly the random mask used. However, due to the situation explained in Section 4.3.2.3, in some cases the algorithm fails to correctly identify some sets. In order to evaluate such errors, we calculated the distance between the mask estimated by the attack and the actual one, measured as the number of incorrectly guessed sets divided by the total number of sets to normalize the results. We repeated each experiment 10 times with randomly generated masks for different Anagram configurations (i.e., varying the size of the n-grams, the mask size, and the number of sets in each mask) and computed the average distance between the recovered and the actual mask.

The number of packets to be used in the voting process depends on the desired accuracy. Figure 4.8 shows the error obtained when varying the number of voting packets for three different Anagram configurations. For the configurations used in this chapter, it was experimentally determined that no significant error reduction is achieved with more than 30-40 votes. For example, in the case of random masks of 128 bytes with 2 sets, only 15 packets would be enough to reveal the mask perfectly. Accordingly, for a mask of 200 bytes and 4 sets, using 20 packets would reveal it as well. Even in the case of a more complex configuration, such as the one with

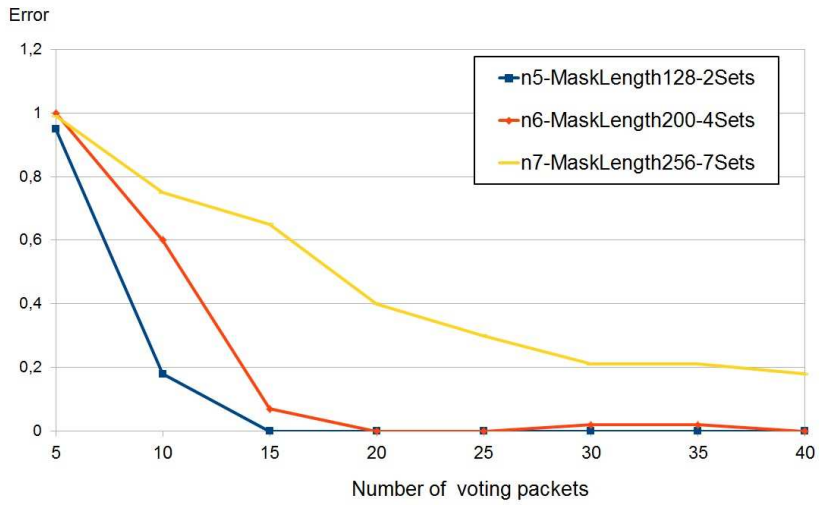


Figure 4.8: Decrease of the average distance between recovered and actual mask as the number of voting packets increase.

a random mask of 256 bytes and 7 sets, 40 packets correctly recover 82% of the mask (i.e. an error of 18%). As we discuss later, this percentage provides enough information to accomplish an evasion attack against Anagram.

In summary, the expected error depends both on the number of sets and the size of the random mask. Table 4.2 shows the experimental results for two different attack configurations using 10 and 40 votes, respectively. Several conclusions can be drawn:

- Binary masks are very easy to recover, no matter the size of the n-grams and, to an extent, the number of voting packets.
- The attack's probability of error increases with the number of sets and the mask length. The size of the n-grams seems to have no significant influence.
- Error decreases as the number of voting packets increases. For example, while in the case of 10 packets the average error falls between 0.08 and 0.68, it is reduced to less than 0.07 when 40 packets are used.

An interesting property of our mask recovery process is that, even if it does not estimate the mask exactly, an adversary can figure out approximately where the delimiters of the mask are by just looking at the votes, as shown in Figure 4.9. This

Table 4.2: Average distance between estimated and actual masks. Each row corresponds to a different number of sets (K), while each column determines the mask length and the size N of the n-grams.

	10 voting packets												
	N=5				N=6				N=7				Average
K	128	160	200	256	128	160	200	256	128	160	200	256	
2	0.18	0.02	0.02	0.02	0.05	0.02	0.00	0.02	0.26	0.02	0.20	0.13	0.08
3	0.42	0.49	0.54	0.58	0.40	0.48	0.53	0.55	0.54	0.56	0.47	0.50	0.51
4	0.39	0.51	0.62	0.70	0.37	0.57	0.60	0.65	0.42	0.56	0.57	0.70	0.56
5	0.50	0.59	0.64	0.71	0.53	0.70	0.63	0.69	0.48	0.60	0.62	0.70	0.62
6	0.53	0.64	0.63	0.75	0.56	0.68	0.64	0.71	0.58	0.65	0.69	0.65	0.64
7	0.61	0.60	0.75	0.75	0.55	0.65	0.77	0.76	0.65	0.64	0.71	0.75	0.68

	40 voting packets												
	N=5				N=6				N=7				Average
K	128	160	200	256	128	160	200	256	128	160	200	256	
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	0.00	0.00	0.00	0.03	0.01	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	0.00	0.06	0.06	0.12	0.05	0.00	0.00	0.04	0.00	0.00	0.11	0.00	0.04
6	0.00	0.00	0.00	0.02	0.08	0.00	0.03	0.16	0.16	0.00	0.04	0.42	0.08
7	0.03	0.00	0.00	0.05	0.11	0.03	0.03	0.11	0.16	0.09	0.12	0.18	0.07

plot represents the number of votes obtained for a random mask of length 128 and 3 sets using 7-grams. The x-axis shows the mask positions, while the y-axis shows the number of votes (i.e., packets indicating that there is a delimiter in this position). Take for example position 74, which is an actual delimiter. The number of votes after iterations 2 and 3 are not enough, as the final count does not reach half of the votes, so the majority voting scheme does not determine that there is a delimiter there. As a consequence, the algorithm fails and the remaining sets may or may not be properly estimated. This limitation can be detected either by a graphical analysis of the voting results or, alternatively, by adjusting the number of votes required to achieve majority. This would allow the adversary to determine which zones of the payload are very likely to contain no delimiters. Thus, even if the exact mask is not recovered, this information may suffice to perform an evasion attack, as we later illustrate in Section 4.4.4.

4.4.3 Efficiency of the Attack

In the previous section we have shown that the proposed attack succeeds in recovering the random mask or, at least, gives enough information about its structure. However, when trying to evade any security system, it is critical to do so spending as few

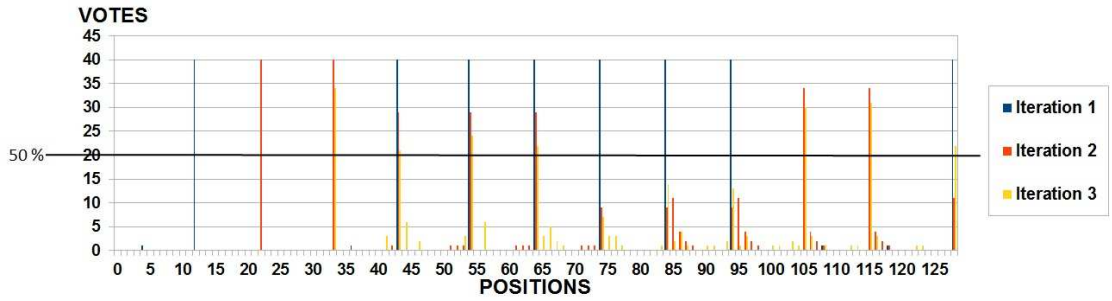


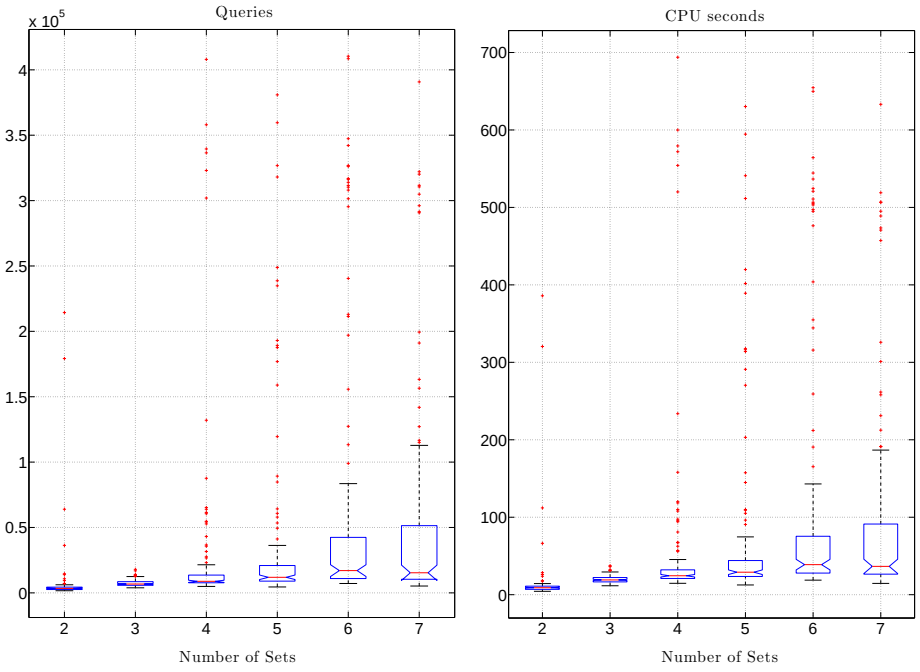
Figure 4.9: Votes obtained when estimating the mask of 3 sets and 128 bytes length

computational resources as possible. Figure 4.10 shows the number of queries to Anagram and the CPU time (in seconds) required by the algorithm for different number of sets. In general, the time required by the attack directly depends on the number of queries made to Anagram. In turn, the number of queries strongly depends on Phase I of the attack, where a "nearly-anomalous" payload is obtained, hence that the data range (size of each box) is relatively large. For example, using 40 voting packets, the attack requires between 40 seconds (for binary masks) and 120 seconds (for masks composed of 7 sets), with the average number of queries ranging between 100 and 6000.

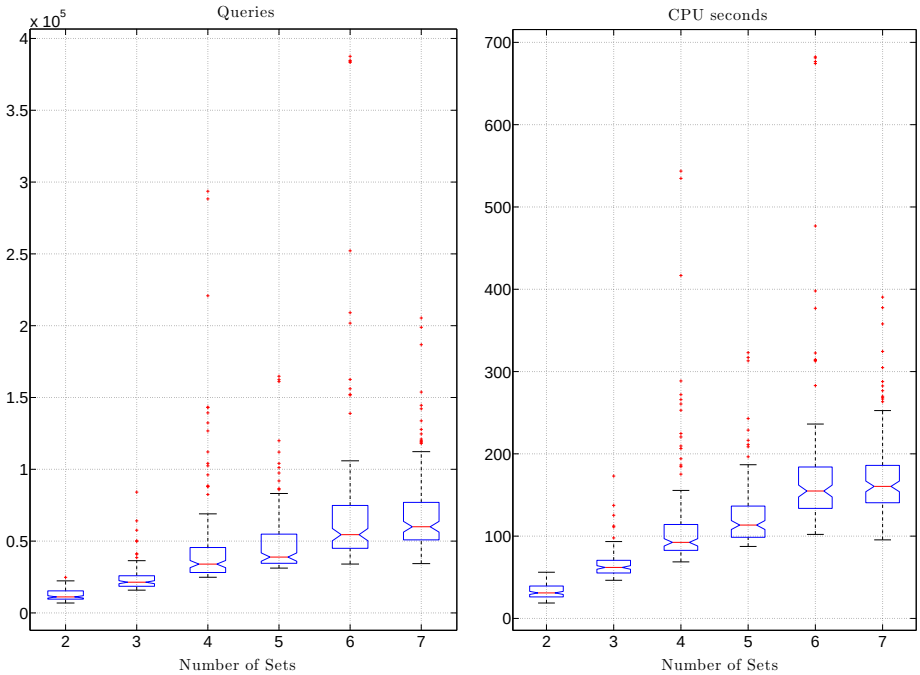
4.4.4 Exploiting Randomization to Evade Detection

Figure 4.6 shows that the anomaly score using randomized testing is typically larger than using normal testing, both for attack-free and PBA payloads. This happens because those payload bytes that are placed in positions around a mask delimiter are partitioned and concatenated with other chunks of data. In this process, several unseen n-grams may appear, even with bytes of normal data. Therefore, in order to achieve a good detection rate while minimizing the false positive rate, the anomaly threshold must be increased.

If an adversary is able to obtain the random mask being applied, then he can use the randomized testing process to evade the system. As mentioned above, when using randomization the threshold should be increased, thus tolerating the presence of more malicious bytes in the payload. Such malicious bytes are supposed to be in the positions of the mask delimiters, so an adversary can generate a packet where the malicious content is placed exactly in these positions, padding the remaining parts of the payload with normal bytes. Moreover, if the adversary suspects that



(a) 10 voting packets



(b) 40 voting packets

Figure 4.10: Queries and CPU time required to estimate the random masks with different number of sets.

parts of the malicious payload appear in the bad-content model of Anagram, then he can distribute this content around the delimiters too, in such a way that it will be split and will no longer match this bad-content model.

Figure 4.11 exemplifies this idea with a simple instance of an XSS attack using javascript. The attack actually just launches a pop-up window in the client side. The first payload only contains the code needed to perform the attack. In the second payload, padding is added to reduce the percentage of unseen n-grams. Assume that "image/" is a string that appears in the normal model. In such a case, padding is inserted as comments of the language (after the characters // and between /* and */).

The third payload in Figure 4.11 refers to the same attack, but is especially crafted to evade detection by using the estimated mask. We assume that the string "<script>" is in the bad-content model, as it is a frequent word in XSS attacks. Therefore, when preparing the payload, this word should be placed around some delimiter in order to have it split. Afterwards, the attacker places the desired amount of padding bytes in proper places to benefit from the randomization process. Figure 4.12 shows an example of such a payload prepared for a random mask of 2 sets. It can be seen that when using randomized testing, the word "<script>" will be divided into the 5-gram "<scr*/" and tested as part of the set 1, and the 5-gram "ipt>/", tested as part of the set 0. Table 4.3 shows the anomaly scores obtained for each case. For the original payload (first row), both anomaly scores (for normal and randomized testing) are very large, which means that the payload is undoubtedly considered as an anomaly. For the modified payload with normal padding (second row) the anomaly score is 0.45, exceeding the threshold established and therefore being considered anomalous again. However, when using randomized testing and preparing the rogue payload with padding inserted in the proper positions (third row), the anomaly score obtained is even lower than the one of the normal test. Moreover, as discussed in Section 4.4.1, the anomaly threshold is higher when using randomization than when using normal testing. In fact, in this case the anomaly threshold is 0.55, so the attack payload will be classified as normal, and thus evasion would succeed. This example illustrates that the randomized testing is a double-edged sword, since if the adversary is able to guess the mask, then he can use it against the detection system.

1) Original attack payload

```
<script>
alert('attack')
</script>
```

2) Attack payload with padding (normal test)

```
image/
<script>
/*image/*/ alert(\'attack;\');
//image/image/image/image/image/image/image/image/image/image/image/image/image/image/imag
</script>
```

3) Attack payload with prepared padding (randomized test)

```
image/
<script>
/*image/*/ alert(\'attack;\');
//image/imageimage/imag/image/imagee/image/imag/image/imae/image/image/image/image/image/
</script>
```

Figure 4.11: Examples of attack payloads used to evade Anagram.
Payload 1 is the original without padding. Payload 2 adds normal padding. Payload 3 is crafted to evade Anagram once the random mask has been estimated.

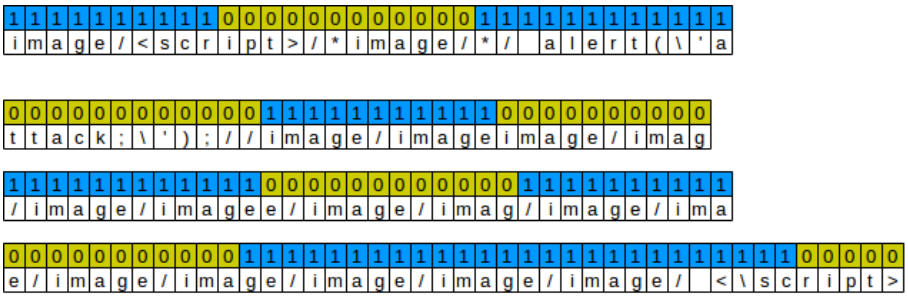


Figure 4.12: Example of how an adversary can set up an attack using the estimated random mask to evade Anagram.

Table 4.3: Anomaly scores obtained with the original payload, the payload with normal padding, and the payload prepared to evade the system using the estimated random mask.

	Normal testing	Random testing
1) Original payload	1.50	2.00
2) Payload with padding	0.45	0.53
3) Payload with prepared padding	0.60	0.38

4.5 Conclusions

In this chapter, we have analyzed the strength of randomized Anagram against mask recovery attacks. Even though the use of randomization certainly makes evasion harder, we have shown that an adversary who manages to find out such a mask could actually take advantage of the randomized detection process to evade Anagram, thus turning a security measure into an undesirable feature. We have proposed and evaluated a procedure to recover the secret mask by querying Anagram with carefully constructed payloads and observing the results. Our attack is quite efficient in terms of the number of queries employed, requiring no more than 2 minutes to recover the mask in the worst scenario for the range of the suggested parameters. As discussed above, we do not make any claims about the feasibility of the proposed attack in real-world scenarios, as this strongly depends on the adversary having the ability to interact with Anagram in the ways detailed in Section 4.3.1.

A possible countermeasure to the proposed attack is to randomize the choosing of random mask itself. Thus, each analyzed packet should be tested against a different random mask, possibly with different parameters too. While this would certainly stop our attacks from being effective, we have not assessed the potential impact of such a double randomization from the detection point of view.

While the attacks presented in this chapter are not directly applicable to other randomized anomaly detectors, the underlying ideas are general and can be used to reverse engineer other schemes based on similar constructions.

5

A Model for Intrusion Detection Networks and Adversarial Attacks

5.1 Introduction

In the previous chapters, we have discussed that hardening IDN nodes against evasion attacks is not a straightforward task. A possible countermeasure to obscure the way the detection is performed is the use of randomization mechanisms. However, these mechanism have not been proven robust. Specifically, in Chapter 4 we detailed how to successfully evade the randomization scheme proposed for defending Anagram against adversaries. Indeed, we showed that, rather than improving its security, randomization makes Anagram more vulnerable once the secret mask has been guessed with a reverse engineering attack. The key problem behind randomization techniques to defend IDN nodes is the need to make them compatible with an accurate detection, which are somehow contradictory goals.

The difficulties found to properly defend the nodes of Intrusion Detection Networks against adversaries motivate our research to protect IDNs globally, assuming the possibility of attacks against some nodes and devising ways of allocating countermeasures optimally. In order to do so, it is essential to model IDNs and the adversarial capabilities against them.

In this chapter, we address the problem of establishing an adversarial model for Intrusion Detection Networks. In Chapter 2 we briefly reviewed the question in related works, and showed the importance of understanding the adversarial capabilities in order to implement proper countermeasures. Concretely, in this chapter we present three main contributions:

1. **We provide a conceptual model for IDNs**, viewed as a network of nodes whose connections and roles determine the architecture of the network. We present a system model that considers a node as a logical entity with four functional modules and four information channels. Depending on the responsibilities of the nodes, they may play different roles. We define these roles in terms of the system model and describe various architectures in terms of these roles. Accordingly, the proposed model focuses on the structure of IDNs [Bye et al., 2010] and it does not depend on specific detection or correlation functions implemented in the nodes.

2. **We present an adversarial model for IDNs**. The state of the art discussed in Chapter 2 shows that most of the research on attacks against cyberdefense systems focuses on independent entities, i.e., the IDN nodes. In this work, we focus on adversarial capabilities that affect the communication between nodes. Concretely, these capabilities are the well known basic threats in network security: interception, blocking, fabrication, and modification. Then, using these capabilities, we provide attack strategies that adversaries may use depending on the objective of the attack.

3. **We provide studies of the adversarial capabilities in two common scenarios for IDNs**. The first scenario is a Mobile Ad-hoc Network (MANET), where several nodes are connected using wireless channels without relying on any central management (all nodes must participate for the proper behavior of the network). We provide examples of attacks against different IDNs proposed in the literature to defend MANETs, which were presented in Chapter 2. The second scenario is a Collaborative IDN (CIDN), where different corporations share resources and information to detect distributed attacks. Concretely, we propose a multi-step attack against one specific CIDN: the Distributed Security Operation Center (DSOC) presented by Karim et al. in [Karim-Ganame et al., 2008].

The remaining of this chapter is structured as follows. Section 5.2 presents the system model for nodes in IDNs. Then, Section 5.3 describes the adversarial model proposed for IDNs, which consider single threats to communications. Section 5.4 provides conceptual attacks on two specific scenarios. Finally, we conclude the chapter in Section 5.5.

5.2 System model

Intrusion Detection Networks (IDNs) include a huge variety of approaches. On the one hand, different entities located in different places and sharing information through the Internet form a Collaborative Intrusion Detection Network (CIDN) [Karim-Ganame et al., 2008; Yegneswaran et al., 2004; Zhou et al., 2010]. On the other hand, different wireless sensors monitoring a Mobile Ad-Hoc Network (MANET) [Li et al., 2012; Zhang et al., 2009] also constitute an IDN. Despite their differences in processing and communication capabilities, both approaches detect distributed intrusions by interconnecting detection nodes, and therefore share common functional modules and information channels.

In this section, we define a conceptual model for nodes in an IDN. The main purpose is to establish common building blocks and thus define common threats, in order to define a generic adversarial model (which is provided in next section). Accordingly, we first present a general, functional overview of these nodes in terms of their logical components, functions, and channels. Then, we define the roles of these nodes in the network architecture, according to their responsibilities. Finally, we present the architectures proposed in the literature for IDNs in terms of the system model and the roles presented.

Every node participating in an IDN contains some basic components, depicted in Figure 5.1. We identify four channels of information and four functional modules. Depending on the role of the node in the network, these components may be activated or not, as we discuss later in section 5.2.3. Finally, nodes are connected to other nodes to share information. Depending on how the nodes are connected and their roles, the network may have different architectures.

5.2.1 Channels

We define a communication channel as any input or output interface used by a node to receive or send information. We consider these channels logically, and do not focus on the physical implementation of the channel. Nodes manage three different types of messages:

1. **Intrusion Detection Messages (IDMsg)**. It comprises any exchanged message containing information related to the detection of attacks. The format

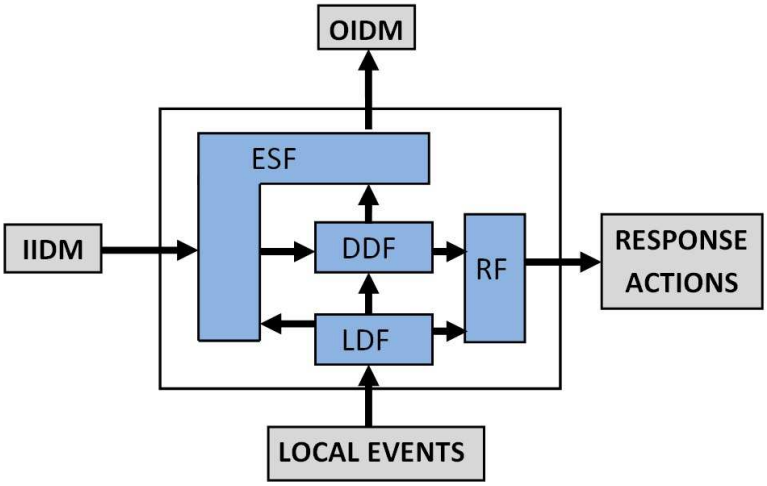


Figure 5.1: Functional view of an IDN node.

of these messages vary for each IDN. For example, the nodes may implement the IDMEF format [Gil-Pérez et al., 2013], which is an RFC standard [Debar et al., 2007] that defines a format for IDS alerts.

2. **Local Events (LE)**. This is the data monitored by sensors locally. It includes both host data (system logs, audit trails, etc.) and network traffic (TCP headers, HTTP payloads, connections, etc.).

3. **Response Actions (RA)**. A response action is triggered whenever a intrusion is detected. This includes activities such as logging alerts in a file, blocking IP addresses or turning devices off, to name a few.

We define four possible channels for nodes, two input channels and two output channels: the Input Intrusion Detection Messages (**IIDM**) to receive IDMsgs from other IDN nodes; the Local Events (**LE**) channel, to gather data locally; the Output Intrusion Detection Messages (**OIDM**), to send IDMsgs to other nodes in the IDN; and the Response Actions (**RA**) channel, from which active responses are triggered.

5.2.2 Functions

Distributed IDN nodes communicate with other nodes to share IDMsgs which may be relevant to detect or block distributed attacks. Every node may run up to four

functions in the detection process. The inputs and outputs of these functions are provided by the channels defined above:

1. **Event Sharing Function (ESF)**. This function manages the communication between nodes in the IDN. Its responsibility in the node is twofold. First, it processes and formats incoming IDMsgs received through the IIDM channel, and provides these data to the Distributed Detection Function (DDF). Second, it processes and formats the output of the DDF and the Local Detection Function (LDF) to send these messages to other nodes through the OIDM channel.

