Chapter 7

EVADING IDSS AND FIREWALLS AS FUNDAMENTAL SOURCES OF INFORMATION IN SIEMS

***Sergio Pastrana*[*], *José Montero-Castillo*[†], *Agustín Orfila*[‡]**
Computer Science Department
University Carlos III of Madrid, Spain

Keywords: Security Information and Event Management (SIEM), Evaluation, Evasion, Intrusion Detection, Firewalls.

Abstract

A Security Information and Event Management system (SIEM) is typically composed of two parts: a central entity gathering, aggregating, correlating and analysing information and a set of independent monitoring entities in charge of supplying the central entity with suitable information (e.g., intrusion alerts or system logs). Evaluating a SIEM requires evaluating three elements: the central entity, the monitoring systems and the communications between the monitoring systems and the central entity (it must be ensured that the information arriving at the central entity is integral and complete). Nowadays, two of the most important monitoring systems forming a SIEM are Intrusion Detection Systems (IDSs) and firewalls. Such technologies are currently so sophisticated detecting and/or blocking malicious activities that it becomes difficult for an attacker to compromise a system without being detected or blocked. For this reason, a new adversarial model has arisen in the recent years: instead of directly attacking a protected network or system, the attacker tries to bypass the security barriers without raising suspicion. In this chapter, we present and analyse some of the most relevant techniques presented in the literature so as to evade IDSs and firewalls.

[*]E-mail address: spastran@inf.uc3m.es
[†]E-mail address: jmcastil@inf.uc3m.es
[‡]E-mail address: adiaz@inf.uc3m.es

1. Introduction

Nowadays, IT systems are so large and complex that they usually need to be deployed in a decentralized manner. In order to ease such a decentralization process, new paradigms of computation have appeared in the latest years, for instance, cloud computing. In spite of the important features that decentralized systems have (e.g., availability and scalability), there is still an important challenge to overcome: security. In a decentralized system, the information is usually distributed and replicated among different servers, accessed by a great number of different users and communicated through many different types of links and devices. Consequently, a large variety of problems can arise in a decentralized system: broken links, security vulnerabilities in the operating system installed in a particular server, careless administrators tackling with security options, etc.

In order to secure a decentralized system, it is essential, among other things, to monitor the system and report all the anomalies detected. Such an operation can be performed by a Security Information Event Management system (SIEM), a tool in charge of collecting audit data (data obtained through monitoring), correlating it, analysing it and using the result of the analysis to decide whether abnormal behaviours are present in the system or not. In a SIEM, audit data can come from very different sources: firewalls, Intrusion Detection Systems (IDSs), system logs (for example, UNIX syslog or Microsoft Windows Event Viewer), traces from user behaviours, etc. In fact, the effectiveness of these systems strongly depends on the quality of the audit data, which is supposed to be real, complete and confident.

Figure 1 shows a SIEM with a centralized architecture monitoring an Intranet. The structure of the Intranet is as follows: firstly, the external traffic is received and distributed by a router; then, the traffic is filtered by a firewall and inspected by a Network IDS (NIDS); finally, a switch organizes the allowed traffic over the different systems connected to the Local Area Network (LAN). As for the monitoring process, all the devices in the Intranet are enabled to collect audit data (firewall alerts, system logs, etc.) and send such data to a central entity in the SIEM. Finally, the central entity in the SIEM aggregates, correlates and analyses the received data and reports an alert in case an anomaly is detected.

System administrators make use of SIEMs to determine if any threatening activity is currently taking place in their systems. The sooner an administrator realizes a malicious activity is happening, the sooner she can take actions to stop it and minimize its impact. As a consequence, if an attacker is able to evade the detection of a system's SIEM, she can stealthily compromise the system and thus, she becomes a serious and risky adversary.

Preventing an attacker from evading a SIEM requires to properly evaluate the SIEM, which is an extremely difficult task due to all the variables involved. In particular, three main steps are important when evaluating a SIEM:

1. The entity or entities aggregating, correlating and analysing the audit data from the monitored system must be evaluated to ensure that the SIEM efficiently detects the major number of attacks by generating the minimal number of false alarms.