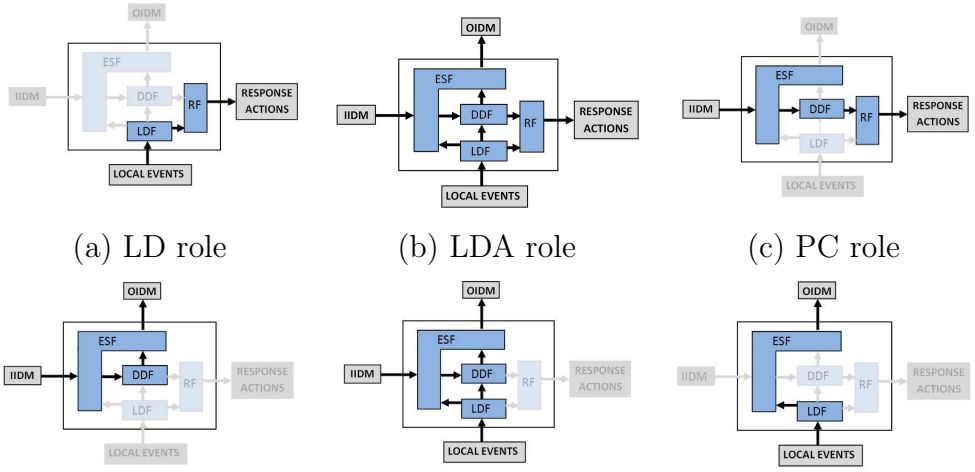
2. **Distributed Detection Function (DDF)**. It aggregates and correlates data from both the LDF and the ESF. Then, the correlated data is sent to the Response Function (RF) if some response is needed, or to the ESF in the case that this data is being shared with other nodes.

3. **Response Function (RF)**. The RF is activated whenever a response action is needed. Among the activities involved in this function are updating blacklists with suspicious IPs, sending remote commands to shut down compromised systems, logging alerts, etc.

4. **Local Detection Function (LDF)**. The LDF uses local data, which may include both host and local network data, to perform local detection. As in the case of the DDF, it may use any of the classical detection techniques employed by IDS, like signature matching, anomaly detection, specification based detection, etc. The output of this function is provided as an input to the other three functions: to the RF if some response action is needed; to the DDF to correlate and aggregate local detection with other IDMsgs; and to the ESF to share the information with other nodes in the IDN.

5.2.3 Node Roles

We define six roles for IDN nodes, which are explained below. Based on these roles, each node activates some or all of the channels and functions, as shown in Figure 5.2:



(a) LD role (b) LDA role (c) PC role

(d) RC role (e) RCD role (f) DC role

Figure 5.2: Logical schemes of roles for IDN nodes

1. **Local Detection (LD)**. The node detects intrusions based on local events only. It gathers local data using the LDF, which includes both network traffic from its LAN and host-based data. Because alerts are neither shared nor received from other nodes, the RF must be enabled to generate the corresponding response action whenever a local intrusion is observed. Figure 5.2-a) shows the schematic view of this role. Classical IDS working independently, like Snort [Roesch, 1999] or Bro [Paxson, 1999], have this role. In IDNs, there are no nodes with this role, because they can neither send nor receive data from other nodes. However, we include them in our study for the sake of completeness.

2. **Local Detection and Alert Sharing (LDA)**. This is the most complex role as it uses all the channels and runs all the functions, as shown in Figure 5.2-b). Nodes perform detection using as input both the local events and the IDMsgs received from other nodes. First, the LDF processes the local data, which are then aggregated and correlated in the DDF with external IDMsgs, after being processed by the ESF. Whenever an intrusion is detected, the RF generates the corresponding response action. Moreover, nodes with this role must share IDMsgs (using the OIDM channel and the ESF) with other nodes in the IDN.

3. **Pure Correlation (PC)**. The node correlates IDMsgs received from other nodes using its IIDM channel and the ESF. It then makes a decision using the DDF and, if needed, generates a response through the RF. Nodes with this

role do not use local data and thus the LE channel is inactive. Well-known SIEM systems [Miller and Pearson, 2011] have this role. Figure 5.2-c) shows the schematic view of this role.

4. **Remote Correlation (RC)**. The nodes receives multiple IDMsgs from other nodes through the IIDM channel and preprocesses them in the ESF. Then, it aggregates and correlates them using the DDF, and share the aggregated data and results from the correlation to other nodes through the OIDM channel. It does not generate any response. Figure 5.2-d) shows the schematic view of this role.

5. **Remote Correlation and Detection (RCD)**. The node performs local detection based on data gathered locally through the LE channel and the IDMsgs received from other nodes through the IIDM channel. Then, it shares the aggregated IDMsgs with other nodes using the OIDM channel, but it does not generate responses (the role is similar to the LDA, but without emitting responses). Figure 5.2-e) shows the schematic view of this role.

6. **Data collection (DC)**. The node is in charge of gathering local events through the LE channel. Then, it processes it with basic filters or routines in the LDF and provides other nodes with the corresponding IDMsgs, using the ESF and the OIDM channel. Figure 5.2-f) shows the schematic view of this role. For example, in a wireless sensor network, different sensors gather data and send it to a central sink for further processing. These sensors may have the DC role, while the sink node would have the PC role.

5.2.4 Node Connections

The system model considers nodes regarding their corresponding functions and channels. Nodes use the ESF, either for sending data (through the OIDM) to other nodes or to receive data (through the IIDM) from other nodes. The nodes are thus connected between them. This connection is unidirectional, i.e., if node A_i sends data to node A_j, then the OIDM channel of A_i is directly connected to the IIDM channel of A_j and the information flows in one direction, from A_i to A_j. In general, we refer to the set of n nodes connected to one node A_i as $C_i = \{A_1, \ldots, A_j, \ldots, A_n\}$, with $j \neq i$.

When considering the information received externally, nodes may either believe this information or they may question its correctness. Accordingly, many approaches in the literature of IDNs propose the concept of trust in the information received by one node [Gil-Pérez et al., 2014; Li et al., 2012]. In our system model, this trust is represented by an influence factor which takes values between 0 and 1. Consequently, each connection is weighted by this influence factor.

For each node, the sum of the influences received from the connected nodes must add up to 1. Concretely, for each node A_i of the IDN, let I_{ij} be the influence from node A_j to node A_i, with $j \neq i$. Thus:

$$\sum_{\forall A_j \in C_i} I_{ij} = 1 \tag{5.1}$$

5.2.5 IDN Architectures

In this section. we review the architectures defined for Intrusion Detection Networks in the literature and analyze them in terms of the proposed system model. A rather complete survey of such architectures can be found in [Zhou et al., 2010].

Centralized

In a centralized architecture, several nodes gather local information and perform local detection. Detection results are sent to a central node. This central node receives multiples IDMsgs, aggregates and correlates them, and performs distributed detection. Figure 5.3 shows the scheme of a centralized architecture with six detection nodes and a central correlation node. Each detection node has the DC role, as they just gather local data, process these data (this may include any routine to obtain a local view from different local sources of data) and send the data with the IDMsg format (using the OIDM channel) to the central node, which has the PC role. This central node, upon the reception of new IDMsgs (using the IIDM channel) performs correlation and aggregation, and activates a response if needed.

Hierarchical

In a hierarchical architecture, the IDN is divided into levels. Nodes in each level gather data received from lower levels or gathered locally, and send aggregated data to the upper level for correlation. In the top level, a global node performs the final

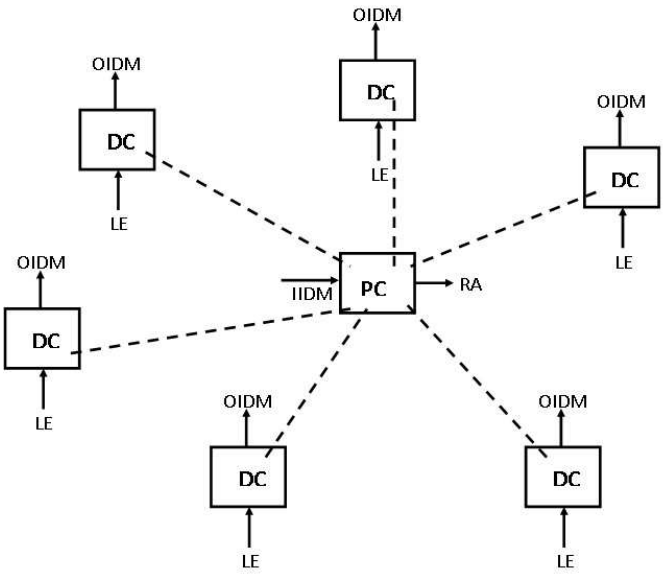


Figure 5.3: Centralized architecture of IDN.

correlation and emits responses if needed. Figure 5.4 shows a hierarchical architecture with three levels. The lowest level is comprised of nodes with the DC role that locally gather and process local events. They send these data in the IDMsgs format to some node in the upper level of the hierarchy, who aggregates and correlates the received IDMsgs. This node, may have the RC role, if it only uses external IDMsgs in the correlation, or the role RCD, if it also includes local data. The correlated data is sent to the upper level, if any, or it is used to emit responses if there are no more levels. Thus, the highest level in the hierarchy contains a single node with the PC role. The response module of PC nodes should command response actions to the lower level nodes through IDMsgs. Thus, the detection events are propagated in a bottom-up design, while the the responses are emitted in a top-down design.

Distributed

Figure 5.5 shows a distributed architecture with five nodes. All nodes have an LDA role. The detection process in the nodes combines local and distributed detection, as it uses both data gathered locally and IDMsgs received from other nodes. Thus, unlike other architectures, in the distributed architecture there are no critical, central nodes with higher responsibilities and all nodes are equally responsible of the cyberdefense. However, because of the large number of connections, attacks to a single node are rapidly propagated to the entire IDN.

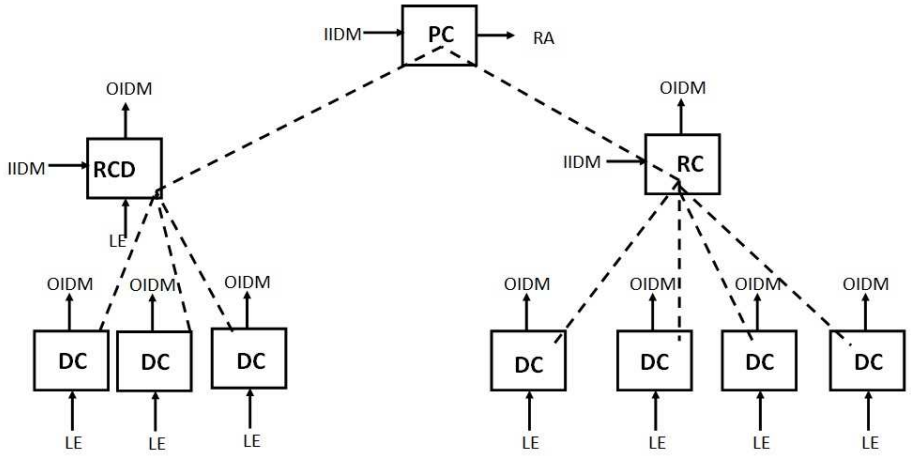


Figure 5.4: Hierarchical architecture of IDN.

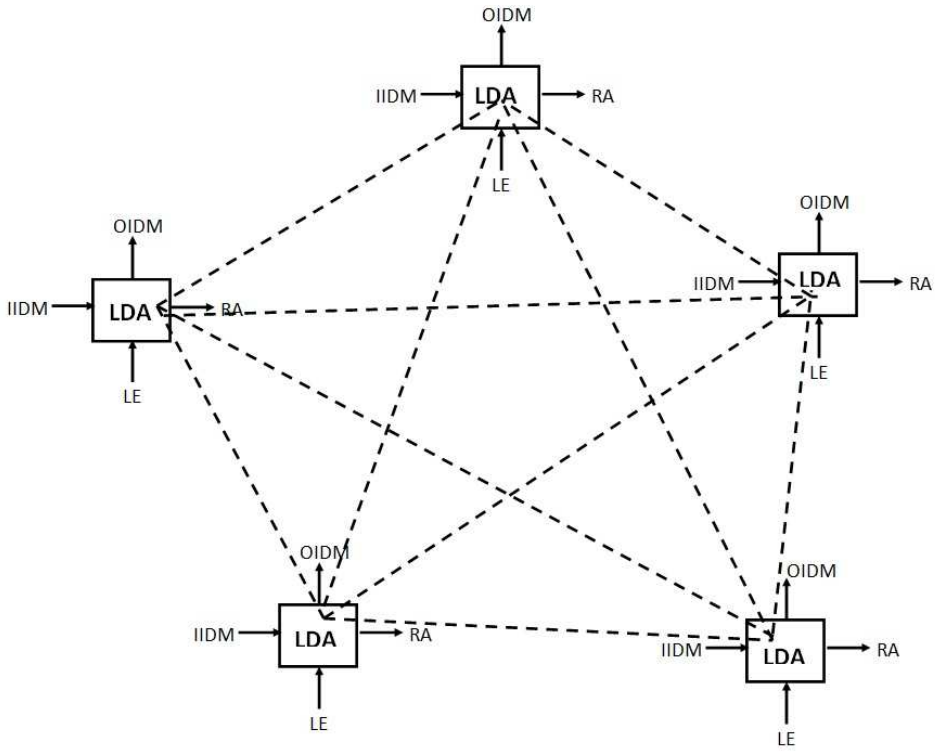


Figure 5.5: Fully-distributed architecture of IDN.

5.3 Adversarial Model

An adversarial model defines the capabilities assumed for an adversary. Though the attack scenario is application-specific and may differ from one IDN to another, it is possible to establish general guidelines to define the adversarial model [Biggio et al., 2013]. Indeed, the abstraction provided by the system model presented above allows us to specify an adversarial model. This section describes the capabilities, in terms of the system model defined in the previous section.

There are two different types of adversaries: external and internal attackers. On the one hand, external adversaries have control of the channels and communications between nodes but are not part of the IDN. Thus, if security protocols are used to provide confidentiality and integrity mechanisms, they may not be able to inject or intercept packets. On the other hand, internal attackers are adversaries who have gained access and have control of, at least, one node within the IDN. They may possess cryptographic keys (if any).

Defending the network from external adversaries can be done using traditional security mechanisms, such as cryptographic protocols and Public Key Infrastructure [Fung, 2011] (PKI) . However, these techniques cannot be afforded in many scenarios. For example, in the Internet the use of a PKI between peers from different institutions may entail administrative conflicts. Actually, when one node receives external information (IDMsgs), it usually cannot determine whether the information is real or it has been forged by the source (i.e., an internal attacker) or manipulated during the communication through the network (by an external attacker). Knowing how much trust can be placed in the received information is one of the key challenges in the design of IDNs [Gil-Pérez et al., 2014]. In the model presented above, this trust is modelled with the use of influence values in the connections between nodes.

In this section, we first describe four basic types of intrusive actions against networks: blocking, modification, interception, and fabrication [Chen et al., 2013]. Then, we provide a description of attack strategies considering the system model presented in Section 5.2.

5.3.1 Basic Intrusive Actions

Nodes in an IDN send and receive data using communication channels, as presented in Section 5.2. The communication consists of the exchange of packets of information using some network protocol and the specific format of the IDMsg. Accordingly, we consider that adversaries in IDNs may perform any of the following intrusive actions:

1. **Interception**. This is a passive attack which compromises the confidentiality of the information. The adversary eavesdrop the contents of the messages transmitted in the channels. For example, a malicious node in a MANET which promiscuously monitors its neighbors, performs interception. This attack is hard to detect, but can be counteracted by cryptographic techniques.

2. **Fabrication**. Fabrication attacks compromise the authenticity of the data. The adversary generates fake data and sends it to the victim. For example, using spoofed addresses, the attacker may fabricate packets that match the signatures of an IDS in order to overstimulate it [Mutz et al., 2003].

3. **Modification**. This attack targets the integrity of the data. The adversary intercepts data, modify its content and forwards it to the actual destination. For example, the adversary may modify the content of an attack to evade the signatures matching process from IDSs [Wagner and Soto, 2002].

4. **Blocking**. This attack targets the integrity and availability of the data. The adversary interrupts the communication or makes it unavailable. Packet Dropping attacks [Li et al., 2012] in MANETs are an example of blocking attacks, where a malicious node drops packets that are supposed to be forwarded.

In the following, we refer to these attacks as *intrusive actions*, in order to distinguish them from more sophisticated attacks, which are described in the next section.

5.3.2 Attack Taxonomy Against IDNs

In this section, we classify the attacks against IDNs in terms of the four functions, four channels, and four intrusive actions presented above. For each function, we

differentiate attacks according to the classification of adversarial attacks against IDS in wired networks proposed by Corona et al. [Corona et al., 2013]. This taxonomy classifies the attacks depending on the adversarial goal (see Chapter 2 for further details). Next we categorize them considering the intrusive actions explained above. Later, in Section 5.4, we apply this taxonomy to different example scenarios.

- **Evasion**. The adversary causes the node to misbehave and stealthily attacks the IDN. It only affects detection functions, i.e., the LDF and DDF, as the objective is to force the actual detection to malfunction. This can be done by means of one of these three intrusive actions:

  1. *Blocking*. In some approaches, the detection process starts when some suspicious packet or message is received, like anomalous routing data in MANETs [Su, 2011]. The attacker may perform the attack blocking these packets in the IDN node to avoid detection.
  2. *Modification*. The attacker carefully modifies the data to hide the intrusion evidence the IDN node is looking for. This way, the adversary can avoid signature matching [Vigna et al., 2004], or it can mimic the statistical properties of a normal model [Fogla and Lee, 2006].
  3. *Fabrication*. The IDN node may be waiting for specific IDMsgs or packets to see whether a node is correctly forwarding packets or not. In such a case, the adversary can generate this packet specifically for the IDN node.

- **Overstimulation**. A set of well-crafted packets are sent to the node to make it trigger a huge amount of responses. Because the objective is to stimulate the system to make it impractical, it can be applied to every function of the nodes. Overstimulation is always performed using *fabrication*, namely, the adversary should generate some specific packet that provokes the node reaction. For example, by fabricating packets that match the signatures of the targeted node [Mutz et al., 2003], the adversary can overwhelm security analysts or overload the IDS resources.

- **Poisoning**. The adversary looks for nodes that update their detection function in real time with new data. The goal is to inject some noise forcing the detection function to learn wrong patterns. Similarly to the evasion goal, this objective is only applicable to the LDF and DDF. Since the objective of the adversary is

to inject specific information in the node, it needs *modification* (of data sent by other nodes in the IDN) or *fabrication* (of new data) attacks. For example, two colluding nodes can report good behavior from each other. This way, their reputation in other IDN nodes would be increased.

- **Denial of Service**. The adversary overloads the resources of the nodes to attack their availability. It may affect the LDF, DDF and ESF, which are the functions receiving external data. To force those functions to stop working, the adversary may either flood them to overload its resources, using *fabrication*, or it can *block* traffic to prevent the node from receiving the required data to work.

- **Response Hijacking**. The adversary sends selected intrusive data to the node, forcing it to generate a specific response. The responses in the nodes are generated in the RF, so this is the only function affected by these attacks. To provoke a specific response in the node, the adversary has to use one of the following techniques:

  1. *Blocking*. As explained above with the evasion, the IDN node may be waiting for specific IDMsgs or packets to confirm that a peer is not malicious. If the adversary blocks this critical data sent by a third peer, the node may erroneously believe that this third peer is malicious.
  2. *Modification*. The adversary may modify reports or IDMsgs to indicate that a third node is malicious.
  3. *Fabrication*. As with the modification, the adversary can generate false reports about a third node to force the detector to trigger an erroneous response.

- **Reverse Engineering**. The adversary gains information about the behavior of the node (architecture, detection function, set of measurements, etc.). It is applicable to every function in the nodes. This could be done using the same techniques employed for an overstimulation attack, but in addition the node must intercept traffic to monitor both the inputs and outputs to the node and make the analysis. A paradigmatic reverse engineering attack in IDNs occurs when the adversary performs traffic analysis of the network in order to locate IDN nodes and responsibilities (for example, to know which nodes

are performing correlation in a hierarchical architecture). Additionally, the attack presented in Chapter 4 shows a query response analysis that allows the adversary to infer secret information used internally by nodes [Pastrana et al., 2014].

In our analysis, we define an Attack Strategy (AS) as a set of techniques that the adversary may use to reach a specific objective. We identify seven different AS:

- AS-1. B $\vee$ M $\vee$ F (Blocking or Modify or Fabricate).
- AS-2. F (Fabricate).
- AS-3. M $\vee$ F (Modify or Fabricate).
- AS-4. (M $\vee$ F) $\wedge$ I (Modify or Fabricate and Intercept).
- AS-5. I (Intercept).
- AS-6. B $\vee$ F (Blocking or Fabricate).
- AS-7. B (Blocking).

Table 5.1 shows which AS should be applied on each channel to target each function. It is important to observe in Figure 5.1 the relationships between functions and channels. For example, for a node with the role LDA, which performs both local and distributed detection, the output of the LDF is given as input to the remaining functions. Thus, attacks targeted against the LDF will affect the other functions as well.

5.4 Attack Scenarios

In Section 5.3 we have proposed a taxonomy of attacks against generic IDNs, indicating for each attack which intrusive actions the adversary can use in each channel. In this section, we provide specific examples of how these attacks could be implemented in two different scenarios: a MANET and a cooperative IDN. In the following, the term *attack* refers to malicious activity against the IDN, and the term *intrusion* to attacks detected by the IDN (e.g., black hole, packet dropping, etc.).

Table 5.1: Taxonomy of attacks. The table shows which attack strategy (AS) should be performed on each channel (LE, IIDM, OIDM and RA) to target each of the functions (LDF, DDF, ESF and RF), depending on the adversarial goal.

4 functions	Objective [Corona et al., 2013]	4 Channels			
		LE	IIDM	OIDM	RA
LDF	Evasion	AS-1	–	–	–
	Overstimulation	AS-2	–	–	–
	Poisoning	AS-3	–	–	–
	Denial of Service	AS-6	–	–	–
	Reverse Engineering	AS-4	–	AS-5	AS-5
DDF	Evasion	AS-1	AS-1	–	–
	Overstimulation	AS-2	AS-2	–	–
	Poisoning	AS-3	AS-3	–	–
	Denial of Service	AS-6	AS-6	–	–
	Reverse Engineering	AS-4	AS-4	AS-5	AS-5
ESF	Overstimulation	AS-2	AS-2	AS-3	–
	Denial of Service	AS-6	AS-6	AS-7	–
	Reverse Engineering	AS-3	AS-3	AS-5	–
RF	Overstimulation	AS-2	AS-2	–	AS-2
	Reverse Engineering	AS-4	AS-4	–	AS-5
	Response Hijacking	AS-1	AS-1	–	AS-1

5.4.1 Scenario 1: IDNs in MANET

In MANETs, the detection is distributed among different nodes in the IDN. In this section, we first present the technical capabilities assumed for the adversary as well as some notation used to explain the attacks. Next, using these technical capabilities, we present methods to perform each of the four intrusive actions against the channels presented above. Finally, we present specific attacks targeted against IDNs proposed in the literature for MANETs.

5.4.1.1 Technical Assumptions and Notation

Nodes in MANETs use an antenna that emits radio waves to transmit data in the wireless medium. Data is usually sent uniformly in all directions with an omnidirectional antenna. A sender node has a transmission range that depends on the power of the signal and the noise in the channel. Once the data is transmitted, every node located within the transmission range of the sender is able to receive it.

We assume that the adversary is able to send information in one preferred direction and with a specific, chosen transmission range. As done in [Johnston and Narula-Tam, 2012], we consider that a node knows exactly the position of every node

within its transmission range, and thus it can choose the receivers of the packets transmitted. We present two techniques an adversary can use for this purpose:

1. *Using a directional antenna* [Takai et al., 2002]. A directional antenna emits radio waves just in one direction. For example, nodes can use a metallic panel oriented to reflect all the radio waves to the desired destination. This situation is shown in Figure 5.6.

2. *Controlling the signal power.* The adversary can modify the signal power to cover a specific transmission range. For example, if two nodes A and B are at distance D_A and D_B from M, respectively, with $D_A \leq D_B$, the adversary may set the communication range of M to a value C such that $D_A \leq C \leq D_B$. This way, the information sent from M will arrive at A but not at B. This situation is shown in Figure 5.7.

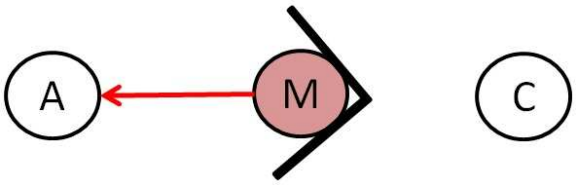


Figure 5.6: Unidirectional antenna in node M.

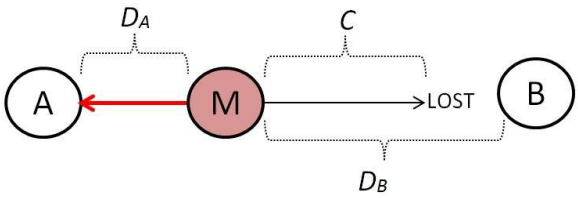


Figure 5.7: Transmission control by node M.
Values D_A and D_B are, respectively, the distances from A and B to M, and C is the communication range of M.

According to the assumptions made above, nodes are able to either send data in every direction of the antenna ($SEND$) or just in one direction (directional send, $DSEND$), with a specific transmission range. They are also able to receive data specifically sent to them or to promiscuously monitor data in the neighborhood. We next describe the notation used:

- $SEND(A, B, P)$. Node A sends the packet P to node B. P arrives to every node within the transmission range of A.

- $DSEND(A, B, P)$. Node A sends the packet P only in the direction of node B, within a specific transmission range. Neighbors located in any other direction or within a distance farther than B do not receive P.

- $RCV(A, B, P)$. Node A receives the packet P from node B.

- $MONITOR(A, B, C, P)$. Node A monitors the packet P, sent from node B to node C.

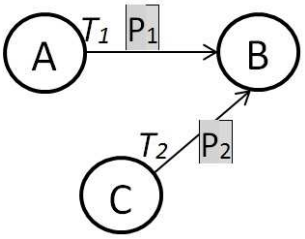
- $COLLISION(B, P_1, P_2)$. There is a collision in node B because of the simultaneous reception of packets P_1 and P_2. A collision occurs when a node receives a signal from different sources in the same channel. This situation is shown in Figure 5.8. Node B receives at the same time the packets sent from nodes C and A. In this case, B is not able to process and obtain the information received. In the absence of adversaries all nodes are cooperative, and whenever a collision is detected, the sender node sends the data again. However, this assumption is no longer valid in the presence of adversaries that provoke collisions.

5.4.1.2 Intrusive Actions in MANETs

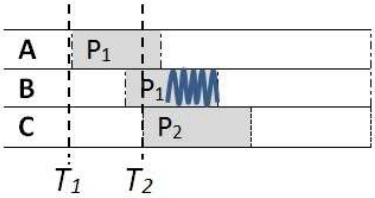
In this subsection, we describe how general attack techniques are performed in MANETs according to the channels affected.

Interception

Due to the mobility of nodes in a MANET, a malicious node M can be placed in the middle of two nodes A and B. Because data is transmitted over a wireless medium, M can easily intercept the communication by monitoring the traffic between A and B. Cryptography is not typically used in MANETs due to resource constraints. Interception is hard to detect, because it is passive and malicious nodes behave according to the routing protocol. Therefore, the adversary can monitor both the input channels (LE and IIDM) and output channels (OIDM and RA) of the nodes.



(a) Graph view of the network.



(b) Slot-time diagram. Each line corresponds to the medium view of the nodes.

Figure 5.8: Example of a collision.
In time T_1, node A sends P_1 to B, and in time T_2 node C sends packet P_2 to B. If $T_2 - T_1 \leq T_P$, T_P being the time spent for a the packet P_1 to reach B from A, then a collision is produced.

Fabrication

A fabrication attack occurs when the adversary generates data that in normal conditions would not exist. A common technique for such a purpose is to perform spoofing attacks, where the adversary counterfeits the source address when sending data to its victim. Typically, authentication is used to counteract against spoofing. However, in MANETs there are further methods to perform fabrication. Let us suppose that a malicious node M sends a packet P directly to a victim node V, i.e., $DSEND(M, V, P)$, using one of the techniques explained in Section 5.4.1.1. If the packet P has a destination address different from V (for instance, C), and V monitors the packet, i.e., $MONITOR(V, M, C, P)$, V would think that the packet is being sent to C.

Even if M does not have a directional antenna, it can prevent node C from receiving P by interrupting its own packet after sending it, as we explain below for the blocking case. This is a case of fabrication in the input channels of V (LE and IIDM). Moreover, if the ESF is active in the victim node, the fabrication of well-crafted LE and IIDM will induce the node to fabricate the corresponding OIDM and RA. We call this a *cross-channel fabrication*.

Modification

To perform a modification attack, the adversary first needs to intercept the communication. Afterwards, the adversary modifies the content of the intercepted data and forwards it to the actual destination. Again, defending the IDN nodes from these actions is done using cryptographic mechanisms. However, these mechanisms are often unsuitable because they consume resources. The channel affected by the attack depends on the data being modified. For example, if the adversary modifies a report of good behavior, it is targeting the IIDM channel. Similarly, if the adversary modifies the sequence number of a routing packet, it is targeting the LE channel of the victim.

Blocking

Blocking occurs when the adversary interrupts the information sent to its victim by any other node. The simplest way to implement a blocking attack in MANETs is by dropping packets. There are several approaches to detect packet dropping attacks [Djenouri et al., 2007], and thus the adversary should stealthy perform this attack. In Section 5.4, we present some evasion attacks for this purpose.

A more sophisticated way to perform blocking is to provoke collisions in the victim node C. If C is receiving the packet P from other node, a malicious node M may send a second packet P' and provoke a collision, i.e., $COLLISION(C, P, P')$. The overall effect is a blocking of packets sent by a third node to the victim C. In some cases, the malicious node M may want to block its own packets in the victim C. In this case, M can wait until it monitors a third node sending a packet to C. Once it observes that the channel is busy, M starts sending the information. This situation is shown in Figure 5.9 and described in Algorithm 1. As mentioned above, this entails a fabrication in A, because M induces A to monitor the packet P while it is not actually reaching C. In the next section we use these methods to describe evasion attacks for the packet dropping scenario.

5.4.1.3 Attacks on MANETs

IDNs in MANETs provide security against a large variety of attacks [Xenakis et al., 2011], reporting good results in terms of efficacy and performance. However, they have been designed to defend against an adversary interested in the assets of the MANET, while not taking into account the possibility of being attacked themselves.

In this section, we provide examples of attacks against IDNs using the actions presented in Section 5.4.1.2.

Evasion

An evasion attack succeeds when a IDN node is not able to detect a misbehaving node. As shown in Table 5.1, evasion is possible in the LDF and DDF using the Attack Strategy 1 (AS-1), in which the adversary should either block, modify, or fabricate data in the LE and IIDM channels of the nodes. Next, we provide examples of evasion against some of the state of the art proposals for IDNs in MANETs presented in Chapter 2:

1. *[Marti et al., 2000].* Suppose that an IDN node A is monitoring a malicious node M to detect whether it forwards a packet P to a third node C or not (*Watchdog*). To evade detection, the adversary has two options:

   - Blocking packet P in C. The malicious node M causes a collision of packet P in the receiver node C, just by waiting for a third node B to use the wireless channel, as represented in Figure 5.9 and described in Algorithm 1. We call this attack *receiver_collision(M,A,C,P).* The victim node A actually monitors the packet P being sent from node B to node C. However, due to the collision, the packet P never reaches the destination (node C), so it can be considered that M fabricates the packet P to A.
   - Fabrication. M can send the packet P in such a way that it reaches A but not C, i.e., $DSEND(M, A, P)$. This way, A would believe that P is forwarded from M to C, i.e., $MONITOR(A, M, C, P)$, while packet P is actually not reaching C.

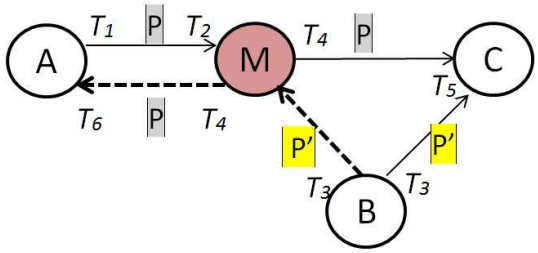
2. *[Kurosawa et al., 2007].* In this work, an IDN node looks for either an anomalous number of route request (RREQ) and route reply (RREP) packets sent or anomalous Sequence Numbers (SN) in these packets. Evasion can be done using some of the following techniques:

   - Blocking. If the adversary blocks RREQ and RREP packets sent to the victim nodes, then the number of packets observed by the IDN nodes would be lower than the actual number of packets sent by the adversary.

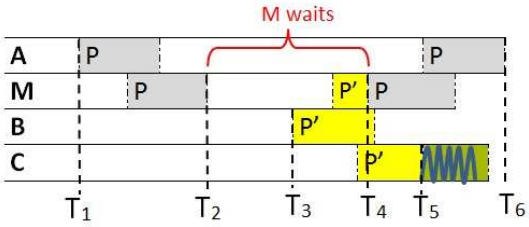
Algorithm 1 *receiver_collision*(M, A, C, P)

$M \rightarrow$ The malicious node
$A \rightarrow$ Node running an IDN node to monitor M
$C \rightarrow$ The victim node (receiver)
$P \rightarrow$ Packet routed from A to C (must be forwarded by M)

$T_1 : SEND(A, M, P)$
$T_2 : RCV(M, A, P)$
[M waits...]
$T_3 : SEND(B, C, P')$
$T_4 : SEND(M, C, P)$
$T_5 : COLLISION(C, P, P') \rightarrow RCV(C, M, P) + RCV(C, B, P')$
$T_6 : MONITOR(A, M, C, P)$



(a) Graph view. A dotted line represent that a node is monitoring the communication and solid lines represent actual communications between nodes



(b) Slot-time diagram. Each line corresponds with the medium view of the nodes.

Figure 5.9: Receiver collision

- Modification. The adversary may change the SN in such a way that it does not exceed the anomaly threshold. This requires that the adversary knows such a threshold, which can be obtaining using a reverse engineering attack.

3. *[Su, 2011]*. In this approach, the detection process of node A starts when it monitors a RREP packet from a monitored node B. Then, A checks whether

B has previously forwarded the corresponding RREQ, checking the sequence number and source IP of the packet stored in a table. If the check fails, A considers B as malicious. Therefore, an adversary in B can perform evasion using the following techniques:

- Blocking. Interrupting RREP in node A to prevent the process from starting.
- Modification. Varying the SN to modify the internal table and thus avoid detection. This is actually a poisoning attack because the detection data is specifically modified, as we explain below. However, if the poisoning is repeated for a while, then it may turn into an evasion.
- Fabrication. Generating RREQ packets in the node A in order to insert an RREQ packet with a SN that the adversary uses in a posterior RREP when performing a black hole attack. This way, when node A monitors this RREP packet, it may check its table and it will see a corresponding RREQ with the same SN.

4. *[Huang and Lee, 2003].* The authors propose a hierarchical approach where the network is divided into clusters. Each cluster head aggregates data received from nodes within its cluster and correlates it with data received from other cluster heads. First, it performs anomaly detection and, if anomalies are detected, a second rule-based process identifies the attack and the malicious nodes. If the adversary is able to evade the anomaly detection phase, it may not be detected. The evasion attack targets the IIDM channel of the cluster heads. Accordingly, it is needed a preliminary reverse engineering attack to identify which nodes act as cluster heads.

   - Blocking. By continuously interrupting the data sent to the cluster head, the adversary can isolate it and thus it may not be able to perform correlation.
   - Fabrication. If the adversary has compromised a cluster head, it can use it to inject false information to other cluster heads. This is a poisoning attack that can be used for evasion purposes.
   - Modification. In a scenario where a malicious node has been detected by a cluster head, the adversary can modify the corresponding IDMsg sent to

others cluster heads and remove its malicious behavior from the reports.

5. *[Zhang et al., 2009].* The authors propose a fully distributed architecture where IDN nodes share data to detect black hole intrusions. In their paper, authors state that collaboration between IDN nodes can detect intrusions even in the presence of various colluding malicious nodes.

   - Blocking. The adversary must interrupt the IDMsgs exchanged between IDN nodes. The authors assume that nodes cannot use a directional antenna and they all have the same transmission power, which is not a realistic assumption. However, even with these assumptions, the adversary is able to evade this scheme with two colluding nodes by provoking collisions deliberately. Normally, when a malicious node sends a forged SREP to the IDN node (victim), a collaborative node within the neighborhood notices that the SREP has been changed, because it monitors both the original and the forged SREPs. As shown in Figure 5.10, the adversary can produce a collision using two colluding malicious nodes by means of a new packet which performs the blocking attack (collision). First, in the precise moment that $M2$ sends the original SREP, $M1$ sends a packet to A to provoke the collision (represented with a red arrow in Figure 5.10a). Then, when A receives the forged SREP from $M2$ (Figure 5.10b), it is not able to compare both SREP numbers and will not notice the change.
   - Fabrication. If we do not consider the assumptions from [Zhang et al., 2009], $M1$ can use a $DSEND$ to prevent A from monitoring the forged SREP.

Overstimulation

The goal in an overstimulation attack is to induce the victim to initiate response actions (RA). The adversary may use a malicious node to perform fabrication against the IDN node, and induce it to generate responses. Then, it can use these responses in a variety of ways. For example, the adversary can intercept the responses and infer the anomaly thresholds used by means of a reverse engineering attack.

By overstimulating a victim node, the adversary can force it to modify the packet routes with some purpose. In some schemes detecting packet dropping, like

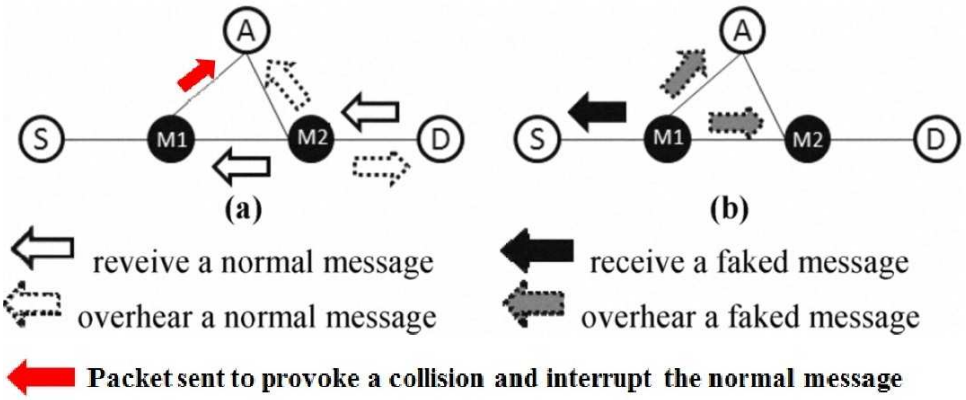


Figure 5.10: Modification of the scheme of [Zhang et al., 2009] to cause a collision and evade the cooperative detection.

Pathrater, explained in [Marti et al., 2000], or Modified AODV (MAODV), from [Su, 2011], the response action consists of modifying the routing protocol to avoid routes containing malicious nodes. An adversary can get information about the IDN topology and architecture using a preliminary reverse engineering attack. Then, she can overstimulate the victim node provoking that certain routes are modified, as shown in Figure 5.11. In this figure, node M overstimulates A to avoid the optimal route (represented in dashed line), because it includes node M which is blocked. Thus, in order to communicate with C, node A uses the alternative path $B_1 \rightarrow B_2 \rightarrow B_3 \rightarrow C$. The benefits for the adversary depend on its goal. For example, it will increase the overall bandwidth of the network or force a specific node in the MANET to process extra packets, thus causing a DoS against it (like B_1 in Figure 5.11), as we explain later.

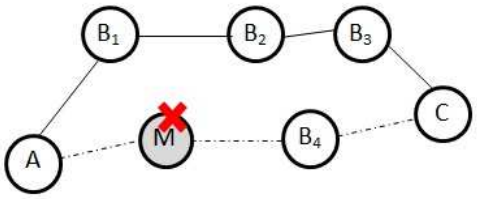


Figure 5.11: Overstimulation attack to make a route selection.
IDN node A is overstimulated to avoid the optimal route (represented in dashed line), because it includes node M, which is blocked.

If the overstimulation is applied to the LDF, then it just affects a single node. In

cooperative architectures, IDMsgs are exchanged between nodes through the network, which increments the communication overhead of the network. The adversary can exploit this using fabrication and overstimulate the ESF of a single node. Because the propagation of information in IDNs, the effect would spread over the entire network. Figure 5.12 shows this situation. Nodes with a double circle represent IDN nodes, and M is the malicious node performing the overstimulation attack. The overstimulation of the node in the left by M provokes an IDMsg to be shared with other IDN nodes in the network. If this situation is continuously repeated, then the network may become overloaded.

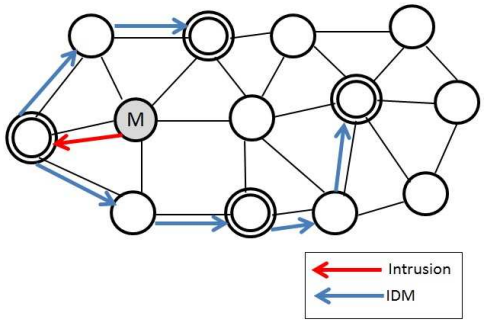


Figure 5.12: Distributed overstimulation attack.
The overstimulation of the node in the left by M provokes an IDMsg to be shared with other nodes in the network.

Poisoning

Poisoning attacks comprise any action produced by the adversary to intentionally modify the detection function or the data used in the detection. In MANETs, IDN nodes store information monitored from nodes in its neighborhoods, like the local view of neighbors and scores of good behaviors [Li et al., 2012]; or the number of RREQ and RREP monitored [Kurosawa et al., 2007]. This information is modified according to the activity monitored in the LE channel and the information received from other nodes by the IIDM channel. An adversary can use fabrication to inject noise into the IDN node, forcing it to store specific, well-crafted information. Afterwards, it can take advantage of this and perform a more severe attack. For example, as explained in Section 5.4.1.3, in the work of Su et al. [Su, 2011] the fabrication of RREQ packets or the modification of sequence numbers allows the adversary to inject noise in the nodes and subsequently evade the detection of black hole intrusions when sending the corresponding RREP packet. In cooperative schemes, where IDN nodes exchange

information about malicious nodes, the adversary may fabricate false reports or modify reports from other nodes. For example, in the scheme of Zhang et al. [Zhang et al., 2009], the adversary can fabricate IDMsgs declaring bad behavior of a benign node, thus decreasing its reputation value. Moreover, a more powerful adversary controlling two or more nodes can collude to talk well about each other.

Poisoning attacks are also useful in adaptive approaches where the detection adapts to the activity of the network. For example, in the approach of Kurosawa et al. [Kurosawa et al., 2007], IDN nodes periodically recalculate the model of normal behavior. This model of normality is the average of the normal events recorded during a previous period, considering "normal" any activity that is measured below a given threshold. The adversary can fabricate packets with high anomaly score but which are below the threshold (again, a preliminary reverse engineering is needed to know which the threshold is). By doing so, the threshold will be increased in the next update. Therefore, step by step the adversary is able to smoothly increase the accepted level of anomaly values.

Denial of Service

A Denial of Service (DoS) attack prevents the victim node from performing detection. In MANETs it is straightforward to perform a DoS attack by forcing the IDN node to consume its battery power. For instance, by continuously performing fabrication against the victim, the adversary causes this node to consume its battery power or resources (flooding attacks [Yi et al., 2006]). Moreover, due to cooperation and routing in MANETs, it is not necessary that the adversary itself sends the data to overload the IDN node. For example, the adversary can use an overstimulation attack to redirect several routes to a victim IDN node, forcing this node to process more packets (see Figure 5.11).

Additionally, some approaches use internal data structures to track the monitored data. A DoS occurs if the adversary is able to overload these structures. For example, in [Su, 2011], IDN nodes use a table with the monitored RREQ and RREP packets from each neighbor. Each time that a new tuple *[src node, dst node, sequence number]* is observed, a new entry in the table is added. The adversary can continuously fabricate packets with different sequence numbers to fill it up and overload this table.

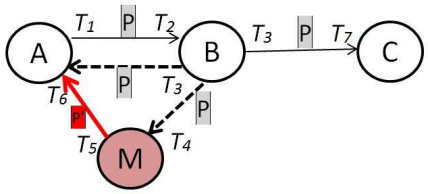
Another specific DoS occurs in a packet dropping detection if the adversary interferes with an IDN node monitoring a node B. Let us consider that A is monitoring

B to watch whether it forwards packets appropriately. If a malicious node M continuously interrupts packets in A that B is actually forwarding, the node A cannot determine whether B forwarded the packet or not. A $monitor\_collision(M, A, B, P)$ attack is shown in Figure 5.13 and described in Algorithm 2. In time T_3, node B sends the packet P to node C, which actually receives the packet in time T_7. However, due to the collision provoked by M in time T_6, A is not able to monitor this communication and thus cannot determine whether B forwarded the packet or not.

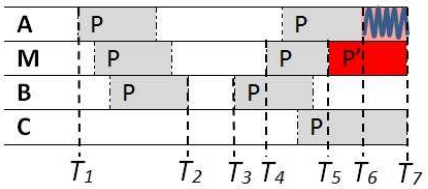
Algorithm 2 $monitor\_collision(M, A, B, P)$

$M \rightarrow$ Malicious node.
$A \rightarrow$ Victim node; it runs an IDN node to monitor B.
$B \rightarrow$ Monitored node.
$P \rightarrow$ Packet sent from A to B to be forwarded to C.
$P' \rightarrow$ Packet fabricated by M to provoke a collision in A.

$T_1 : SEND(A, B, P)$
$T_2 : RCV(B, A, P)$
$T_3 : SEND(B, C, P)$
$T_4 : MONITOR(M, B, C, P)$
$T_5 : DSEND(M, A, P')$
$T_6 : COLLISION(A, P, P') \rightarrow RCV(A, M, P') + MONITOR(A, B, C, P)$
$T_7 : RCV(C, B, P)$



(a) Graph view. A dotted line represents a node is monitoring the corresponding communication and solid lines represent actual communications between nodes.



(b) Slot-time diagram. Each row corresponds to a node channel view.

Figure 5.13: Monitor collision attack.

The adversary can cause a DoS attack to the ESF by continuously blocking IDMsgs destined to the victim node. Thus, the victim is isolated from the network, as shown in Figure 5.14. If the isolated IDN node requires some IDMsg acknowledgement to detect any of the malicious nodes (M), it would never receive such an IDMsg. Another DoS against IDN nodes occurs if the adversary overloads the IDN through an overstimulation attack, as explained above. In this case, the entire MANET would suffer from a DoS, and no IDMsgs could be shared.

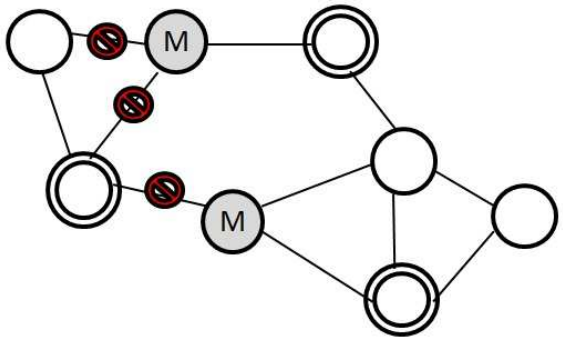


Figure 5.14: Denial of Service attack by means of isolation.

Response Hijacking

In a response hijacking attack, the adversary induces the RF of the IDN nodes to generate false response actions which may affect itself or a third victim node. The severity of this attack depends on the type of response action performed by the IDN node. A malicious node can block packets and prevent the IDN node from monitoring some required information. For instance, if packet dropping detection is performed by A in the scenario shown in Figure 5.13, node A monitors node B to see whether it forwards the packet P to C. Node M can block the packet P to prevent A from observing the packet forwarded by B. As explained above, A is victim of a DoS. However, after sending several packets and waiting for B to forward them, node A may decide that B has dropped all the packets and thus consider it malicious. In such a case the DoS attack turns into a response hijacking attack. Similarly, in the black hole detection scheme presented in [Su, 2011], the attacker may block RREQ packets in A each time that a benign node (say, B) forwards these RREQ packets. Due to the working procedure of [Su, 2011], when B forwards an RREP packet, A does not notice any RREQ previously forwarded by B (due to the blocking action), and therefore the intrusion score of B would be increased. This is a

case of poisoning attack, because only the reputation score in the table is increased. However, when this score exceeds a threshold, node B would be considered malicious, and an erroneous response would trigger a response hijacking attack.

When IDN nodes share IDMsgs containing reports of bad behavior, like in [Zhang et al., 2009], the adversary can poison the view of an IDN node by continuously fabricating false IDMsg packets about a benign node (victim). After the fabrication of several fake reports, the IDN node may classify the victim node as malicious.

Reverse Engineering

A reverse engineering attack allows the adversary to gather information about the detection procedure or architecture used by the IDN in the MANET. We identify two possibilities:

1. Localization of the dominating set. Using traffic analysis attacks, the adversary can discover the roles and localization of IDN nodes. This is rather useful information to perform more serious attacks.

2. Interception. In MANETs, detection is performed by monitoring suspicious activity from other nodes. Thus, it is quite difficult to detect intruders that apparently behave according to the routing protocol, but are actually performing passive attacks. This gives advantage to an adversary to mislead the IDN. By promiscuously monitoring IDN nodes, the adversary can perform a query-response analysis to obtain information about the internal detection function. This way, the adversary may be able to find out the thresholds used in the detection process or the role and responsibilities of the nodes in the IDN. Moreover, due to cooperative nature of the IDN, each time an IDN node observes a malicious behavior, it may fabricate IDMsgs to inform other IDN nodes. Therefore, it is easy for an adversary to query the node and force it to output alerts. Then, using interception, the adversary can obtain the responses, analyze them, and gain information about the detection process. Indeed, in Chapters 3 and 4 we have presented two reverse engineering attacks against IDS that exploit this technique.

5.4.2 Scenario 2: Distributed Security Operation Center (DSOC)

In this scenario we provide two ways to attack a cooperative IDN, namely DSOC, presented by Karim et al. in [Karim-Ganame et al., 2008]. DSOC is a hierarchical approach composed of different node roles. In the lower level, nodes called CBoxes are responsible of gathering local data from local systems and networks in a specific site, which can be either a LAN, a WAN, or a big corporate network geographically distributed. These CBoxes format the data according to some standard (corresponding to the IDMsg in our system model) and send it to a Local Intrusion Database (LIDB), located in the same site. In each site, an LA node reads the logs from the LIDB and performs correlation, sending the alerts to a Global Intrusion Database (GIDB) located in a different site (using a SSL tunnel). Finally, a Global Analyzer (GA) analyzes alerts from the GIDB, correlates them, and performs distributed detection. There are also special CBoxes that gather data from critical sensors. The special CBoxes send their data to a Local Analyzer (LA) situated in a different site, using an SSL tunnel through the Internet. These Cboxes are called Remote Cboxes (R-Cbox).

Next, we explain how the DSOC architecture fits within the system model presented in Section 5.2. The Cboxes have the role of Data Collection (DC). They only use the LE channel to gather local information and the OIDM channel to transmit the data to the LIDB. LA nodes have the role of Remote Correlation (RC), which takes logs from the LIDB using the IIDM channel, performs correlation, and send alerts to the GA using the OIDM channel. Additionally, if a system runs both an LA node and a CBox node, its role would be Remote Correlation and Detection (RCD). Finally, the GA node has the role of Pure Correlation (PC), since it only retrieves data from the GIDB (through the IIDM channel), performs distributed detection, and generates responses through the RA channel. It can be observed that the architecture of DSOC [Karim-Ganame et al., 2008] matches perfectly with our representation of a hierarchical architecture shown in Figure 5.4. In DSOC, the communications between the middle and top layers are encrypted, as well as some of the communications from the lower to the middle layers (i.e., the ones that correspond to R-CBoxes). Accordingly, below we provide a set of attacks against hierarchical architectures, giving specific instances of how these attacks could be

implemented against DSOC.

First, we explain which specific intrusive actions from the adversarial model presented in Section 5.3 are required to perform different attacks against DSOC, which are then used to perform a combined attack.

1. Fabrication (1). The adversary is capable of sending malicious data to every node located in its LAN.

2. Fabrication (2). The adversary can spoof the source address of the transmitted data in the site or modify its identity (IP or MAC address) at will.

3. Interception. The site traffic is non-encrypted and, thus, the adversary can perform man-in-the-middle attacks to intercept the traffic sent by any node in the site to the Internet. Even though the data is encrypted the addresses are sent in cleartext, and thus the adversary is still able to know the identities of the sender and the receiver nodes.

4. Blocking. Same as in the case above, the adversary can use a man-in-the-middle attack to drop packets sent to Internet.

Using these actions, a combined attack is presented that isolates a site from the rest of the hierarchical architecture and thwarts alert sharing. Thus, the goal is to produce a denial of service targeting the ESF function. The attack is composed of three correlative sub-attacks with different goals. These sub-attacks are depicted in Figure 5.15, which has been obtained from [Karim-Ganame et al., 2008] and modified to incorporate our attack:

1. **Overstimulation** attack on the ESF of the DC nodes in the site (the CBoxes in DSOC). As a consequence of this attack, the DC nodes will produce several IDMsgs which are sent via the OIDM channel to the RC node (in DSOC, it is implemented by storing logs in the LIDB). For such a purpose, according to Table 5.1 the adversary should use the Attack Strategy AS-2 on the LE channel, which requires fabrication of traffic to stimulate the IDN. In DSOC, the adversary may fabricate malicious data to some of the systems connected to the site (there is no need of targeting all of them), in order to stimulate the CBoxes.

2. **Reverse Engineering** attack to know which systems implement IDN nodes with the RC role. The goal is to discover what nodes running the ESF share alerts with the PC node in the top of the hierarchy (in DSOC, this means discovering which systems runs the LA and R-CBoxes). The adversary should implement the Attack Strategy AS-5, which means intercepting the OIDM channel to discover who is responding to the previous overstimulation attack. In DSOC, the adversary should perform a man-in-the-middle attack to the Internet access point (router) of the site under attack and perform traffic analysis of the systems sending information to Internet.

3. **Denial of Service** attack against the ESF of these RC nodes. Once an adversary discovers which are the RC nodes, she may try to deny the service of the ESF in order to isolate the site from the rest of the IDN. It uses the Attack Strategy AS-7 on the OIDM (see Table 5.1), which in DSOC means that the alerts sent from the LA and R-CBox to other sites through Internet are blocked. Once the denial of service attack succeeds, the RC nodes are blocked and the site is isolated from the IDN.

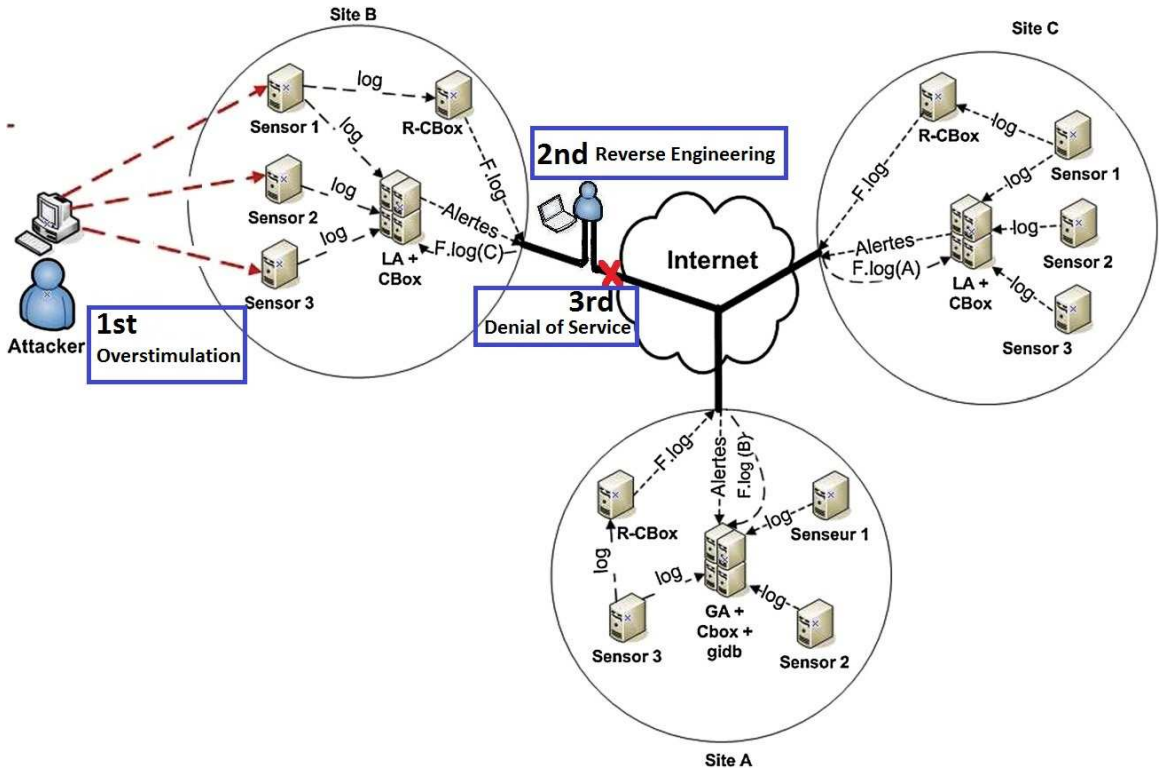


Figure 5.15: Isolation attack of a site on the DSOC architecture [Karim-Ganame et al., 2008].

5.5 Conclusions

Intrusion Detection Networks constitute a key component of current cyberdefense infrastructures to detect distributed attacks. They facilitate different entities to share information about attacks detected within their local networks. This make IDN an effective countermeasure for detecting complex intrusion scenarios like distributed DoS or multi-step attacks. IDNs are composed of different nodes interconnected, and thus they offer various entry points for potential adversaries.

Research on IDNs typically considers the way that detection is performed, i.e., how data is correlated, detection functions, trustworthiness algorithms, etc. However, it is also required to assess the security from the structural point of view. Indeed, it is unclear how the attacks targeting nodes placed in one part of the IDN would affect nodes located in other areas or performing different functions. This may depend on several factors, like the IDN architecture and the adversarial model considered.

In this chapter, we have presented a system model for IDNs. The model considers common building blocks that we have observed in the works proposed in the literature. The abstraction of the IDN allows for the definition of common threats. Concretely, we consider four basic threats in network communications: blocking, modification, interception and fabrication. The likelihood of these threats to materialize depends on the adversarial model considered. The strongest adversary may be able to perform any of these intrusive actions in every node, while a weaker adversary may be only capable of intercepting data in certain network locations.

By means of the system model and the basic intrusive actions defined, we have provided a comprehensive set of attacks to IDNs, following the taxonomy presented by Corona et al. [Corona et al., 2013]. We have provided examples of such attacks for two different scenarios. The first includes a review of different defense systems proposed for IDNs in Mobile Ad Hoc Networks (MANETs), and the second is a collaborative IDN where different entities collaborate to detect distributed attacks [Karim-Ganame et al., 2008].

Both the system model and the attack description presented in this chapter allows for further investigation of IDNs. For example, given a common adversarial model, it could be studied which architectures are more robust, or which are the settings that make the IDN more secure. Additionally, when deciding where to implement countermeasures in the IDN, the system model also allows to perform quantitative

studies in terms of cost and risk. The next chapter addresses some of these problems and presents a framework that leverages the system model presented in this chapter to facilitate the design of resilient IDNs.

6

Optimal Allocation of Countermeasures in Intrusion Detection Networks

6.1 Introduction

In this chapter we present DEFIDNET, a framework to evaluate the risk of IDNs and to optimally set countermeasures to mitigate it. Due to the connectivity of nodes in IDNs, threats affecting one node may propagate and affect the entire network. To completely mitigate such threats, it is required the protection of every single node which is at risk. In many scenarios this is not possible due to resource constraints (in terms of time, human and financial costs, etc.) and the problem turns into investing in security measures optimally to minimize the residual risk. Similarly, given an acceptable level of the residual risk, the budget spent can be minimized.

In this sense, the placement of countermeasures in IDNs plays an important role. Deciding what has to be protected, i.e., where to allocate countermeasures, is a complex task. This decision depends on many factors, such as the adversarial model, the impact of the attacks, or the cost of implementing the countermeasures. These factors vary considerably even within the same network. For example, the cost in terms of time and energy of implementing cryptography mechanisms for wireless communications is different depending on the operating system and the software used [Almenares et al., 2013]. Similarly, the impact of a denial of service attack on a Security Operations Center (SOC) is typically higher than that of an anti-virus solution for a personal computer.

Due to the wide variety and complexity of IDNs proposed in the literature, the risk-rating of these networks is typically done in an ad-hoc manner, what makes it expensive and error prone. Thus, in this chapter we develop a generic risk rating framework, called DEFIDNET. This framework uses the system model presented in Chapter 5 and tailors it to specific IDNs, by means of the customization of different factors of the network and the adversarial model. Then, it incorporates procedures to asses the risk of the IDN. Concretely, considering that some nodes may be targeted, the framework evaluates the propagation of intrusive actions through the network considering the influences between nodes. It then estimates the impacts of these actions regarding different attack strategies. Accordingly, the risk of the network is calculated considering the impacts and likelihood of attacks. Finally, the framework provides the set of optimal countermeasures in terms of cost and mitigated risk. To show the benefits of DEFIDNET, we provide experimental results using different network architectures and a case study using a complex cooperative network.

The chapter is organized as follows. In Section 6.2 we describe the framework in detail, including the threat, risk-rating and allocation modules. We provide the experimental results in Section 6.3, and in Section 6.4 we present a case study. Finally, Section 6.5 concludes the chapter summarizing the main contributions.

6.2 Description of the Framework

In this section, we describe DEFIDNET, a framework for risk-rating and optimal allocation of countermeasures in IDNs. The framework uses the system model presented in Chapter 5, which allows for the abstraction of nodes composing the IDN. The framework is divided into three modules, depicted in Figure 6.1: the threat module, the risk-rating module, and the allocation module. We next describe each module in detail.

6.2.1 Threat Module

This module is used to define the threats to which IDN nodes are exposed and the propagation of the intrusive actions throughout the network. It receives an specification of the IDN based on the system model introduced in Section 5.2. For

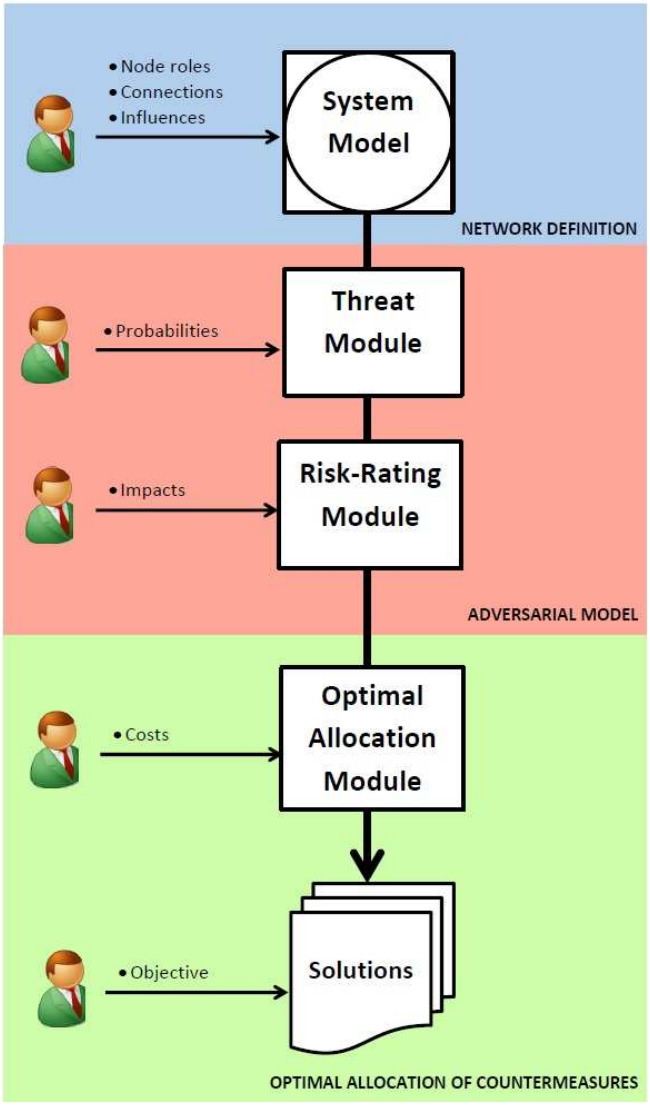


Figure 6.1: Modules of the framework DEFIDNET.
It uses as input a network abstraction and applies the different modules to the network. The output is a set of optimal solutions (the choice of a specific allocation depends on the particular objective)

.

every IDN node, this specification defines four channels: input of local events (LE), input of IDMsg (IIDM), output of IDMsg (OIDM), and output of response actions (RA). Each channel is exposed to four different intrusive actions: blocking (B), modification (M), interception (I), and fabrication (F).

We consider an adversary who targets a certain number of nodes at a time, performing one or more intrusive actions. The threat module is applied in to steps. First, it receives the probabilities of each intrusive action being performed for every

channel and every IDN node. These probabilities are manually entered. Then, the threat-module automatically propagates these initial probabilities throughout the network. We next explain these two steps of the module.

6.2.1.1 Attack Probabilities in Node Channels

The threat module receives as input the system model and the probabilities of intrusive actions for each channel of every node. If the nodes uses the same physical channel for input and/or output of data, then the probabilities would be the same. For example, if a computer uses the same network interface to monitor local traffic and to receive messages from other computers, then blocking the data in that network interface may block the LE and IIDM channels of the node. Additionally, in many adversarial scenarios, the probabilities of the intrusive actions are similar. For example, if the adversary is able to perform modification, it is probable that she may perform fabrication too. In any case, we do not consider in our framework how these probabilities are established, because it may strongly depend on the adversarial model and the network specifications. They are just defined manually, as shown in Figure 6.1.

This step of the threat module provides information about which nodes have been targeted initially. Thus, the threat module indicates where countermeasures should be allocated to protect nodes against the intrusive actions. However, it does not specify where it is better to place these countermeasures if the resources are limited, because it depends on the damage of each action and the cost of implementing the associated countermeasures. Moreover, as explained in the next step, the probabilities of intrusive actions are propagated throughout the nodes, and thus the damage may vary depending on the architecture of the IDN. These issues are dealt with by the the allocation and the risk-rating modules, respectively, which are described later in this chapter.

6.2.1.2 Propagation of Probabilities Throughout the IDN

Nodes in an IDN share information using IDMsg. Accordingly, the intrusive actions that affect one node are propagated throughout the connected nodes. The degree of propagation depends on the influences of the nodes connected. Consequently, a

single intrusive action in a channel of a node may not increase considerably the risk of the network. However, due to propagation, an adversary who wisely select its victims or combines several actions can provoke a serious damage in the network.

For example, suppose that a node A with the role DC is connected to a node B with the role PC. A provides data to B, who correlates this data and emits responses, if needed. Suppose that the adversary modifies the data in the LE channel of A. This intrusive action affects the OIDM channel of A as well, because the LE and the OIDM channels are internally connected through the event sharing function (ESF). Moreover, the OIDM channel of A is connected to the IIDM channel of B, and thus the modification would affect B as well. The degree of damage caused in B depends on the influence of A in B. The lower the influence, the lower the damage propagated from A to B.

The initial probabilities are manually entered for each node, and define the nodes that are vulnerable to external attacks. Then, the probabilities are propagated throughout the network as follows. First, the nodes propagate the probabilities internally. Depending on the role and the functions enabled in each node, the IIDM and LE channels may be connected to the RA and/or OIDM. Thus, the probabilities of every action affecting the input channels are propagated to the output channels in each node. Second, the probabilities of the OIDM channels are propagated to the IIDM channels of the connected nodes, weighted by the influence of the connection. Equation 6.1 indicates the propagation between connected nodes. In this expression, C_i represents the set of nodes connected to A_i, and I_{ij} is the influence from the node A_j to node A_i, with $j \neq i$. If one node previously had a probability in the IIDM channel (P_{IIDM}) greater than zero, then the new probability after propagation is the maximum between the previous probability and the sum of the probabilities propagated from connected nodes.

$$P_{IIDM}(A_i) = \max\{P_{IIDM}(A_i), \sum_{\forall A_j \in C_i} P_{OIDM}(A_j) * I_{ij}\} \tag{6.1}$$

Figure 6.2 shows an example of the propagation effect in a network with three nodes. The boxes in each node correspond with the probabilities of the four intrusive actions (B, M, I, and F) in each channel of the node. The nodes with role DC (*Slave1* and *Slave2*) are connected to a node with role PC (*Global*). In the initial state, only

Slave1 and *Slave2* are targeted, but after the propagation effect, the probabilities of the node *Global* are updated following Equation 6.1, and thus it becomes at risk.

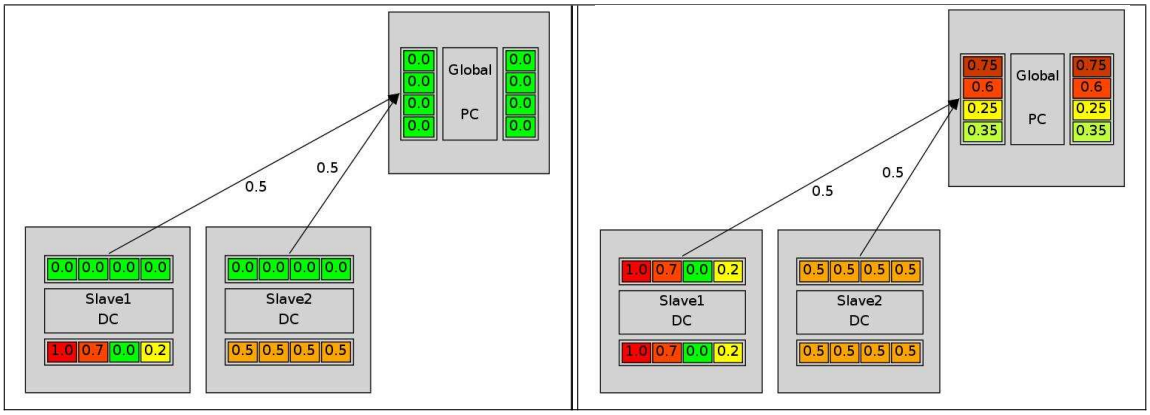


Figure 6.2: Effect of propagation of the probabilities of attack in an IDN with three nodes. The channels are represented as in Figure 5.1, i.e., the LE is on the bottom, the IIDM is on the left side, the OIDM is on top, and the RA is on the right side. Each box corresponds to the probabilities of the four intrusive actions (B, M, I, and F) in each channel of the node. The influence is represented a weight associated to each with arrow.

6.2.2 Risk-Rating Module

Risk in ICT systems is calculated as the product of the damage caused by attacks to the system times the likelihood (i.e., probability) of these attacks happening [Stoneburner et al., 2002]. As explained in the previous section, the threat module outputs the probability of each intrusive action in each channel of every node. The risk-rating module first receives the impact of the attacks, and then calculates the risk using the probabilities of these attacks happening. Next, we explain how the impacts and likelihood are defined, and how the risk is calculated from these metrics.

6.2.2.1 Attack Impacts

The adversarial model of IDNs is rather different to other ICT networks, where the main objective of attackers is to compromise the information confidentiality, integrity, or availability of data. In IDNs, though, attacks may have different goals. We adopt the taxonomy proposed by Corona et al. in [Corona et al., 2013], which

classifies attacks to Intrusion Detection Systems. This taxonomy is further explained in Chapter 2.

The impacts of attacks in each node depends on its consequences and it may be different depending on the adversarial model, the architectures, etc. For example, a DoS attack on a node with role DC may have higher impact in a distributed network than in a centralized network. Thus, in DEFIDNET the impacts are defined for each node and each attack (i.e., evasion, overstimulation, DoS, poisoning, reverse engineering, and response hijacking), by manually entering a damage value.

6.2.2.2 Attack Likelihoods

When targeting IDNs, adversaries may use different attack strategies. To assess the risk, each possible attack strategy should be considered. For example, a DoS could be performed by blocking the IDMsgs sent to the node, or by flooding the node with local events. Table 6.1 summarizes the intrusive actions required depending on the attack goal. In Chapter 5, we provide details of each specific strategy and examples of how they are actually implemented in real scenarios. We next provide three examples of attacks using these strategies.

1. *Evasion with modification in LE.* An evasion occur if the adversary modifies the data to blend with statistical properties of a normal model [Fogla and Lee, 2006]. This implies the adversary acting on the LE channel of the attacked node.

2. *DoS with fabrication in LE.* Some approaches use internal data structures to track the monitored data, like observed anomalous behavior of nodes in MANETs [Su, 2011]. A DoS occurs if the adversary is able to overload these structures by fabricating specific packets, which implies acting in the LE channel.

3. *Reverse engineering with fabrication in LE and interception in OIDM.* By performing query-response analysis, the adversary can infer information used internally by the nodes [Pastrana et al., 2014]. Moreover, if the goal of the adversary is to discover the roles of nodes in an IDN, it can perform a traffic analysis attack. For example, by injecting intrusive packets in the IDN (LE

channels of nodes) and observing who is responding (OIDM channel) and the destination of the IDMsg, the adversary can determine who is gathering data to perform correlation.

Accordingly, using the probabilities of intrusive actions to node channels established in the threat module, and the attack strategies showed in Table 6.1, the likelihood of each attack is calculated as the maximum probability of actions that conduct to it. For example, consider the *Global* node in Figure 6.2 after the propagation. The probabilities of intrusive actions in the IIDM channel (left side of the node) are 0.75 for blocking, 0.6 for modification, 0.25 for interception, and 0.35 for fabrication. As shown in Table 6.1, evasion in the node *Global* can be done by either blocking, modifying or fabricating in the channel IIDM. The most probable action is blocking (0.75).

Table 6.1: Taxonomy of attacks showing which intrusive actions may use the adversary on each channel to achieve different goals.

Adversarial attack	intrusive actions on channels			
	LE	IIDM	OIDM	RA
Evasion	B ∨ M ∨ F	B ∨ M ∨ F	–	–
Overstimulation	F	F	M ∨ F	–
Poisoning	M ∨ F	M ∨ F	–	–
Denial of Service	B ∨ F	B ∨ F	B	–
Reverse Engineering	(M ∨ F) ∧ I	(M ∨ F) ∧ I	I	I
Response Hijacking	B ∨ M ∨ F	B ∨ M ∨ F	–	B ∨ M ∨ F

6.2.2.3 Calculation of the Risk

Once the impact of attacks are entered, and the likelihood of these attacks happening is calculated, the risk-rating module calculates the risk of one attack as the product of the likelihood of this attack multiplied by its impact on the node. Assuming that the impact of evasion in the *Global* node from Figure 6.2 is 100 and the likelihood of evasion is 0.75, then the risk of the *Global* node being evaded would be $0.75 * 100 = 75$.

The risk-rating module outputs the total risk of the IDN, and for each node, the risks for each attack and its aggregated risk (sum of all the attack risks). The total risk of the IDN is the sum of the risks of all the individual nodes. This information together with the information about which nodes have been targeted (given by the threat module), is given to the allocation module described in the next section.

6.2.3 Allocation Module

DEFIDNET is a framework to optimally allocate countermeasures in an IDN. In this section, we consider the problem of reducing the estimated risk using the lowest possible quantity of resources. The allocation module first receives the cost of the countermeasures, and then calculates optimal allocation of these countermeasures to reduce the risk. We next explain the two steps involved in this module.

6.2.3.1 Cost of Countermeasures

We define the cost of a countermeasure as the quantity of resources required to protect a single channel for one node against a specific intrusive action. We consider this cost as a single value, and we do not consider neither what exactly it is (money, time, energy, etc.) nor how it is measured. For example, to protect against interception, it can be used cryptographic mechanisms to encrypt the communications. These mechanisms may require the use of secret keys or a PKI. Depending on the network and the scenario of application, this may be more or less costly. Moreover, the cost of protecting against interception is not the same in different nodes and channels. For example, encrypting the communication in a MANET is usually more costly than encrypting a wired link. Similarly to the probabilities, DEFIDNET uses as input the cost to protect each intrusive action on each channel of the nodes.

In the following, we consider a solution as a set of countermeasures to be applied to the IDN. On the one hand, when a countermeasure is applied to one channel to counter an intrusive action, the probability of this action happening in this channel becomes zero. However, since not all the channels are protected, after applying the countermeasures of a solution, some residual risk is left behind in the IDN. On the other hand, each countermeasure has an individual cost, and thus, applying a set of countermeasures has a total cost calculated as the sum of each individual cost.

6.2.3.2 Optimization of the Cost-Risk Tradeoff

For each solution, the more risk is mitigated, the higher the cost. Ideally, optimal solutions should minimize both the risk and the cost. However, these are mutually conflicting objectives, and there is not a single optimal solution. Thus, a tradeoff between risk and cost must be considered. Accordingly, we use Multi-Objective

Optimization (MOO) to obtain the set of optimal solutions that conform the pareto set. In MOO with two objectives, a solution from the pareto set is called non-dominated if there is not any other solution that improves one of the objectives without degrading the other objective. The set of non-dominated solutions is called the pareto front.

There are several algorithms to obtain the pareto front. In our experiments, we use an evolutionary MOO algorithm known as SPEA2 [Zitzler et al., 2001] (an optimization of the Strength Pareto Evolutionary Algorithm). SPEA2 is one of the most popular MOO evolutionary algorithms and has been successfully applied in the intrusion detection domain [Sen and Clark, 2011; Sen et al., 2010]. Indeed, it is one of the two MOO algorithms implemented in the ECJ framework [Luke, 2010], which we use in our experiments. The other algorithm implemented in ECJ is NSGA2 (Non-dominated Sorting Genetic Algorithm) [Deb et al., 2000]. While both of them are valid algorithms, SPEA2 obtains further optimization in the central points of the pareto front than NSGA2, which is more convenient to obtain solutions in the boundaries of the pareto front. In our particular domain, solutions that are very costly or that reduce very low risk are generally not recommended. Accordingly, the main purpose is to optimize the points where it is unclear the tradeoff between cost and risk, which are the central points of the pareto front. For these reasons, we choose SPEA2 in our experiments.

As explained in Chapter 2, evolutionary algorithms perform heuristic search to explore a solution space. The algorithm manages a population of "individuals", which in our case are the different solutions to allocate countermeasures expressed as a binary mask (1 means that the concrete countermeasure is applied, 0 that it is not applied). Figure 6.3 shows an schematic view of an individual. Each position of the mask indicates which actions (i.e, blocking, modification, interception, and/or fabrication), for each channel, and for every IDN node, are counteracted by the solution represented. The evolution applies various genetic operators to the individuals to obtain each new generation. We next explain such operators:

- **Crossover**. Given two individuals, (i.e., binary masks), the crossover breeds a new individual by mixing certain genes from the individuals (i.e., chunks from the masks). Concretely, in our experimentation, the crossover randomly divides the masks of both individuals in 5 chunks, which are swapped between

them uniformly.

- **Mutation**. Given one individual, the mutation operator flips a 10% of the bits from its mask. These bits are randomly chosen.

- **Selection**. This operator selects individuals from one generation to compose the next generation. We use the tournament selection, which creates several "tournaments" randomly choosing a number k of individuals from the population. Then, the best individual in each "tournament" is selected.

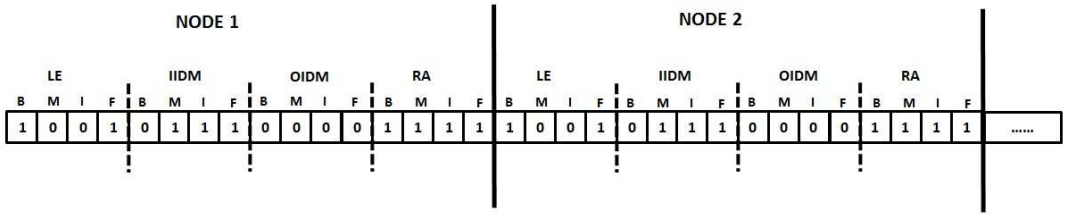


Figure 6.3: Representation of a SPEA2 solution to allocate countermeasure as a binary mask. Each position in the mask indicates whether a countermeasure to each intrusive action must be applied (1) or not (0) in the corresponding channel of every IDN node.

.

The population evolves during several generations, increasing the "fitness" of individuals. The fitness is a value associated with each individual, which indicates how good it is. Concretely, the individuals are evaluated as follows. First, the countermeasures indicated by the mask are applied (i.e., the corresponding probabilities are set to zero) and its corresponding cost is calculated. Then, the remaining probabilities of intrusive actions in the IDN nodes (i.e., these whose action have not been counteracted) are propagated, and the risk is re-calculated. Finally, the pair "cost-risk" is given to the algorithm to calculate the fitness. In the case of SPEA2, the fitness of each solution considers the number of points that are dominated by the pareto points in the solution.

When it is required to reduce the risk completely or when there are unlimited resources, then all the nodes are protected completely (i.e, all the risk is mitigated). However, when the cost is limited or the IDN tolerates some risk, the pareto front indicates which are the optimal solutions. These solutions indicate which are the countermeasures to be applied in order to solve one of the two following problems:

1. Given a tolerable risk, the problem is "selecting the cheapest set of countermeasures that mitigates the risk below a tolerable level of risk".

2. Given an available budget, the problem is "selecting the set of countermeasures that reduce the risk the most while spending less resources than the given budget".

If the budget is limited, the allocation solution must reduce the risk the most. If there is a tolerable risk, the allocation solution must be the cheapest that decreases the risk below the tolerated level. In some situations, though, neither the cost nor the risk are limited. In these cases, it is helpful to know whether it is worth to spend more resources to reduce the risk or not. When defending an IDN, one may think that the more resources are spent, the more risk is mitigated. However, this is not always the case. In the following section, we show experimental results that confirm this intuition.

6.3 Experimental Work and Discussion

As it has been discussed in the previous section, the main advantage of DEFIDNET is that it allows to model IDNs through the definition of architectural parameters (size of the IDN, role of the nodes, connections, and influences) and per node parameters (probabilities of intrusive actions, impact of attacks, and cost of countermeasures).

In this section, we provide analytic results using DEFIDNET. Concretely, we model two IDNs with different architectures: a hierarchical network and a centralized network. First, we explain how the networks are modeled and the parameters established. Second, we analyze and discuss the tradeoff between cost and risk on each IDN, which helps to decide the placement of countermeasures and whether it is worth or not to spend resources, depending on the mitigated risk. Finally, we discuss the use of DEFIDNET in the modeling of IDNs and how it provides optimal alternatives to allocate the countermeasures depending on the resources available or the tolerable risk.

6.3.1 Network Modeling

We have conducted experiments using two IDNs, each of them with 101 nodes, but distributed with different architectures: a centralized network and a hierarchical network. We next describe the use of DEFIDNET applied to each of these networks.

6.3.2 IDN Definition

For each IDN, we first define the system model, which is composed of the number of nodes, the role of each node, and the connections between them. Depending on the role of each node, some of their functions and channels are activated, as explained in Chapter 5.

In a centralized network, a central node correlates the information received from several nodes and generates responses. We use a network with 100 nodes that collect data and send it to a central correlation node. In a hierarchical IDN, the network is divided into levels. We have established 3 levels in the network. In the top level there is one global node (with role PC). In the middle level, there are 10 nodes (with role RC) connected to the global node. Finally, in the bottom level, 10 nodes nodes are connected to each of RC nodes in the middle level. The nodes in the bottom level have the role DC.

The influences between nodes determine how the intrusive actions that affect one node are propagated to the connected nodes. In our experimental work, we establish different values for these influences, according to different data distributions. In the following sections, we retake this and detail how the influences are established.

6.3.3 Adversarial Model

Once the system is modeled, we establish the attack probabilities on each node and their impacts. We consider a powerful adversary who targets all the DC nodes that collect data locally, with a probability equal to 1. These DC nodes only send data to a correlation node and do not emit responses. Thus, the impact of the attacks in DC nodes is low, as opposite to the impact in the correlation nodes, which is one hundred times greater. Since the impact is greater in the correlation nodes, and

because only the DC nodes are targeted, the risk of the entire IDN depends on how the risk propagation within the IDN affects other systems.

6.3.4 Allocation Module

The allocation module requires as input the cost of the countermeasures and the output of the threat module, i.e., the definition of which nodes are at risk and which countermeasures should be taken. Optimal allocation of countermeasures depends on two factors, the residual risk of the network after the countermeasures are implemented, and the cost of these countermeasures. On the one hand, the risk of the network is affected by the propagation of attacks, which in turn depends on the influences between nodes. On the other hand, the cost of the countermeasures varies from one node to another.

We conduct experiments using different values for the influences and costs of the countermeasures. Concretely, these values are established following four different data distributions (in the following, we refer to the value of influence or cost given to each node as *quantity*):

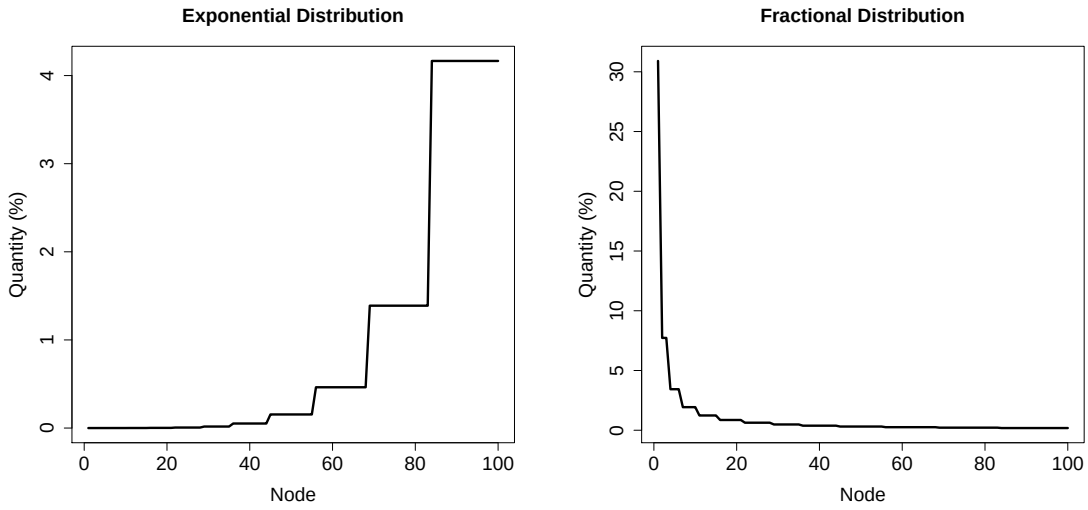
1. *Exponential (E).* This distribution is shown in Figure 6.4-(a). With this distribution, the lower the quantity is, the fewer the number of nodes receive this quantity. Thus, the highest quantity is given to many nodes.

2. *Fractional (F).* As shown in Figure 6.4-(b), with this distribution the highest value is given to a single node, and then the lower the quantity is, the more the number of nodes that receive it.

3. *Gaussian (G).* The assignment follows a normal distribution. The majority of the nodes receive a medium quantity, while some few nodes receive low quantities and some few nodes receive high quantities. Figure 6.4-(c) shows this distribution.

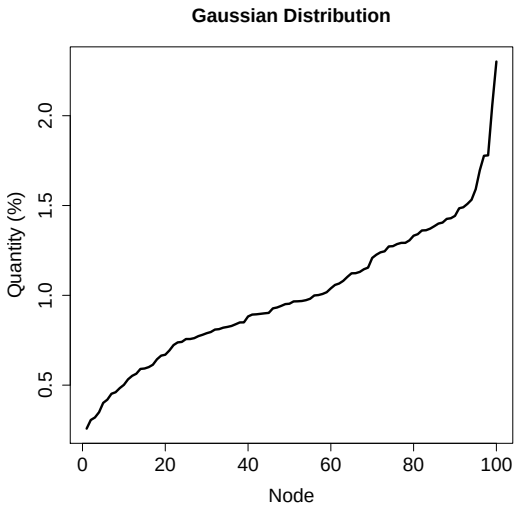
4. *Uniform (U).* All the nodes receives the same quantity.

In all the experiments, the total cost of the network distributed among the nodes is 100. This means that, if we consider a network with 100 nodes and an uniform distribution, the cost to defend each node is one.



(a)

(b)



(c)

Figure 6.4: Data distributions used. The plots show the percentage of cost or influence (quantity) assigned to each node in a network of 100 nodes.

For each distribution of influences and each distribution of cost, we use DEFIDNET to analyze the tradeoff between the cost of the countermeasures and the mitigated risk. Thus, for each IDN we conduct sixteen experiments.

6.3.5 Analysis of the Obtained Results

We have run 16 experiments, using four distributions of influence, and for each of them, four distributions of cost. Each experiment can be viewed as an alternative to distribute the influences and costs over the same IDN. The analysis provides insights on which of these alternatives is optimal in different situations.

We provide the results both as a numerical analysis, showing the actual percentages of risk mitigated and associated cost by the solutions, and a graphical analysis, plotting the pareto fronts and analyzing their trend lines.

6.3.5.1 Numerical Analysis

Table 6.2 shows the quartiles of the amount of risk mitigated. Each quartile indicates the percentage of the cost required to mitigate the 25% (Q1), 50% (Q2), 75% (Q3), and 100% (Q4) of the risk. The total cost is calculated by applying all the countermeasures in the network, i.e., spending all the resources.

The symbols F, E, G and U corresponds to fractional, exponential, gaussian and uniform distributions respectively. We have highlighted the lowest values for each quartile, which indicates which are the cheapest alternatives. These values indicate the situations where some distributions of cost and influence are better than other. For example, in the centralized network, to reduce the risk 25%, the best alternative would be the fractional distribution of influences and exponential distribution of costs (referred as IF-CE). This alternative only requires 14.6% of the cost. The alternative with exponential influence and gausssian cost (referred as IE-CG) would require more than the double of the cost (31.6%) to mitigate the same risk. However, the table shows that to reduce 75% of the risk, the best alternative is precisely this IE-CG, which barely requires 69% of the cost.

The comparison between the alternatives IF-CE and IE-CG shows that the election of one alternative or another depends on the goal. On the one hand, for the centralized network, the IF-CE alternative must be selected if, 1) its required to completely mitigate the risk, or 2) reducing 25% of the risk is enough. On the other hand, if the goal is to mitigate the risk substantially (75%), the best alternative to use is the IE-CG.

A similar analysis can be done with the hierarchical network. In this case, the IU-CF alternative reduces 25% of the risk spending only 9.8% of the cost, while the IG-CG alternative requires three times more cost (30.5%). However, the IG-CG alternative is the best choice to mitigate the risk completely. A general observation is that the percentage of risk to be mitigated determines which alternative is better to save resources.

Table 6.2: Percentage of the cost required to mitigate 25% (Q1), 50% (Q2), 75% (Q3), and 100% (Q4) of the risk in centralized and hierarchical IDNs.

Distributions		Centralized architecture				Hierarchical architecture			
Influence	Cost	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
F	F	20.7	54.8	88.0	96.7	21.7	44.9	83.5	96.2
	E	**14.6**	39.2	71.0	**89.4**	20.8	42.3	76.7	94.6
	G	31.1	43.7	70.6	92.4	28.4	45.0	73.0	93.9
	U	32.1	43.2	69.2	91.5	25.1	46.0	73.8	94.8
E	F	18.7	**29.7**	81.8	94.0	17.6	40.7	77.7	96.4
	E	26.9	54.6	87.3	95.1	19.0	49.3	85.3	94.0
	G	31.6	45.8	**69.0**	91.4	34.9	50.2	**68.3**	93.5
	U	29.5	48.7	71.1	92.2	21.8	48.1	77.6	95.3
G	F	14.7	39.5	80.9	94.9	10.8	**37.8**	85.1	94.7
	E	20.6	45.3	80.0	92.8	13.1	43.8	81.5	91.6
	G	29.2	52.3	75.9	93.2	30.5	51.9	72.0	**91.0**
	U	27.6	53.0	73.6	92.8	17.5	49.5	80.9	94.3
U	F	18.8	38.4	86.5	95.0	**9.8**	42.4	87.9	95.0
	E	15.7	50.3	83.8	95.7	14.8	46.3	85.3	92.3
	G	27.1	52.9	74.5	90.5	25.4	53.4	72.4	91.7
	U	25.5	52.7	75.3	94.1	22.1	51.6	78.4	92.5

6.3.5.2 Graphical Analysis

Figures 6.5 and 6.6 shows the solutions of the pareto fronts obtained for the centralized and hierarchical networks, respectively. For the sake of illustration, we have grouped the pareto fronts in 4 plots, each one corresponding to an influence distribution. Each plot shows the percentage of residual risk (y-axis) and the percentage of cost (x-axis) associated with the solutions. A 100% of risk means that no countermeasures are applied (i.e., 0% of the cost spent), while a 100% of the cost means that all the required countermeasures to protect the network are applied.

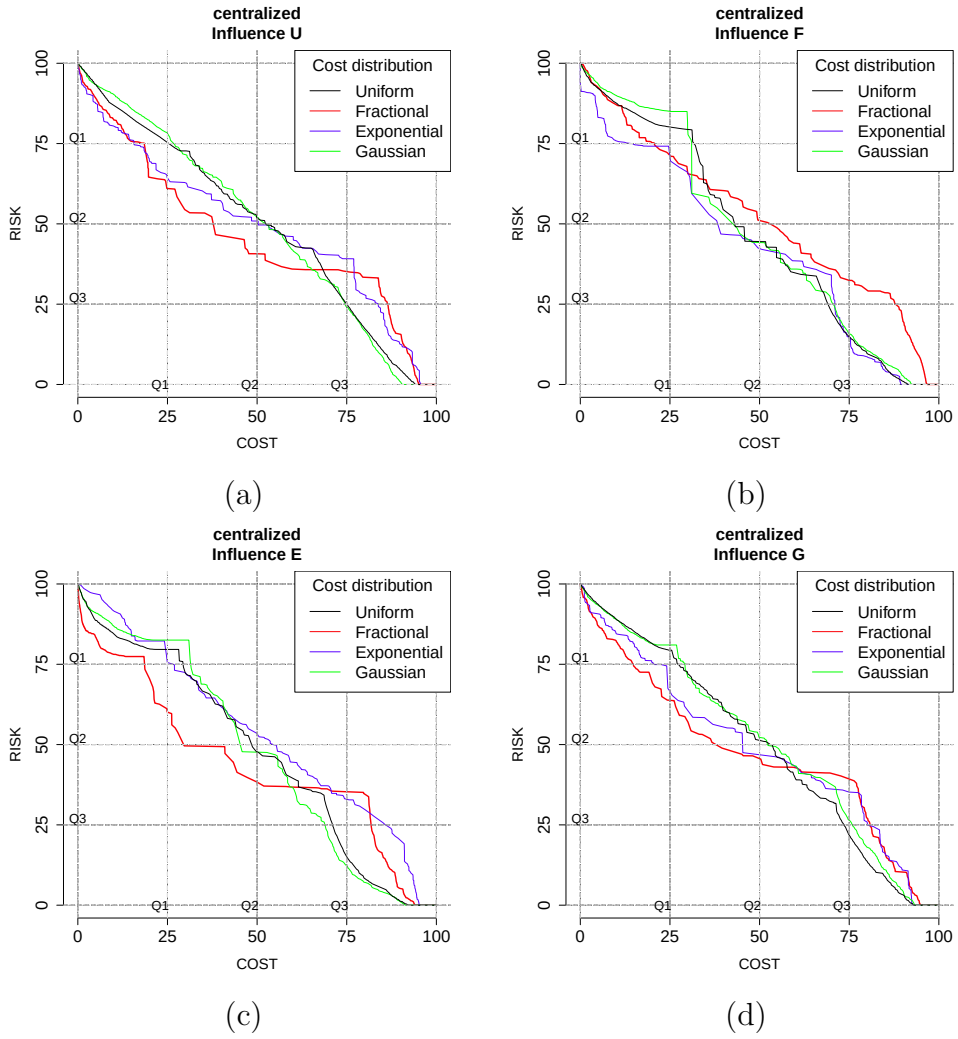


Figure 6.5: Cost-risk tradeoff in the centralized IDN.
Each plot shows the pareto front obtained for each cost distribution and one influence distribution.

Assume that, in a given time, some countermeasures have been applied in the IDN, and thus the risk has been reduced partially. At this point, is it better to spend more resources to further reduce the risk, or is it better to do nothing, thus saving resources? The analysis of the trend line of the obtained pareto front helps to answer these questions. We next provide two examples:

- *Example 1.* In the centralized IDN, consider the pareto front of the IE-CF alternative depicted in the red lines in Figure 6.5-(c). The trend line indicates that spending half of the cost (Q2) reduces almost the same risk than spending 75% of the cost (Q3). Thus, supposing that the IDN risk has been reduced up

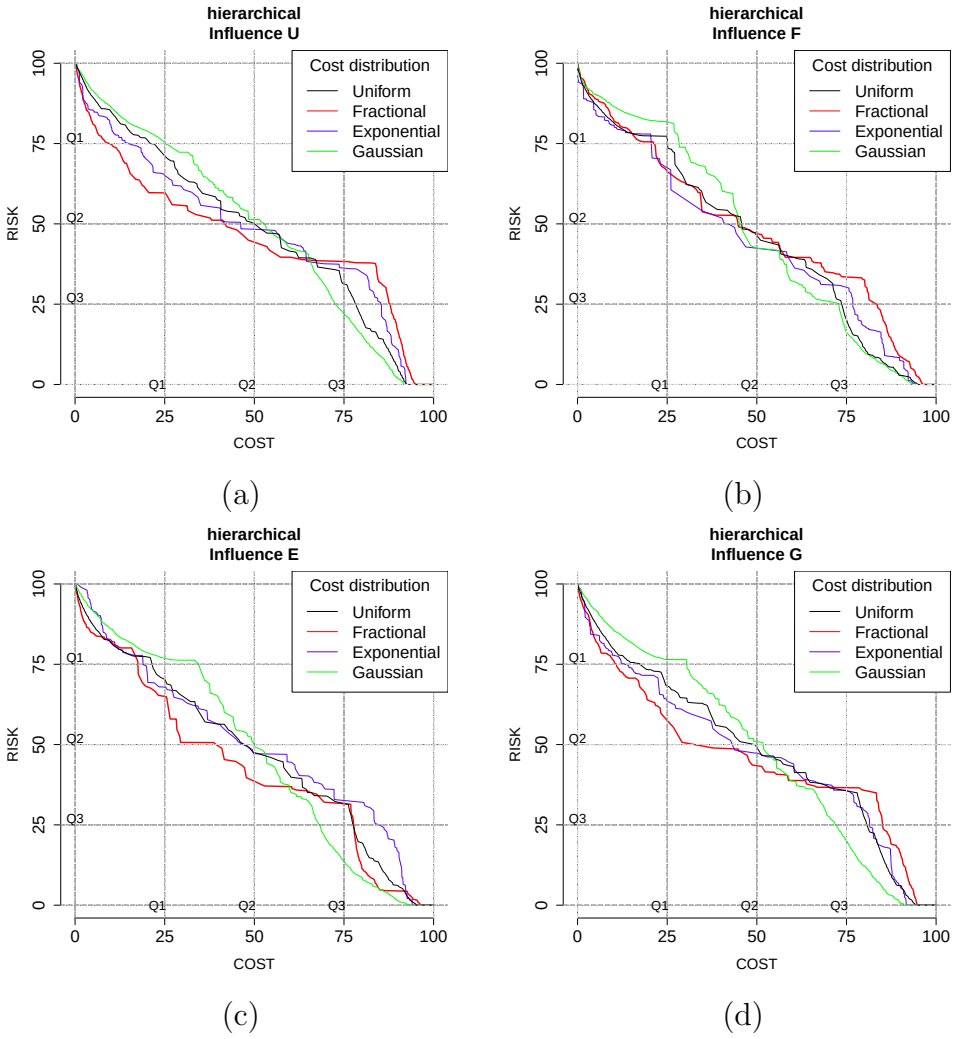


Figure 6.6: Cost-risk tradeoff in the hierarchical architecture.
Each plot shows the pareto front obtained for each cost distribution and one influence distribution.

to 30% (using half of the budget), the next reduction of risk would require to spend 30% more cost. In this case, if there are no restrictions about the risk, an intelligent decision would be to save resources and do not implement more countermeasures.

- *Example 2.* In the hierarchical IDN, consider the IF-CG alternative depicted in green line of Figure 6.6-(b). In this case, with 28% of the cost, the risk is only reduced to a 80%. However, analyzing the trend line, it can be observed that with only some more cost (around 12% more) the risk is drastically reduced to less than 50%. Thus, contrarily to *Example 1* above, it is worth spending

some more resources because it reduces a significant amount of risk.

Similarly to the numerical analysis, the trend line of the pareto fronts can be used to compare between different alternatives. In this case, the decision of what alternative to choose is done by analyzing the lines: the line "arriving" first to the desired quartile is the best alternative. In the centralized network, for example, with fractional influence (IF), the blue line (CE) is the first to reach the first quartile (Q1) of the risk. Accordingly, the IF-CE alternative is the best choice to reduce a 25% of the risk. This result matches with the numerical analysis done in the previous section.

6.3.6 Discussion

In this section we have discussed how DEFIDNET helps to decide when it is better to spend more resources to reduce the risk, and where. In many IDNs, establishing countermeasures in a indiscriminate manner may reduce the risk proportionally to the cost, i.e., spending more resources involves reducing more the risk. Graphically, the trend line of the pareto front of this approach would have a constant decrement, like the black line of Figures 6.5-(a) and 6.6-(a). However, the analysis performed in this section shows that in some IDNs, the previous effect is not true, and increasing the cost does not always translates into a risk reduction. In this case, the trend line of the pareto front would have substantial leaps, like the lines analyzed in this section.

In order to save resources, it is useful to know when it is convenient to allocate new countermeasures, and where should they be placed. The decision depends on several parameters, like the architecture of the network, the influences between nodes, the cost of setting countermeasures in the nodes, etc. In the presented experiments, we have analyzed different alternatives for rather small and simple IDNs. We defined different alternatives by randomly assigning influences and costs using different distributions. However, when dealing with bigger networks and having non-trivial alternatives (i.e., which are not random), the value of DEFIDNET is even greater. In the next section, we provide experimental results using a larger and more complex IDN than the ones used here.

6.4 Case Study

In this section, we use DEFIDNET to analyze a Cooperative Intrusion Detection Network (CIDN). In a CIDN, several entities (companies, organizations, governments, etc.) share the information that they have obtained from their own corporate network. Each of these corporate subnetworks are proprietary of one entity, and thus they may have different architectures, sizes, and also the adversary model varies for each of them.

In this section, we first define the IDN and all its parameters. Second, we show the tradeoff between cost and risk of different solutions in terms of the pareto front. Finally, we select one specific solution according to a budget and discuss the countermeasures that this solution suggests.

6.4.1 Architecture Design and Adversarial Model

Figure 6.7 shows the IDN used for the case study, once the probabilities are entered and propagated. The network simulates a scenario where five entities share information of intrusion attempts in their corporate subnetworks. Each corporate subnetwork has a global node in charge of the communication with global nodes from other corporations. The global nodes correlate data received from their corporate subnetwork and from the other global nodes. If needed, global nodes emit responses. Because the corporations are independent and they are vested with the same authority, the global nodes are interconnected in a distributed fashion, and each of them is connected with all the remaining global nodes. Next we describe each corporate subnetwork:

1. **Subnetwork centralized-100**. Within this network, 100 nodes with role DC gather data locally and send it to the global node. The influence of these nodes to the global one is established with a fractional distribution, and the costs of each DC node is established with an exponential distribution. This subnetwork is highly targeted: 70% of the DC nodes have full probability of being attacked (i.e., the probability for each intrusive action is set to 1).

2. **Subnetwork centralized-20**. This network has a centralized architecture but, unlike the previous network, it has only 20 nodes collecting data. Moreover,

these nodes do not communicate directly with the global node of the entity, but with a proxy who actually communicates with this global node. The distribution of influences and costs of the subnetwork are uniform, and only 10% of the nodes are targeted.

3. **Subnetwork hierarchical**. The global node of this entity is the root of a hierarchical subtnetwork. The hierarchy has two middle level nodes (with role RC), each of them correlating data gathered from five DC nodes. This network is completely targeted, and thus all the nodes have their probability of being attacked equal to one. The distribution of the influences is uniform, and the distribution of the costs is fractional.

4. **Subnetwork ring**. This subnetwork uses a special architecture with a ring of hierarchies. Four nodes are interconnected within a ring architecture (i.e., a node is connected to just another node and receives data from a different one). At the same time, each of these nodes correlates data received from three nodes who gather data locally. One of the nodes in the ring, acting as proxy, is connected to the global node of this entity. Both the influences and costs are uniformly distributed, and only three nodes in the lower level of the hierarchy are targeted.

5. **Subnetwork distributed**. The detection within this network is distributed, and all the nodes have the same responsibility. Each node has the role LDA and is connected to 10% of the remaining nodes of the subnetwork. One of the nodes is the proxy connected to the global node. The distributions of the costs and influences are uniform. Only a small subset of these nodes are targeted (5%). However, it can be observed in Figure 6.7 that, due to the propagation of the probabilities, almost every node within this network is put at risk. The impact of compromising each of these nodes, though, is one-hundred times lower than the impact of compromising the global node of this entity (following we explain the impacts assigned to the nodes of the network).

Each global node, besides cooperating with the remainder global nodes, generates responses.[1] Accordingly, the impact of attacks on these nodes is higher than in the remaining nodes of the network. The impacts established for each node role are:

[1]Note that we do not consider in our study the specific responses emitted by nodes

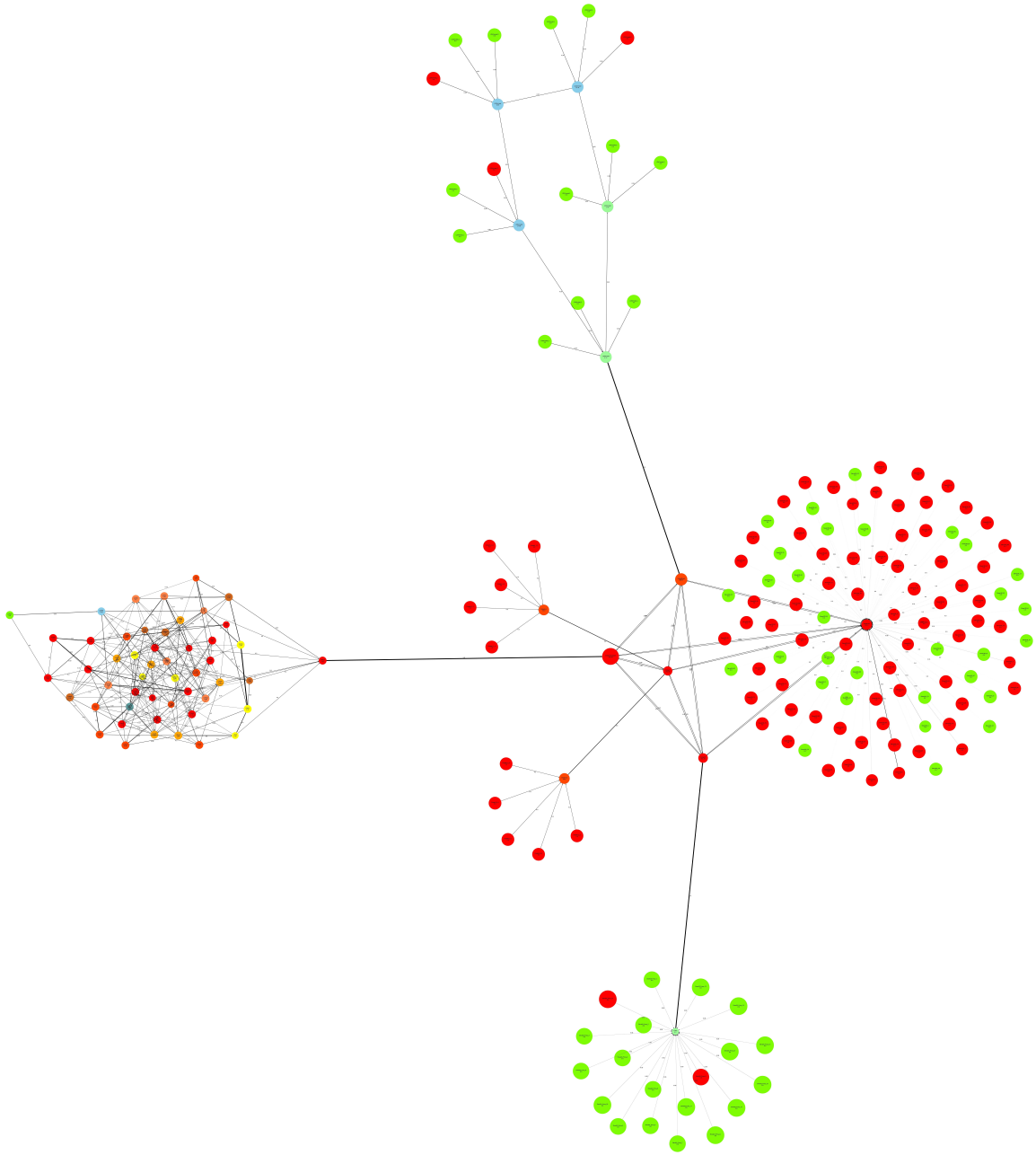


Figure 6.7: CIDN for the case study in its initial state.

- Global nodes: 100
- Proxy nodes (those that are connected to a global node): 10
- Data Collector and peers in the partially-distributed: 1

A priori, it is not easy to determine where it is better to set countermeasures in this network. Should every node in the centralized-100 subnetwork being protected first?, or is it better to protect the hierarchical subnetwork because it is smaller? We next use DEFIDNET to analyze the cost-risk tradeoff by plotting the pareto front obtained. Then, we analyze specific solutions suggested by DEFIDNET.

6.4.2 Cost-Risk Tradeoff

Figure 6.8 shows the pareto front of the solutions for the cooperative network. As explained above, each point in the pareto front correspond to an optimal solution for each risk level with the least cost. In the figure, we have highlighted certain points which we analyze further in the next section. The trend line of the pareto indicates that, spending few resources in the beginning, the risk rapidly decreases, and spending approximately 10% of the cost, the risk decreases to 75%. In that point, the gradient of the line becomes zero for an appreciable interval. That means that there is no improvement in the mitigated risk spending 15% and 22% of the cost. In the next subsection we analyze these two solutions and analyze why this situation occurs. Suddenly, the trend line substantially falls below 50% (between the orange triangle and the blue circle points), which means that, a solution that spends only 7% more of the cost than the previous one, reduces the risk 24%. Again, in the next section we analyze why this improvement occurs.

Following the analysis of the trend line, we observe that once the risk has been reduced below 50%, the trend line relatively enters in a decreasing interval for a while. However, it can be observed that it is required a greater percentage of the cost than the corresponding risk reduction. Finally, just before the line crosses the third quartile of the risk (i.e., just before reducing the risk below 25%), the trend line suffers a significant decrease, and the network becomes practically secured by spending 80% of the cost.

The analysis of this specific line shows that, if the budget for defending the network is low, then a good option is to reduce the risk at least 50%, because it can be achieved with little more than 25% of the budget. However, if the budget is high, it is better to spend the 80% because in such a case the risk is practically reduced to zero.

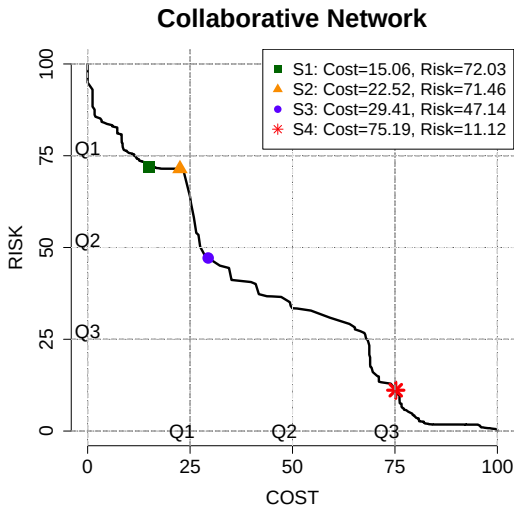


Figure 6.8: Pareto front showing the cost-risk tradeoff of the CIDN studied. The highlighted points are the solutions analyzed in Section 6.4.3.

6.4.3 Analysis of Specific Solutions

As explained above, each point in the pareto front correspond with one solution in the network, i.e., the set of countermeasures to be applied to optimally reduce some risk using some cost. In this section, we analyze four solutions suggested by DEFIDNET which are highlighted in Figure 6.8.

The two solutions S1 and S2 mitigate little risk and spend few resources. The difference between them is that, while S2 mitigates only 1% more than S1, it costs 7% more. However, analyzing the line in the point S2, it can be seen that spending again another 7% of the cost provokes that the risk is reduced 25% (as seen in the solution S3 highlighted in Figure 6.8). What exactly is the difference between S1, S2 and S3?

In Table 6.3 we show the number of countermeasures that solutions S1, S2, and S3 allocate in each of the corporate subnetworks. The biggest different between solutions S1 and S2 is in the subnetwork centralized-100 and the subnetwork distributed. This difference suggests that setting countermeasures in these subnetworks does not reduce substantially the risk, unless all the risk in these subnetworks is mitigated. In the case of the centralized-100 subnetwork, the problem is that all the DC nodes are directly connected to the global node. In the case of the distributed subnetwork, the problem is that a single targeted node will propagate the risk throughout the

entire network, and because one of the participants in this subnetwork is directly connected to the global node, the attack is propagated to the entire CIDN.

Regarding the difference between the solutions S2 and S3, it can be observed that in the hierarchical subnetwork, the defenses allocated by S3 are doubled, and even in the centralized-20 there is one less countermeasure. It can be observed that it spends more resources in protecting the hierarchical subnetwork and the overall risk is considerably reduced. The costs in the hierarchical network follow a fractional distribution, which means that some of the nodes has a high cost while the remainder has low cost. This means that, if there is enough budget to protect the highest cost nodes, then it is worth doing so, because the remaining nodes are cheap and then it becomes easier to protect the entire subnetwork.

Figure 6.9 shows the countermeasures allocated per action type. As it can be observed, in the solution S4, countermeasures to interception are almost the same than in solutions S3 and S2. This suggests that avoiding interception is not optimal unless there are enough resources. Actually, as shown in Table 6.1, the interception is only valuable for an adversary to perform a reverse engineering attack. Thus, protecting against interception only reduces the risk for reverse engineering. Another conclusion that can be extracted from Figure 6.9 is the importance of avoiding fabrication actions. Indeed, between S1 and S2, the countermeasures to protect against fabrication are the same, while there are considerable differences in the blocking, modification, and interception. As seen in Figure 6.8, solutions S1 and S2 mitigate almost the same risk, S2 being a 7% more expensive than S1. Solution S3, though, by spending of few more resources than S2 to protect nodes against fabrication, decreases the risk considerably.

Table 6.3: Analysis of the solutions highlighted in Figure 6.8.

Solution	Number of countermeasures in each subnetwork				
	Centralized-100	Centralized-20	Hierarchical	Ring	Distributed
S1	96	3	5	3	7
S2	131	3	7	4	11
S3	139	2	14	6	14
S4	167	5	33	11	33

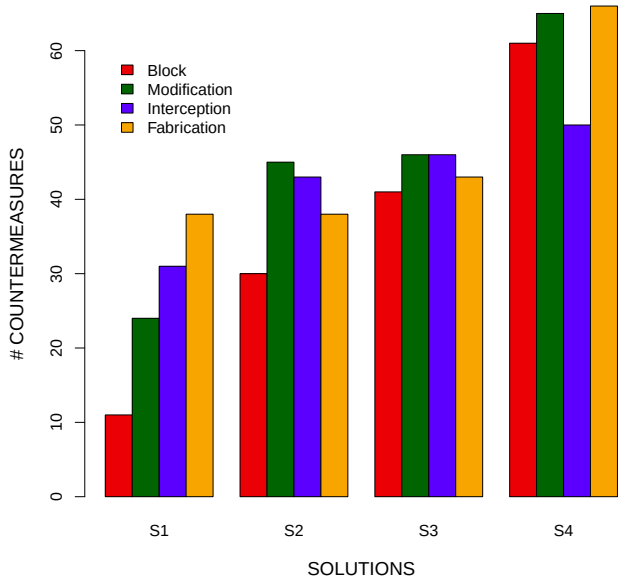


Figure 6.9: Number of countermeasures per action type.

Finally, Figure 6.10 shows the CIDN after the solution S4 is applied. It can be observed that, as it was suggested in the analysis of solutions S1, S2, and S3, when allocating countermeasures, the optimal approach is to leave the centralized-100 network at the end of the queue and protect the other subnetworks first.

6.5 Conclusions

Intrusion Detection Networks are used to detect complex, distributed attacks. They aggregate several nodes with different roles that are interconnected to share information. Accordingly, a compromised node may expose the entire IDN to a risk. Due to the adversarial scenarios in which these networks operate, the design of robust architectures is critical to maintain an acceptable level of security.

In this chapter, we have presented DEFIDNET, a framework that assesses the risk of IDNs against specific attacks in the nodes. Node abstraction allows the definition of single probabilities of intrusive actions in the channels of each node, which is simpler than defining the probability of complex attacks in the entire network. Then, considering these probabilities and their propagation throughout the network, the

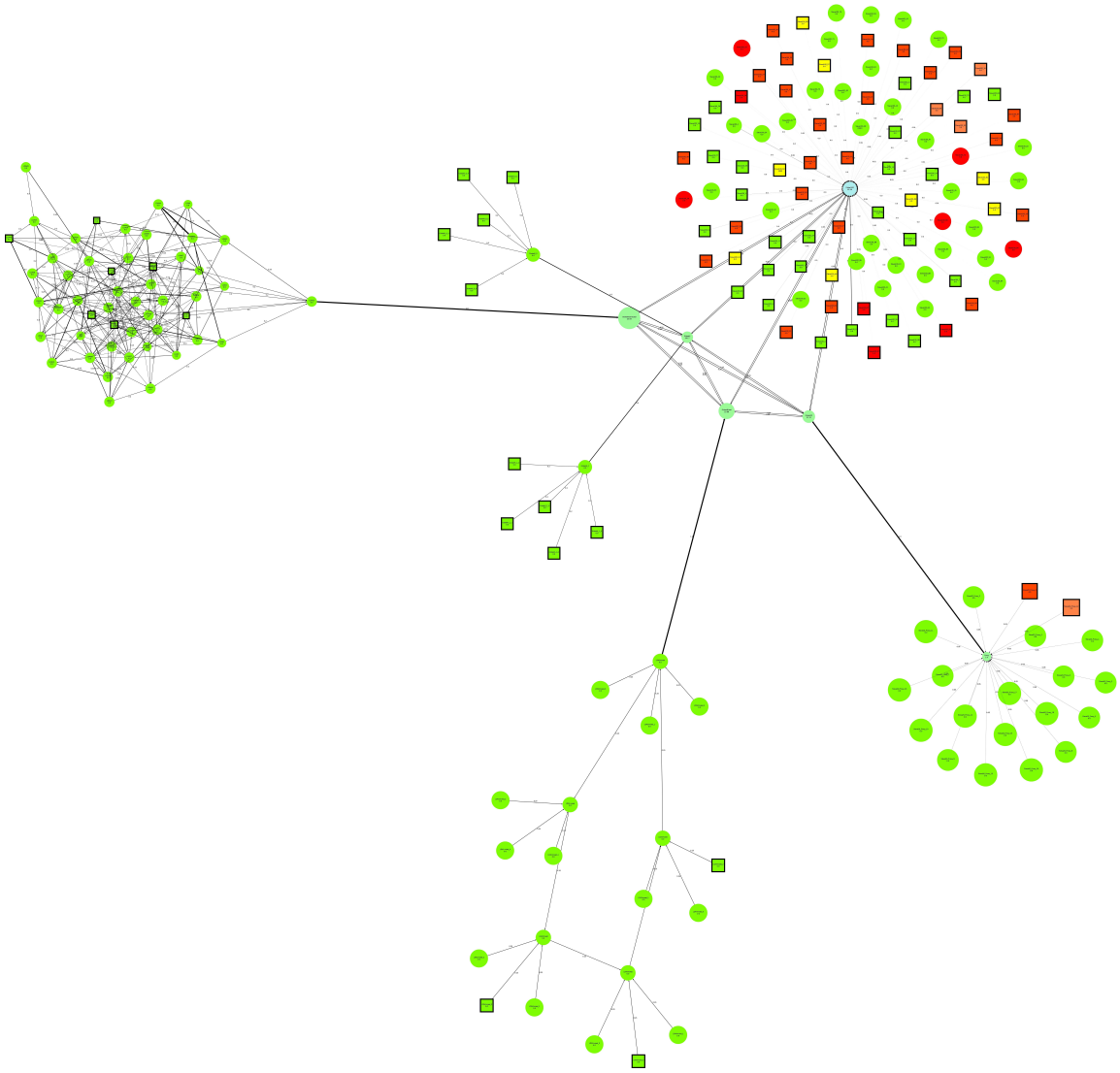


Figure 6.10: CIDN after applying the countermeasures of the solution S4.

likelihood of different attacks being happening is calculated. These attacks are defined regarding its consequences on the IDN, and they have an associated impact. Using the likelihood and the impact of attacks, the global risk of the IDN is calculated.

In order to save resources, it is important to analyze the tradeoff between cost and risk of implementing countermeasures in the channels. To this end, we use a Multi-Objective Optimization algorithm to get optimal allocations of these countermeasures. Concretely, we use an evolutionary algorithm known as SPEA2. This algorithm provides solutions that are pareto optimal, where a solution is the set of countermeasures to be applied in order to protect the channels of the IDN nodes.

In our experimental results we have seen that, depending on the cost and the connections in the network, deciding what to fix and deciding whether it is worth doing can help saving resources. Moreover, we have provided an analysis of a case study that suggest that the use of DEFIDNET helps to determine the optimal allocation of resources in big complex networks.

7

Conclusions

Intrusion Detection Networks (IDNs) are mechanisms that provide security to ICT systems. IDNs constitute a primary component for securing computing infrastructures, and thus have become themselves the target of attacks. This Thesis has focused in the protection of IDNs against adversaries. In this chapter we provide the conclusions of our work. We first summarize the main contributions and discuss how they meet the objectives established. Second, we discuss open issues and future work. Finally, we mention the list of results published related to this Thesis and other publications obtained from parallel work carried out while working on this Thesis.

7.1 Summary of Contributions and Conclusions

We next summarize the contributions and discuss the main conclusions that arise from them:

- In Chapter 2 we have reviewed the state of the art of IDNs in adversarial environments. We have seen that since the beginning of 21st century, many works have considered the possibility of attacks against IDS. Additionally, since a commonly used approach used in anomaly detection is the application of ML algorithms, recently the research community has questioned the robustness of such algorithms in adversarial settings. From the analysis of the state of the art, several conclusions are drawn:

    1. Though the security of ML is an active research topic, little progress has been made over the last few years in the design of resilient algorithms.

Moreover, there is a substantial lack of experimental work exploring the problems derived from an attacker who can modify instances at will to subvert the detection function.

2. In order to secure IDSs against attacks, many state of the art solutions have proposed the use of random components in the detection process that are kept secret for the adversaries. Some of these solutions assume that a potential adversary could not know which parts of the events are being processed by the IDS. However, a formal security analysis of such solutions is still missing.
3. While strong analysis models for attacks on individual IDN nodes have been explored, almost no research works have focused on the study of resilient IDNs in the face of adversaries.

- In Chapter 3 we have presented reverse engineering and evasion attacks, which corroborate the need for robust machine learning algorithms. The reverse engineering process derives a model from a training distribution assumed to be the same the detector uses. This model is then processed by a searching algorithm which suggests evasion strategies. Furthermore, we show that IDS that rely on lightweight feature construction algorithms are easily manipulable by an adversary, facilitating the mapping of feature vectors into real world evasions. The experimental work carried out leads to the following conclusions:

  1. The use of machine learning for intrusion detection, though it is effective and efficient, must consider robustness against adversarial manipulation.
  2. The dataset used to train the classifiers should represent properly the complete data space. Otherwise, the classifiers may learn patterns that are valid for a dataset with such distribution, but are not robust enough to classify data specifically modified by an adversary.
  3. The use of lightweight feature construction methods allows an adversary to obtain real world evasions from the feature vectors. Ideally, in adversarial environments, the feature construction should be a one-way function, i.e., whose invert function is computationally hard to calculate.

- In Chapter 4 we have evaluated the randomization of the detection boundary, which is one of the proposed solutions to counteract reverse engineering and

evasion attacks. We have focused on a popular anomaly detector in the research community called Anagram [Wang et al., 2006]. The proposed reverse engineering attack shows that not only the attacker can infer the decision boundary, but this knowledge indeed makes it easier for an adversary to evade the detector. While the attacks presented in Chapter 4 are not directly applicable to other randomized anomaly detectors, the underlying ideas are general enough and can be used to reverse engineer other schemes based on similar constructions. The work presented leads to the following conclusions:

1. In general, the use of query-response analysis allows an adversary to build "nearly-anomalous" events which may be close to the detection boundary. Then, by performing small incremental modifications and observing the output, the adversary can learn what the decision boundary is.
2. Randomization provides security, but it may turn into a loss of effectiveness, because the inputs are slightly modified internally to hide how they are processed. An adversary who manages to find out the secret information used in the detection, could actually take advantage of the less efficient randomized detection process to evade the IDS, thus turning a security measure into an undesirable feature.
3. From the above conclusions, it can be observed that, while randomization is a promising countermeasure to protect IDSs, further improvements to this technique are required to counteract reverse engineering and evasion attacks.
4. The contributions presented in Chapters 3 and 4 meet the objective **O1** stated in the beginning of this Thesis, which was to "Study techniques to reverse engineer and evade IDSs based on ML algorithms".

- In Chapter 5 we have proposed a system model for IDN that integrates the key features of individual nodes of an IDN and existing architectural options. The system model facilitates the definition of goals, tactics, and capabilities of adversaries aiming at disrupting the IDN operation. After analyzing the main features of IDNs, both in wired an wireless networks, we have built a general model from common building blocks. Accordingly, we have defined a set of common threats against these communication channels, that lead us to the provision of a list of attacks against IDNs. Finally, we have presented

such attacks in two scenarios: MANETs and Collaborative IDN. Regarding this work, we state the following conclusions:

1. The different nodes operating in an IDN share common functional components, which are generalized in a system model. With the proposed model it is possible to define the assets and the adversarial model of an IDN, which facilitates the risk assessment and the design of defense strategies for the IDN.
2. The main goal stated at the beginning of this Thesis was to "improve the security of intrusion detection systems and networks operating in adversarial settings by developing techniques to analyze their vulnerabilities and countermeasures to increase their resiliency". Our findings shows that perfectly securing IDNs in real scenarios is an almost impossible mission. Indeed, it would be required that each independent node in the IDN is properly secured, which is unrealistic in real settings where economical and operational constraints apply. Consequently, it is necessary to provide resilient architectures that maintain the protection operative, even assuming that some nodes are being targeted. Concretely, the system model presented in Chapter 5 accomplishes the objective **O2**.
3. A risk assessment of IDNs involves knowing the assets and threats to which the IDN is exposed. Then, deciding what to fix and how many resources to spend presents a tradeoff between cost and risk. This tradeoff helps to make decisions about when and where it is worth to implement countermeasures. Indeed, depending on the cost and the specific settings of the network, deciding where to allocate countermeasures can aid in saving resources.

- In Chapter 6 we have proposed a framework called DEFIDNET. This framework can be used to obtain a set of countermeasures and evaluate the cost and risk tradeoff. The main steps of the framework are summarized as follows. First, the model of nodes discussed above allows the definition of probabilities of different intrusive actions in each communication channel. Second, the connections and influences between nodes determine how intrusive actions targeted to one node affect the IDN, i.e., how threats are propagated across the IDN. Third, the risk of the IDN is calculated from the probabilities and the impacts of

the attacks. Finally, the framework calculates the set of countermeasures to optimally protect the network, along with a tradeoff between cost and risk of these countermeasures, using a multi-objective optimization algorithm. We have also provided a case study that uses DEFIDNET to determine the optimal allocation of resources in a complex IDN. Next we provide two conclusions from the work carried out in this chapter:

1. IDNs may have many different architectures and operational settings, which makes them a complex scenario. Traditionally, the more complex a system is, the more security breaches it may expose. Accordingly, it is critical to design methods to provide operators with global awareness of the IDNs, including the assets of the IDNs and the threats to which it is exposed. Thus, these methods may facilitate the security evaluation of IDNs.
2. The abstraction offered by DEFIDNET provides several advantages to design resilient architectures for IDNs. On the one hand, it facilitates the definition of the assets of the IDNs and the adversarial capabilities, which facilitates the risk assessment of the IDN. On the other hand, it allows to devise defense strategies, optimizing the allocation of countermeasures that save resources. Accordingly, the objective **O3** established at the beginning of the Thesis is accomplished.

7.2 Open Issues and Future Work

As stated in this Thesis, the sophistication of attackers evolves parallel to the robustness of defenses. Thus, the design of robust countermeasures seems to be a never-ending research topic. In this Thesis, we have provided contributions to counteract current attacks. These contributions admit extended work and open new interesting research challenges. In what follows we discuss future work directions that arise from this Thesis:

- **Defending machine learning from reverse engineering and evasion attacks.** The attacks against ML based IDSs presented in Chapter 3 made some assumptions for the adversary that nowadays are reasonable. Concretely, that

the attacker knows the training data distribution and the feature construction method. Even assuming that this information is available to the adversary, an effective mechanism would be to hide some other relevant information for the detection. This way, the attacker would not know how to find attack vectors that evade the classifier. A recent approach is to use keys in the detection function. These keys, which are secret, determine the internal behavior of the detector. However, as we have also shown in this Thesis, the use of secret information might be vulnerable to reverse engineering attacks if it is not done properly. Thus, further research must be done to improve the robustness of this solution.

The attacks presented actually succeed because the adversary can easily invert the feature construction process, and thus obtain real world evasions from the feature vectors. Accordingly, research on one way feature construction methods (i.e., which cannot be inverted) may counteract such attacks. Still, it would be required a security analysis of these functions before considering them for real scenarios.

- **Generalization of attacks against randomized IDS.** Another open issue derived from this work is the generalization of reverse engineering attacks to randomized IDSs. In Chapter 4, we have described why these attacks succeed and we have detailed a concrete attack against Anagram. Since the same idea can be extended to other randomized detectors using a formal definition, more research work is required to generate similar attack strategies.

- **Improve the robustness of Anagram.** Even though the attack presented in Chapter 4 allows an adversary to recover the random mask used by Anagram, this detector is still a valid candidate to use in real settings. Thus, another open issue is the design of countermeasures against reverse engineering attacks for Anagram. For example, a possible countermeasure to the proposed reverse engineering attack is to randomize the choice of the random mask itself. However, the potential impact of such a double randomization from the detection point of view must be further studied.

  The three points commented until now suggest that attacks and defenses to strengthen the security of IDSs is a race between attackers and defenders. As pointed out before, this race makes the design of attacks and defenses for IDSs

a promising research topic.

- **Implementation of real attacks on state of the art proposals for IDNs.** In Chapter 5, we have presented several attack strategies to IDNs, describing how these strategies would be applied in two scenarios. Concretely, we analyzed IDNs proposed in the literature for MANETs and Collaborative IDNs. Since we have provided conceptual attack strategies, it would be interesting to actually implement such attacks in either simulated networks or actually in real networks. For example, since many IDNs studied for MANETs have been tested using simulated networks, an interesting work would be to simulate adversaries for such IDNs.

- **Update DEFIDNET to facilitate dynamic analysis of IDNs.** One of the advantages of the proposed framework DEFIDNET is that it facilitates the assessment of the risk of IDNs, by virtually defining the assets and adversarial capabilities in the IDN. Thus, it can be applied in dynamic scenarios by properly setting the parameters in real time. The dynamic analysis assumes that the adversarial model changes over time, due to the establishment of new countermeasures in the node channels, the addition of new nodes and connections in the IDN, changes on the influences, etc. This dynamism requires a constant reconfiguration. For example, if it is known that a certain node is compromised and setting countermeasures in this node cannot be afforded, then it may be useful to decrease the influence on this node to reduce the propagation of the risk. Currently, reconfiguration is not optimized in DEFIDNET as it must be performed manually. Thus, automatic reconfiguration of the network would allow to perform a faster, dynamic risk analysis.

  A possible implementation of DEFIDNET with dynamic analysis would be its integration with cloud computing platforms designed to deploy and manage large networks of virtual machines. These virtual machines would be instantiated as nodes of the IDN. Thus, whenever a new virtual machine is created in the network, DEFIDNET may automatically suggest reconfiguration alternatives and countermeasures to reduce the risk of the IDN.

- **Application of DEFIDNET in real scenarios.** We have evaluated DEFIDNET with a simulated network, obtaining promising results. However, it is desirable to assess the framework with real IDNs which are actually protecting

some infrastructure. The framework should be applied in simulated scenarios, which in turn would not differ substantially from the case study provided in Chapter 6. We are currently developing a graphical user interface and we plan to make it publicly available. Thus, users will be able to download and use it, reporting any inconsistency or improvement that they consider to the framework.

7.3 Results

The research work done during this Thesis has resulted in some publications and contributions to journals, conferences, and a book chapter. In this section we first describe the publications that have a direct relation with the Thesis presented in this document. Then, we describe publications obtained from research work carried out while working on this Thesis.

7.3.1 Publications Directly Related to the Thesis

Journal papers

1. **Title:** Randomized Anagram Revisited.
   Authors: Sergio Pastrana, Agustin Orfila, Juan E. Tapiador, Pedro Peris-Lopez.
   Journal: Journal of Network And Computer Applications.
   Reference information: Volume 41. Pages 182-196, May 2014 [Pastrana et al., 2014].
   Impact Factor 2012: 1.467.
   Journal Ranking: Position 11/50, Hardware and Architecture (Q1).

Conference papers

1. **Title:** Anomalous Web Payload Detection: Evaluating the Resilience of 1-grams Based Classifiers.
   Authors: Sergio Pastrana, Carmen Torrano-Gimenez, Hai Than Nguyen and Agustin Orfila.

Conference: 8th International Symposium on Intelligent Distributed Computing.
Reference information: September 2014, Madrid (Spain).

2. **Title:** A Functional Framework to Evade Network IDS.
Authors: Sergio Pastrana, Agustin Orfila and Arturo Ribagorda.
Conference: Hawaii International Conference on Systems Sciences (HICSS44).
Reference information: January 2011, Kauai (USA) [Pastrana et al., 2011].
Conference Ranking: CORE A [Research and Education, 2013].

3. **Title:** Modeling NIDS Evasion Using Genetic Programming.
Authors: Sergio Pastrana, Agustin Orfila and Arturo Ribagorda.
Conference: World Congress in Computer Science, Computer Engineering and Applied Computing.
Reference information: July 2010, Las Vegas, USA [Pastrana et al., 2010].
Conference Ranking: CORE C [Research and Education, 2013].

Book chapters

1. **Title:** Evading IDS and Firewalls as Fundamental Sources of Information in SIEMS.
Authors: Sergio Pastrana, Jose Montero, Agustin Orfila.
Book: Advances in Security Information Management: perceptions and outcomes.
Reference information: NOVA Publishers, ISBN 978-1-62417-221-2 (2013)[Pastrana et al., 2013].

Submitted papers

1. **Title:** DEFIDNET : A Framework for Optimal Allocation of Cyberdefenses In Intrusion Detection Networks.
Authors: Sergio Pastrana, Juan E. Tapiador, Agustín Orfila and Pedro Peris-López.
Journal: Computer Networks (Impact Factor=1.231)
Submission date: May 2014

2.

7.3.2 Related Publications

Journal papers

1. **Title:** Evaluation of Classification Algorithms for Intrusion Detection in MANETs.
   Authors: Sergio Pastrana, Aikaterina Mitrokotsa, Agustin Orfila, Pedro Peris-Lopez.
   Journal: Knowledge Based Systems.
   Reference information: Volume 36. Pages 217-225. December 2012 [Pastrana et al., 2012].
   Impact Factor 2012: 4.104.
   Journal Ranking: Position 6/115, Artificial Intelligence (Q1).

Conference papers

1. **Title:** Artificial Immunity-Based Correlation System.
   Authors: Guillermo Suarez-Tangil, Esther Palomar, Arturo Ribagorda and Sergio Pastrana.
   Conference: International Conference on Security and Cryptography (SECRYPT).
   Reference information: July 2011, Sevilla, Spain [Suarez-Tangil et al., 2011].
   Conference Ranking: CORE B [Research and Education, 2013].

Glossary of Terms

AI	Artificial Intelligence
APT	Advanced Persistent Threat
AS	Attack Strategy
B	Blocking (attack to communications)
C_{ID}	Intrusion Detection Capability index
CIDN	Collaborative Intrusion Detection Network
DC	Data Collection (role)
DDF	Distributed Detection Function
DoS	Denial of Service
EA	Evolutionary Algorithm
ESF	Event Sharing Function
F	Fabrication (attack to communications)
FC	Feature Construction
GP	Genetic Programming
HIDS	Host based Intrusion Detection System
I	Interception (attack to communications)
ICS	Industrial Control System
ICT	Information and Communication Technology
IDMsg	Intrusion Detection Message

IDN	Intrusion Detection Network
IDS	Intrusion Detection System
IIDM	Input Intrusion Detection Message (channel)
LD	Local Detection (role)
LDA	Local Detection and Alert sharing (role)
LDF	Local Detection Function
LE	Local Events (channel)
M	Modification (attack to communications)
MANET	Mobile Ad-hoc Network
ML	Machine Learning
MOO	Multi-Objective Optimization
NIDS	Network based Intrusion Detection System
NSGA2	Non-dominated Sorting Genetic Algorithm
OIDM	Output Intrusion Detection Message (channel)
PC	Pure Correlation (role)
PKI	Public Key Infrastructure
RA	Response Action (channel)
RC	Remote Correlation (role)
RCD	Remote Correlation and Detection (role)
RF	Response Function
SPEA2	Strength Pareto Evolutionary Algorithm v2

References

Al Ameen, M., Liu, J., and Kwak, K. 2012. Security and privacy issues in wireless sensor networks for healthcare applications. *Journal of medical systems 36,* 1, 93–101.

Almenares, F., Arias, P., Marin, A., Diaz-Sanchez, D., and Sanchez, R. 2013. Overhead of using secure wireless communications in mobile computing. *Consumer Electronics, IEEE Transactions on 59,* 2.

Amer, S. H. and Hamilton, J. A. 2010. Intrusion detection systems (IDS) taxonomy - a short review. *Journal of Software Technology 13.*

Antichi, G., Ficara, D., Giordano, S., Procissi, G., and Vitucci, F. 2009. Counting bloom filters for pattern matching and anti- evasion at the wire speed. *IEEE Network: The Magazine of Global Internetworking 23*, 30–35.

Ariu, D., Tronci, R., and Giacinto, G. 2011. HMMPayl: An intrusion detection system based on hidden markov models. *Computers and Security 30,* 4, 221–241.

Ateniese, G., Felici, G., Mancini, L. V., Spognardi, A., Villani, A., and Vitali, D. 2013. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *arXiv preprint arXiv:1306.4447*.

Aziz, A., Salama, M., ella Hassanien, A., and El-Ola Hanafi, S. 2012. Detectors generation using genetic algorithm for a negative selection inspired anomaly network intrusion detection system. In *2012 Federated Conference on Computer Science and Information Systems (FedCSIS)*. 597–602.

Bace, R. and Mell, P. 2001. NIST special publication on intrusion detection systems. Tech. rep., National Institute of Standards and Technologies.

Barreno, M., Nelson, B., Joseph, A. D., and Tygar, J. 2010. The security of machine learning. *Machine Learning 81,* 2, 121–148.

Barreno, M., Nelson, B., Sears, R., Joseph, A. D., and Tygar, J. 2006. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*. ACM, 16–25.

Biggio, B., Fumera, G., and Roli, F. 2010. Multiple classifier systems for robust classifier design in adversarial environments. *International Journal of Machine Learning and Cybernetics 1,* 1-4, 27–41.

Biggio, B., Fumera, G., and Roli, F. 2013. Security evaluation of pattern classifiers under attack. *IEEE Transactions on Knowledge and Data Engineering 99,* 1, 1.

Biggio, B., Nelson, B., and Laskov, P. 2012. Poisoning attacks against support vector machines. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*. 1807–1814.

Bloom, B. H. 1970. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM 13,* 7, 422–426.

Brumley, D. and Boneh, D. 2005. Remote timing attacks are practical. *Computer Networks 48,* 5, 701–716.

Bye, R., Camtepe, S. A., and Albayrak, S. 2010. Collaborative intrusion detection framework: characteristics, adversarial opportunities and countermeasures. In *Proceedings of CollSec: Usenix Workshop on Collaborative Methods for Security and Privacy*.

Champion, T. and Durst, R. 1999. Air force intrusion detection system evaluation environment. In *Recent Advances in Intrusion Detection (RAID)*.

Chen, B., Mashayekh, S., Butler-Purry, K. L., and Kundur, D. 2013. Impact of cyber attacks on transient stability of smart grids with voltage support devices. In *Power and Energy Society General Meeting (PES)*. IEEE, 1–5.

Chen, S., Wang, R., Wang, X., and Zhang, K. 2010. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *Security and Privacy (SP)*. IEEE, 191–206.

Chen, Y., Boehm, B., and Sheppard, L. 2007. Value driven security threat modeling based on attack path analysis. In *Hawaii International Conference on System Sciences. HICSS07.* IEEE, 280a–280a.

Chung, S. and Mok, A. 2006. Allergy attack against automatic signature generation. In *Recent Advances in Intrusion Detection (RAID)*. Springer, 61–80.

Chung, S. and Mok, A. 2007. Advanced allergy attacks: Does a corpus really help? In *Recent Advances in Intrusion Detection (RAID)*. Springer, 236–255.

C.I.A. 2010. The world factbook. https://www.cia.gov/library/publications/the-world-factbook/geos/xx.html.

Common-Criteria. 2012. Common criteria for information technology security evaluation: Part i. Tech. rep. September. http://www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R4.pdf.

Corona, I., Giacinto, G., and Roli, F. 2013. Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues. *Information Sciences 239*, 201–225.

Crosby, S. A. and Wallach, D. S. 2003. Denial of service via algorithmic complexity attacks. In *Proceedings of the 12th USENIX Security Symposium*. Washington: USENIX, 29–44.

Crow, B. P., Widjaja, I., Kim, J. G., and Sakai, P. T. 1997. IEEE 802.11 wireless local area networks. *Communications Magazine, IEEE 35,* 9, 116–126.

Dalvi, N., Domingos, P., Sanghai, S., Verma, D., et al. 2004. Adversarial classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 99–108.

Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. 2000. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA2. *LNCS 1917*, 849–858.

Debar, H., Curry, D., and Feinstein, B. 2007. The intrusion detection message exchange format (IDMEF). RFC-4765. http://tools.ietf.org/html/rfc4765.

Denning, D. E. 1987. An intrusion-detection model. *IEEE Transactions on Software Engineering 2*, 222–232.

Detristan, T., Ulenspiegel, T., Malcom, Y., and Underduk, M. 2003. Polymorphic shellcode engine using spectrum analysis. http://phrack.org/issues/61/9.html.

Djenouri, D., Mahmoudi, O., Bouamama, M., Llewellyn-Jones, D., and Merabti, M. 2007. On securing MANET routing protocol against control packet dropping. In *Proceedings of the International Conference on Pervasive Services*. IEEE Computer Society, CA, USA, 100–108.

Estévez-Tapiador, J., Orfila, A., Ribagorda, A., and Ramos, B. 2013. Key-recovery attacks on KIDS, a keyed anomaly detection system. *IEEE Transactions on Dependable and Secure Computing In Press*.

Falliere, N., Murchu, L. O., and Chien, E. 2011. W32. stuxnet dossier. *White paper, Symantec Corp., Security Response Version 1.4*.

Fogla, P. and Lee, W. 2006. Evading network anomaly detection systems: Formal reasoning and practical techniques. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*. ACM, Alexandria, VA, USA, 59–68.

Fogla, P., Sharif, M., Perdisci, R., Kolesnikov, O., and Lee, W. 2006. Polymorphic blending attacks. In *Proceedings of the 15th USENIX Security Symposium*. USENIX, Vancouver, BC, Canada, 241–256.

Forrest, S., Hofmeyr, S., Somayaji, A., and Longstaff, T. 1996. A sense of self for unix processes. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, Oakland, CA, USA, 120–128.

Fung, C. 2011. Collaborative intrusion detection networks and insider attacks. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications 2,* 1, 63–74.

Fung, C. and Boutaba, R. 2013. *Intrusion Detection Networks: A Key to Collaborative Security*. CRC Press.

Gascon, H., Orfila, A., and Blasco, J. 2011. Analysis of update delays in signature-based network intrusion detection systems. *Computers & Security 30,* 8, 613–624.

Ghosh, A. and Sen, S. 2005. Agent-based distributed intrusion alert system. In *Distributed Computing - IWDC 2004*. LNCS Series, vol. 3326. Springer Berlin Heidelberg, 240–251.

Giacinto, G., Roli, F., and Didaci, L. 2003. Fusion of multiple classifiers for intrusion detection in computer networks. *Pattern recognition letters 24,* 12, 1795–1803.

Gil-Pérez, M., Gómez-Mármol, F., Martínez-Pérez, G., and Skarmeta-Gómez, A. F. 2013. RepCIDN: A reputation-based collaborative intrusion detection network to lessen the impact of malicious alarms. *Journal of Network and Systems Management 21,* 1, 128–167.

Gil-Pérez, M., Gómez-Mármol, F., Martínez-Pérez, G., and Skarmeta-Gómez, A. F. 2014. Building a reputation-based bootstrapping mechanism for newcomers in collaborative alert systems. *Journal of Computer and System Sciences 80,* 3, 571–590.

Gil-Pérez, M., Tapiador, J. E., Clark, J. A., Martínez-Pérez, G., and Skarmeta-Gómez, A. F. 2014. Trustworthy placements: Improving quality and resilience in collaborative attack detection. *Computer Networks 58,* 0, 70 – 86.

Goseva-Popstojanova, K., Anastasovski, G., Dimitrijevikj, A., Pantev, R., and Miller, B. 2014. Characterization and classification of malicious web traffic. *Computers and Security 42*, 92–115.

Gu, G., Fogla, P., Dagon, D., Lee, W., and Skorić, B. 2006. Measuring intrusion detection capability: an information-theoretic approach. In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*. ACM, Taipei,Taiwan, 90–101.

Guzzo, A., Pugliese, A., Rullo, A., and Saccà, D. 2014. Intrusion detection with hypergraph-based attack models. In *Graph Structures for Knowledge Representation and Reasoning*. Springer, 58–73.

Hadziosmanovic, D., Simionato, L., Bolzoni, D., Zambon, E., and Etalle, S. 2012. N-gram against the machine: On the feasibility of the n-gram network analysis for binary protocols. In *Recent Advances in Intrusion Detection (RAID)*. LNCS Series, vol. 7462. Springer, 354–373.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. 2009. The weka data mining software: an update. *ACM SIGKDD explorations newsletter 11,* 1, 10–18.

Hu, B. and Shen, Y. 2012. Machine learning based network traffic classification: a survey. *Journal of Information and Computational Science 9,* 11, 3161–3170.

Huang, L., Joseph, A. D., Nelson, B., Rubinstein, B. I., and Tygar, J. D. 2011. Adversarial machine learning. In *ACM workshop on Security and Artificial Intelligence*. AISec '11. ACM, New York, NY, USA, 43–58.

Huang, Y.-a. and Lee, W. 2003. A cooperative intrusion detection system for Ad Hoc networks. In *ACM workshop on Security of Ad Hoc and Sensor Networks*. ACM, 135–147.

Huang, Y.-a. and Lee, W. 2004. Attack analysis and detection for ad hoc routing protocols. In *Recent Advances in Intrusion Detection (RAID)*. LNCS Series, vol. 3224. Springer, 125–145.

Johnston, M. and Narula-Tam, A. 2012. On the impact of transmission radius on routing efficiency. In *Proceedings of the first ACM MobiHoc workshop on Airborne Networks and Communications*. ACM, 7–12.

Jordan, T. and Taylor, P. 1998. A sociology of hackers. *The Sociological Review 46,* 4, 757–780.

Kachirski, O. and Guha, R. 2003. Effective intrusion detection using multiple sensors in wireless ad hoc networks. In *Hawaii International Conference on System Sciences (HICSS'03)*. IEEE, 8–16.

Karim-Ganame, A., Bourgeois, J., Bidou, R., and Spies, F. 2008. A global security architecture for intrusion detection on computer networks. *Computers & Security 27,* 1, 30–47.

Kayacik, H. G., Zincir-Heywood, a. N., and Heywood, M. I. 2011. Evolutionary computation as an artificial attacker: Generating evasion attacks for detector vulnerability testing. *Evolutionary Intelligence 4,* 4, 243–266.

Kolesnikov, O. and Lee, W. 2005. Advanced polymorphic worms: Evading IDS by blending in with normal traffic. Tech. rep., Georgia Institute of Technology.

Koza, J. R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. The MIT Press.

Kruegel, C., Kirda, E., Mutz, D., Robertson, W., and Vigna, G. 2005. Automating mimicry attacks using static binary analysis. In *Proceedings of the 14th Conference on USENIX Security Symposium*. Vol. 14. USENIX, Baltimore, MD, USA, 11–11.

Kruegel, C., Kirda, E., Mutz, D., Robertson, W., and Vigna, G. 2006. Polymorphic worm detection using structural information of executables. In *Recent Advances in Intrusion Detection (RAID)*. Springer, 207–226.

Kumar, G., Kumar, K., and Sachdeva, M. 2010. The use of artificial intelligence based techniques for intrusion detection: a review. *Artificial Intelligence Review 34,* 4, 369–387.

Kurosawa, S., Nakayama, H., Kato, N., Jamalipour, A., and Nemoto, Y. 2007. Detecting blackhole attack on aodv-based mobile ad hoc networks by dynamic learning method. *IJ Network Security 5,* 3, 338–346.

Land, S. and Fischer, S. 2012. Rapidminer in academic use. Tech. rep., Rapid-I GmbH. 08. http://rapidminer.com/learning/academic-programs/.

Lee, S., Kim, G., and Kim, S. 2011. Self-adaptive and dynamic clustering for online anomaly detection. *Expert Systems with Applications 38,* 12, 14891–14898.

Lee, W., Stolfo, S. J., and Mok, K. W. 1999. A data mining framework for building intrusion detection models. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*. IEEE, 120–132.

Li, W., Parker, J., and Joshi, A. 2012. Security through collaboration and trust in MANETs. *Mobile Networks and Applications 17,* 3, 342–352.

Liao, H.-J., Lin, C.-H. R., Lin, Y.-C., and Tung, K.-Y. 2013. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications 36,* 1, 16–24.

Lippmann, R. P., Fried, D. J., Graf, I., Haines, J. W., Kendall, K. R., McClung, D., Weber, D., Webster, S. E., Wyschogrod, D., Cunningham, R. K., et al. 2000. Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings*. Vol. 2. IEEE, 12–26.

Lowd, D. and Meek, C. 2005. Adversarial learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 641–647.

Luke, S. 2010. The ECJ owner's manual – A user manual for the ECJ evolutionary computation library. Tech. rep. oct. Department of Computer Science, George Mason University.

Macaulay, S. 2007. ADMmutate: Polymorphic shellcode engine. http://www.ktwo.ca/c/ADMmutate-0.8.4.tar.gz.

Marti, S., Giuli, T. J., Lai, K., Baker, M., et al. 2000. Mitigating routing misbehavior in mobile ad hoc networks. In *International Conference on Mobile Computing and Networking*. Vol. 6. 255–265.

McHugh, J. 2000. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM transactions on Information and system Security 3,* 4, 262–294.

Mell, P. and Grance, T. 2011. The NIST definition of cloud computing (draft). *NIST special publication 800,* 145, 7.

Miller, D. and Pearson, B. 2011. *Security information and event management (SIEM) implementation*. McGraw-Hill.

Mrdovic, S. and Drazenovic, B. 2010. KIDS: keyed intrusion detection system. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. DIMVA'10. Springer, 173–182.

Mutz, D., Kruegel, C., Robertson, W., Vigna, G., and Kemmerer, R. A. 2005. Reverse engineering of network signatures. In *Auscert Asia Pacific Information Technology Security Conference*.

Mutz, D., Vigna, G., and Kemmerer, R. 2003. An experience developing an IDS stimulator for the black-box testing of network intrusion detection systems. In *Proceedings of the 19th IEEE Annual Computer Security Applications Conference*. 374–383.

Nechaev, B., Allman, M., Paxson, V., and Gurtov, A. 2010. A preliminary analysis of TCP performance in an enterprise network. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*. USENIX Association, 7–7.

Nelson, B., Rubinstein, B. I., Huang, L., Joseph, A. D., and Tygar, J. 2011. Classifier evasion: Models and open problems. In *Privacy and Security Issues in Data Mining and Machine Learning*. Springer, 92–98.

Newsome, J., Karp, B., and Song, D. 2005. Polygraph: Automatically generating signatures for polymorphic worms. In *Security and Privacy, 2005 IEEE Symposium on*. IEEE, 226–241.

Newsome, J., Karp, B., and Song, D. 2006. Paragraph: Thwarting signature learning by training maliciously. In *Recent Advances in Intrusion Detection (RAID)*. Springer, 81–105.

Nguyen, H. T., Torrano-Gimenez, C., Alvarez, G., Franke, K., and Petrović, S. 2013. Enhancing the effectiveness of web application firewalls by generic feature selection. *Logic Journal of IGPL 21,* 4, 560–570.

Orfila, A., Estevez-Tapiador, J. M., and Ribagorda, A. 2009. Evolving high-speed, easy-to-understand network intrusion detection rules with genetic programming. In *EvoWorkshops on Applications of Evolutionary Computations*. LNCS Series, vol. 5484. Springer, 93–98.

Pastrana, S., Mitrokotsa, A., Orfila, A., and Peris-Lopez, P. 2012. Evaluation of classification algorithms for intrusion detection in MANETs. *Knowledge-Based Systems 36,* 0, 217 – 225.

Pastrana, S., Montero-Castillo, J., and Orfila, A. 2013. Evading IDSs and firewalls as fundamental sources of information in SIEMs. In *Advances in Security Information Management: Perceptions and Outcomes*. Nova Science Publishers, Inc.

Pastrana, S., Orfila, A., and Ribagorda, A. 2010. Modeling NIDS evasion with genetic programming. In *Worldcomp 2010: Security and Management*. 444–448.

Pastrana, S., Orfila, A., and Ribagorda, A. 2011. A functional framework to evade network IDS. In *Hawaii International Conference on System Sciences (HICSS'11)*. IEEE, Koloa,Hawaii,USA, 1–10.

Pastrana, S., Orfila, A., Tapiador, J. E., and Peris-Lopez, P. 2014. Randomized anagram revisited. *Journal of Network and Computer Applications 41*, 182–196.

Patel, A., Taghavi, M., Bakhtiyari, K., and Celestino Junior, J. 2013. An intrusion detection and prevention system in cloud computing: A systematic review. *Journal of Network and Computer Applications 36,* 1, 25–41.

Paxson, V. 1999. Bro: a system for detecting network intruders in real-time. *Computer networks 31,* 23, 2435–2463.

Perdisci, R., Ariu, D., Fogla, P., Giacinto, G., and Lee, W. 2009. McPAD: A multiple classifier system for accurate payload-based anomaly detection. *Computer Networks 53*, 864–881.

Perdisci, R., Dagon, D., Lee, W., Fogla, P., and Sharif, M. 2006. Misleading worm signature generators using deliberate noise injection. In *Security and Privacy, 2006 IEEE Symposium on*. IEEE, 15–pp.

Perkins, C. 2003. RFC 3561: Ad hoc On-Demand Distance Vector (AODV) Routing. Status: EXPERIMENTAL.

Ptacek, T. H. and Newsham, T. N. 1998. Insertion, evasion, and denial of service: Eluding network intrusion detection. Tech. rep., Secure Networks, Inc., Syracuse, NY, USA.

Research, C. and Education. 2013. The CORE conference ranking exercise. [http://core.edu.au/index.php/categories/conferenceaccessed Mars-2014].

Rieck, K., Trinius, P., Willems, C., and Holz, T. 2011. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security 19,* 4, 639–668.

Roesch, M. 1999. Snort: Lightweight intrusion detection for networks. In *Proceedings of the 13th Systems Administration Conference*. USENIX, Seattle, WA, USA, 229–238.

Rubinstein, B. I., Nelson, B., Huang, L., Joseph, A. D., Lau, S.-h., Rao, S., Taft, N., and Tygar, J. D. 2009. Antidote: Understanding and defending against poisoning of anomaly detectors. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*. IMC '09. ACM, New York, NY, USA, 1–14.

Rubinstein, B. I., Nelson, B., Huang, L., Joseph, A. D., Lau, S.-h., Taft, N., and Tygar, D. 2008. Compromising PCA-based anomaly detectors for network-wide traffic. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2008-73*.

Santora, M. 2013. In hours, thieves took $45 million in ATM scheme. *New York Times May, 9*.

Satpute, K., Agrawal, S., Agrawal, J., and Sharma, S. 2013. A survey on anomaly detection in network intrusion detection system using particle swarm optimization based machine learning techniques. *Advances in Intelligent Systems and Computing 199 AISC*, 441–452.

Scarfone, K. A. and Mell, P. M. 2007. SP 800-94. guide to intrusion detection and prevention systems (IDPS). Tech. rep., Gaithersburg, MD, United States.

Sekar, R., Gupta, A., Frullo, J., Shanbhag, T., Tiwari, A., Yang, H., and Zhou, S. 2002. Specification-based anomaly detection: a new approach for detecting network intrusions. In *Proceedings of the 9th ACM conference on Computer and communications security*. ACM, 265–274.

Sen, S. and Clark, J. A. 2009. A grammatical evolution approach to intrusion detection on mobile ad hoc networks. In *Proceedings of the Second ACM Conference on Wireless Network Security*. WiSec'09. ACM, NY, USA, 95–102.

Sen, S. and Clark, J. A. 2011. Evolutionary computation techniques for intrusion detection in mobile ad hoc networks. *Computer Networks 55,* 15, 3441 – 3457.

Sen, S., Clark, J. A., and Tapiador, J. E. 2010. Power-aware intrusion detection in mobile ad hoc networks. In *Ad Hoc Networks*. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering Series, vol. 28. Springer, 224–239.

Shankar, U. 2003. Active mapping: Resisting NIDS evasion without altering traffic. In *Security and Privacy*. IEEE, Oakland, California, USA, 44–61.

Shin, S. and Gu, G. 2010. Conficker and beyond: a large-scale empirical study. In *Proceedings of the 26th Annual Computer Security Applications Conference*. ACM, 151–160.

Smith, R., Estan, C., and Jha, S. 2006. Backtracking algorithmic complexity attacks against a NIDS. In *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*. IEEE, 89–98.

Snapp, S. R., Brentano, J., Dias, G. V., Goan, T. L., Heberlein, L. T., Ho, C.-L., Levitt, K. N., Mukherjee, B., Smaha, S. E., Grance, T., et al. 1991. DIDS: distributed intrusion detection system -motivation, architecture, and an early prototype. In *National Computer Security Conference*. 167–176.

Sommer, R. and Paxson, V. 2010. Outside the closed world: On using machine learning for network intrusion detection. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*. SP '10. IEEE Computer Society, Washington, DC, USA, 305–316.

Song, J., Takakura, H., Okabe, Y., and Nakao, K. 2013. Toward a more practical unsupervised anomaly detection system. *Information Sciences: an International Journal 231*, 4–14.

Song, Y., Locasto, M. E., Stavrou, A., Keromytis, A. D., and Stolfo, S. J. 2007. On the infeasibility of modeling polymorphic shellcode. In *Proceedings of the 14th ACM conference on Computer and communications security*. CCS '07. ACM, New York, NY, USA, 541–551.

Stoneburner, G., Goguen, A., and Feringa, A. 2002. Risk management guide for information technology systems. *Nist special publication 800,* 30, 800–30.

Su, M.-Y. 2011. Prevention of selective black hole attacks on mobile ad hoc networks through intrusion detection systems. *Computer Communications 34*, 107–117.

Suarez-Tangil, G., Palomar, E., Pastrana, S., and Ribagorda, A. 2011. Artificial immunity-based correlation system. In *International Conference on Security and Cryptography (SECRYPT 2011)*. 422–425.

Suarez-Tangil, G., Tapiador, J. E., Peris, P., and Ribagorda, A. 2013. Evolution, detection and analysis of malware for smart devices. *IEEE Communications Surveys & Tutorials PP,* 99, 1–27.

Sun, B., Wu, K., and Pooch, U. 2006. Zone-based intrusion detection system for mobile ad hoc networks. *International Journal of Ad Hoc and Sensor Wireless Networks 2*.

Sun, B., Wu, K., and Pooch, U. W. 2003. Routing anomaly detection in mobile ad hoc networks. In *International Conference on Computer Communications and Networks. ICCCN 2003.* IEEE, 25–31.

Takai, M., Martin, J., Bagrodia, R., and Ren, A. 2002. Directional virtual carrier sensing for directional antennas in mobile ad hoc networks. In *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*. ACM, 183–193.

Tan, K., Killourhy, K., and Maxion, R. 2002. Undermining an anomaly-based intrusion detection system using common exploits. In *Recent Advances in Intrusion Detection (RAID)*. Springer-Verlag, Zurich, Switzerland, 54–73.

Tankard, C. 2011. Advanced persistent threats and how to monitor and deter them. *Network security 2011,* 8, 16–19.

Torrano-Gimenez, C., Nguyen, H. T., Alvarez, G., and Franke, K. 2012. Combining expert knowledge with automatic feature extraction for reliable web attack detection. *Security and Communication Networks*.

Torrano-Gimenez, C., Perez-Villegas, A., and Alvarez, G. 2009. A self-learning anomaly-based web application firewall. In *Computational Intelligence in Security for Information Systems*. Springer, 85–92.

Torrano-Gimenez, C., Perez-Villegas, A., and Alvarez, G. 2010. The HTTP dataset CSIC 2010. http://www.isi.csic.es/dataset/.

Tsai, C.-F., Hsu, Y.-F., Lin, C.-Y., and Lin, W.-Y. 2009. Intrusion detection by machine learning: A review. *Expert Systems with Applications 36,* 10, 11994–12000.

Ullrich, J. 2000. Dshield. www.dshield.org.

Varghese, G., Fingerhut, J. A., and Bonomi, F. 2006. Detecting evasion attacks at high speeds without reassembly. In *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM, Pisa, Italy, 327–338.

Vigna, G., Robertson, W., and Balzarotti, D. 2004. Testing network-based intrusion detection signatures using mutant exploits. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*. ACM, Washington, DC, USA, 21.

Vutukuru, M., Balakrishnan, H., and Paxson, V. 2008. Efficient and robust TCP stream normalization. In *IEEE Symposium on Security and Privacy*. IEEE, Oakland, California, USA, 96–110.

Wagner, D. and Soto, P. 2002. Mimicry attacks on host-based intrusion detection systems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*. ACM, Washington, DC, USA, 255–264.

Wang, K. 2007. Network payload-based anomaly detection and content-based alert correlation. Ph.D. thesis, New York, NY, USA.

Wang, K., Parekh, J. J., and Stolfo, S. J. 2006. Anagram: A content anomaly detector resistant to mimicry attack. In *Recent Advances in Intrusion Detection (RAID)*. LNCS Series, vol. 4219. Springer, Hamburg, Germany, 226–248.

Wang, K. and Stolfo, S. 2004. Anomalous payload-based network intrusion detection. In *Recent Advances in Intrusion Detection (RAID)*. LNCS Series, vol. 3224. Springer, 203–222. 10.1007/978-3-540-30143-1_11.

Watson, D., Smart, M., Malan, R. G., and Jahanian, F. 2004. Protocol scrubbing: network security through transparent flow modification. *IEEE/ACM Transactions on Networking 12,* 2, 261–273.

Xenakis, C., Panos, C., and Stavrakakis, I. 2011. A comparative evaluation of intrusion detection architectures for mobile ad hoc networks. *Computers & Security 30,* 1, 63–80.

Yegneswaran, V., Barford, P., and Jha, S. 2004. Global intrusion detection in the DOMINO overlay system. In *NDSS*.

Yi, P., Hou, Y. F., Zhong, Y., Zhang, S., and Dai, Z. 2006. Flooding attack and defence in ad hoc networks. *Journal of Systems Engineering and Electronics 17,* 2, 410 – 416.

Ylipulli, J., Suopajarvi, T., Ojala, T., Kostakos, V., and Kukka, H. 2013. Municipal wifi and interactive displays: Appropriation of new technologies in public urban spaces. *Technological Forecasting and Social Change In Press*.

Zhang, X., Sekiya, Y., and Wakahara, Y. 2009. Proposal of a method to detect black hole attack in manet. In *International Symposium on Autonomous Decentralized Systems. ISADS'09.* IEEE, 1–6.

Zhang, Y. and Lee, W. 2000. Intrusion detection in wireless ad-hoc networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*. MobiCom. ACM, New York, NY, USA, 275–283.

Zhang, Y., Lee, W., and Huang, Y. 2003. Intrusion detection techniques for mobile wireless networks. *Wireless Networks 9*, 545–556.

Zhou, C. V., Leckie, C., and Karunasekera, S. 2010. A survey of coordinated attacks and collaborative intrusion detection. *Computers & Security 29,* 1, 124–140.

Zitzler, E., Laumanns, M., and Thiele, L. 2001. SPEA2: Improving the strength pareto evolutionary approach. *EUROGEN 2001. Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*, 95–100.

A

Detailed Description of the Reverse Engineering Attack Against Anagram

The attack to recover the secret mask is divided into several functions. In order to limit the execution of the algorithm, it takes as inputs two estimated thresholds: the number of sets and the maximum random mask length (see Figure 4.2). We next provide a pseudocode description of the attack divided into 4 algorithms, namely Algorithm 3, 4, 5 and 6.

As shown in Algorithm 3, the attack starts by initializing the mask with the value '-1'. The value '-1' in the position I means that the algorithm has not yet obtained the set corresponding to I. The algorithm also initializes an empty list of the positions that are first delimiters. These two structures are global to all the processes. At each iteration of the algorithm's main loop, the function *findSet* is called with the starting point I_s and the current set $S_{current}$. This starting point is the position from which the algorithm will search delimiters of the set starting in this position ($S_{current}$). In the first step, the algorithm starts at 0 and obtains the delimiters of the set 0 of the mask. Once obtained, it will proceed similarly but starting from a new point (line 9 in Algorithm 3), specifically the next delimiter whose set has not been processed yet (i.e., a delimiter whose next position in the mask is still '-1').

The function *findSet*, shown in Algorithm 4, receives the starting position I_s and the current set $S_{current}$. This is the core function of the attack. First, it selects a payload P which is considered "normal" by calling the function *anagramTest* (lines 4-9 in Algorithm 4). Next, it looks for a "nearly-anomalous" payload P^* that is used in the following steps of the algorithm. It does so by calling the function *getValidPayload*, which is shown in Algorithm 6 and explained in Figure 4.3. The

malicious byte μ is inserted into the 10 positions immediately next to the starting point of payload P (lines 2-5 in Algorithm 6). If this modification causes the payload to become anomalous (line 7 in Algorithm 6), then it removes the byte μ backwards, one position at a time, until the packet becomes normal again. This process will only work with payloads P that are close enough to become anomalous. As we can only use 10 positions following the starting point I_s (by definition these are the only positions that belong to the same set with certainty), we need payloads classified as normal that, with the addition of just a few previously unseen bytes, become anomalous.

Due to the behavior of the randomized test of Anagram, P^* (function *findSet* showed in Algorithm 4) will become anomalous only when μ is inserted into chunks of data that are mapped into the set $S_{current}$. The key point here is that when μ is inserted in the region of a set different from $S_{current}$, the *anagramTest* will still output "normal", as this μ is no longer considered within the bytes of the set S. Therefore, by sliding the malicious byte through the payload (lines 17-28 in Algorithm 4), as shown in the Figure 4.4, and looking where the output of *anagramTest* changes, we can estimate where the delimiters of the set $S_{current}$ are. Regarding Figure 4.4, in the steps 1, 2, 3 and 4, *anagramTest* may output "anomaly" because the malicious byte is still inserted in the *Set 1*, whereas steps 5 to 14 may output "normal". However, in the step 15, as the malicious byte is inserted again into the *Set 1*, it may output again "anomaly". In order to optimize the process and to avoid unnecessary queries to Anagram, this step is skipped (line 18 in Algorithm 4) if the position where μ is going to be inserted has a value different from '-1' in the estimated random mask. We repeat this process for several payloads and record the delimiters indicated by each of them, which we call VOTES (line 24 in Algorithm 4). The final step of the algorithm is to process all votes using the function *processVotes* (line 31 in Algorithm 4).

The function *processVotes*, shown in Algorithm 5, processes the votes obtained by all payloads. We consider that a position is a delimiter of $S_{current}$ if it is supported by at least half of the votes (line 4 in Algorithm 5). If so, we add this position to the final list of delimiters of $S_{current}$: $D_1, D_2, ..., D_d$. The algorithm also saves the list of possible next starting points ($FIRST\_DELIMITERS$). Then, the estimated mask is updated by setting the set $S_{current}$ to the positions between the consecutive delimiters D_a and D_b (lines 15-17 of Algorithm 5). Finally, the function obtains and

returns the next starting delimiter whose set has not been already obtained (lines 22-26 in Algorithm 5). This $NEXT\_I_s$ delimiter will be the next starting point (I_s) for the algorithm (line 5 in Algorithm 3).

Algorithm 3 *FindMask*

Input: Number of estimated Sets NS and maximum estimated random length MAX_LENGTH

Output: Estimated $MASK$.

1: $MASK = \{-1, ..., -1\}$
2: $FIRST\_POSITIONS = \emptyset$
3: $I_s \leftarrow 0$
4: **for** $S = 0 \rightarrow NS$ **do**
5: $\{NEXT\_I_s\} \leftarrow findSet\,(I_s, S)$
6: **if** $NEXT\_I_s = NULL$ **then**
7: **return** $MASK$
8: **end if**
9: $I_s \leftarrow NEXT\_I_s$
10: $S \leftarrow S + 1$
11: **end for**
12: **return** $MASK$

Algorithm 4 *findSet*

Input: Initial position I_s and current set S
Output: Next starting positions $NEXT\_I_s$.

```
1:  VALID = 0
2:  VOTES = {0, ..., 0}
3:  while VALID < NUMPAYLOADS do
4:    P ← getPayloadFromPool ()
5:    OUTPUT ← anagramTest (P)
6:    while OUTPUT ≠' NORMAL' do
7:      P ← getPayloadFromPool ()
8:      OUTPUT ← anagramTest (P)
9:    end while
10:   P* ← getValidPayload (P, I_s)
11:   if P* ≠ NULL then
12:     VALID ← VALID + 1
13:     POSITION ← lastIndexOf (μ, P*) + 1
14:     LOOKER ← P*
15:     LOOKER[POSITION] = μ
16:     PREVIOUS ← anagramTest (LOOKER)
17:     while POSITION < MASK.LENGTH do
18:       if MASK[POSITION] < 0 then
19:         POSITION ← POSITION + 1
20:         LOOKER ← P*
21:         LOOKER[POSITION] = μ
22:         OUTPUT ← anagramTest (LOOKER)
23:         if OUTPUT ≠ PREVIOUS then
24:           VOTES[POSITION] ← VOTES[POSITION] + 1
25:         end if
26:       end if
27:       PREVIOUS ← OUTPUT
28:     end while
29:   end if
30: end while
31: return processVotes (VOTES, VALID, S)
```

Algorithm 5 *processVotes*

Input: An array of votes $VOTES$, the number of valid packets $VALID$, the current set S, the current starting position I_s

Output: Next starting delimiter $NEXT\_I_s$.

1: $DELIMITERS \leftarrow \{I_s\}$
2: $END\_DELIMITER \leftarrow' TRUE'$
3: **for** $I = 0 \rightarrow VOTES.LENGTH$ **do**
4: **if** $VOTES[I] >= VALID/2$ **then**
5: $DELIMITERS.add\,(I)$
6: **if** $END\_DELIMITER$ **then**
7: $END\_DELIMITER \leftarrow' FALSE'$
8: $FIRST\_DELIMITERS.add\,(I)$
9: **else**
10: $END\_DELIMITER \leftarrow' TRUE'$
11: **end if**
12: **end if**
13: **end for**
14: **for** $I = 0 \rightarrow DELIMITERS.LENGTH - 1$ **do**
15: **for** $J = DELIMITERS[I] \rightarrow DELIMITERS[I + 1]$ **do**
16: $MASK[J] = S$
17: **end for**
18: $I \leftarrow I + 2$
19: **end for**
20: $sort\,(FIRST\_DELIMITERS)$
21: $NEXT\_I_s \leftarrow FIRST\_DELIMITER[0]$
22: **while** $MASK[NEXT\_I_s + 1] > 0\, AND\, NEXT\_I_s \neq NULL$ **do**
23: $FIRST\_DELIMITERS.remove\,(0)$
24: $NEXT\_I_s \leftarrow FIRST\_DELIMITER[0]$
25: **end while**
26: **return** $NEXT\_I_s$

Algorithm 6 *getValidPayload*

Input: A payload P and the starting position I_s.
Output: A payload P^* or "$NULL$".

```
1:  P* ← P
2:  for i = I_s → I_s + 10 do
3:      P*[i] = μ
4:      i ← i + 1
5:  end for
6:  output ← anagramTest(P)
7:  if output ='ANOMALY' then
8:      for i = I_s + 10 → I_s do
9:          P*[i] = P[i]
10:         output ← anagramTest(P*)
11:         if output ='NORMAL' then
12:             return P*
13:         end if
14:         i ← i − 1
15:     end for
16: end if
17: return NULL
```