2. Each audit data collector must be independently evaluated in order to guarantee that the collected data is correct and real.

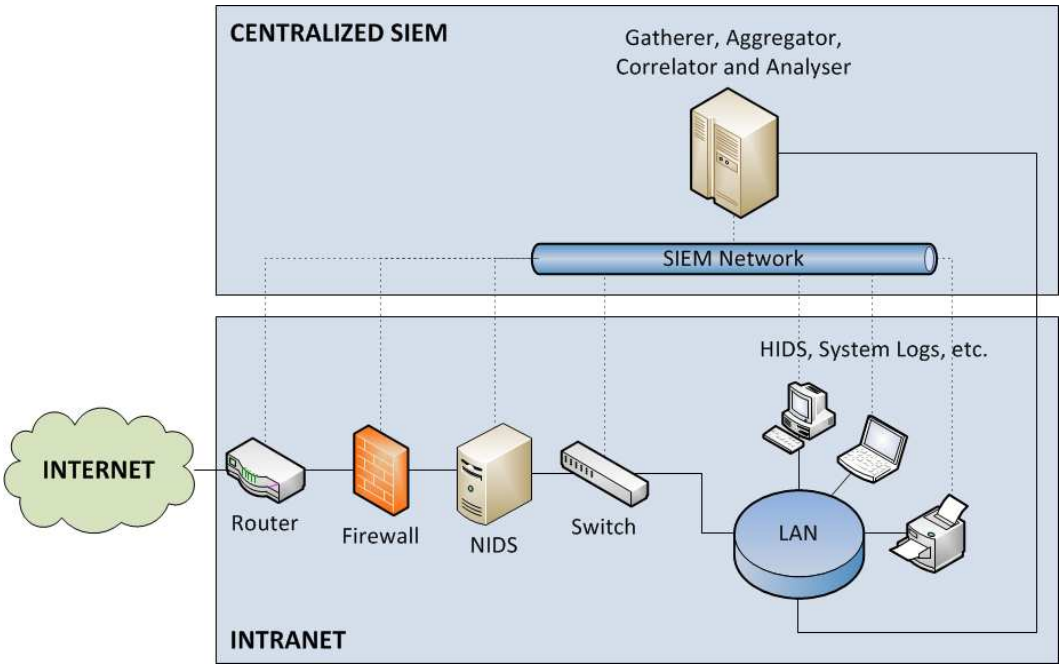

Figure 1. Architecture of a centralized SIEM monitoring an Intranet

3. The communications between the different entities in the SIEM must be secured in order to avoid attacks such as packet injection or packet modification, whose effects can clearly compromise the analyses carried out by the SIEM.

This chapter focuses on the second step of SIEMs evaluation. In particular, the chapter presents an overview of the existing methods aimed to evade two of the most important audit data collectors, namely IDSs and firewalls.

The rest of the chapter is organized as follows. Section 2 provides a description of IDSs and firewalls. Section 3 presents the most important techniques used to evade IDSs. Section 4 comments on the current state of firewall evasion. Finally, Section 5 provides some general conclusions about SIEM evasion.

2. Background

In this section, the audit data collectors that will be analysed later in this chapter are described in detail.

2.1. Intrusion Detection System (IDS)

An IDS is a system that analyses data so as 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 (see Figure 2), 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 following:

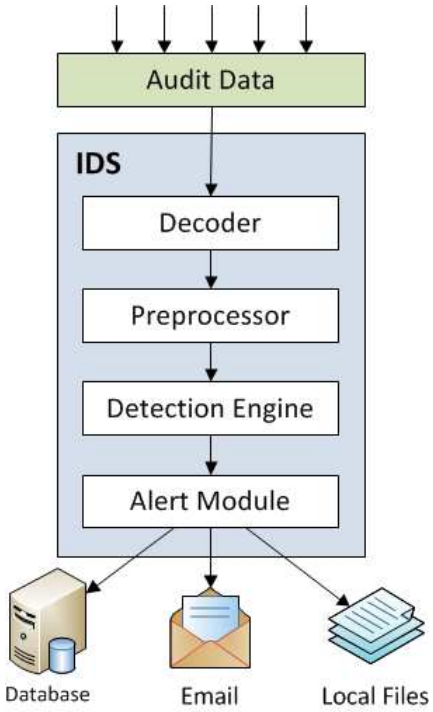

Figure 2. Architecture of a classical IDS

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 receives the pieces of data transformed by the decoder, analyses them to determine which pieces are dependant on each other and treats dependant pieces in such a way that they can be later scrutinized by the detection engine. A typical preprocessor widely used in NIDSs 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.

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.

As for IDS classification, there exist many different taxonomies. Following, the most important and currently used ones are presented:

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

   (a) Network IDSs (NIDSs): they analyse network traffic. The level of detection may vary from one NIDS to another, but most of them have modules in charge of analysing traffic from the network, transport and application layers in the OSI model. For instance, Snort [1], 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 (see Figure 1).
   (b) Host IDSs (HIDSs): they analyse the data in one particular device. Most of them analyse the sequence of system calls of the programs running in the device perform. Within these sequences, optimal HIDS analyse system call arguments, memory registers, stack states, system logs, user behaviours, etc.

2. Regarding the model used to detect malicious activity, an IDS can be signature-based, anomaly-based or hybrid:

   (a) Signature-based IDSs: in this type of IDS, abnormal behaviours are modelled as rules (also known as signatures) and intrusion detection is accomplished by comparing these rules with the behaviours taking place in the monitored system.
   (b) Anomaly-based IDSs: in this type of IDS, the normal behaviours of the system are represented in a model and any activity falling out of the model is considered abnormal.
   (c) Hybrid IDS: in this type of IDS, both signature and anomaly-based techniques are combined. Normally, the preprocessor performs the anomaly-based detection process and the detection engine performs the signature-based one.

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

   (a) Passive IDS: when a malicious behaviour 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, working so as an Intrusion Prevention System (IPS).

Apart from the previous classifications, there are many other possible ones. For example, in [2], 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.

Table 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

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) measures the effectiveness of an IDS by indicating the percentage of intrusions that it detects (see Equation 1). The false positive rate (denoted F) measures the accuracy of an IDS by indicating the percentage of false alarms that it raises (see Equation 2). In order to calculate these two statistics, the following four values are necessary (see contingency matrix in Table 1): the number of real intrusions detected (true positives), the number of real intrusions undetected (false negatives), the number of alarms raised without any real intrusion taking place (false positives) and the number of normal events considered normal (true negatives).

$$H = \frac{TP}{TP + FN} \tag{1}$$

$$F = \frac{FP}{FP + TN} \tag{2}$$

Another important statistic that in the recent years has become relatively popular in the field of IDSs evaluation is the intrusion detection capability index [3] (denoted C_{ID}, see Equation 3). It measures the amount of uncertainty of the input resolved once the IDS output is obtained, and takes into account the prevalence (B in the formula) in the dataset besides the hit rate and the false positive rate. Since not all systems have the same probability of being attacked, the intrusion detection capability index provides a more accurate measure than the hit rate and the false positive rate. Moreover, evaluating an IDS using H and F is ambiguous, as it must be defined an optimal trade-off 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 (the higher the C_{ID} is, the better the IDS is).

$$\begin{aligned} C_{ID} = & -BH \log \frac{BH}{BH + HF} - \\ & - B(1-H) \log \frac{B(1-H)}{B(1-H) + (1-B)(1-F)} - \\ & - (1-B)(1-F) \log \frac{(1-B)(1-F)}{(1-B)(1-F) + B(1-H)} - \\ & - (1-B)F \log \frac{(1-B)F}{(1-B)F + BH} \end{aligned} \tag{3}$$

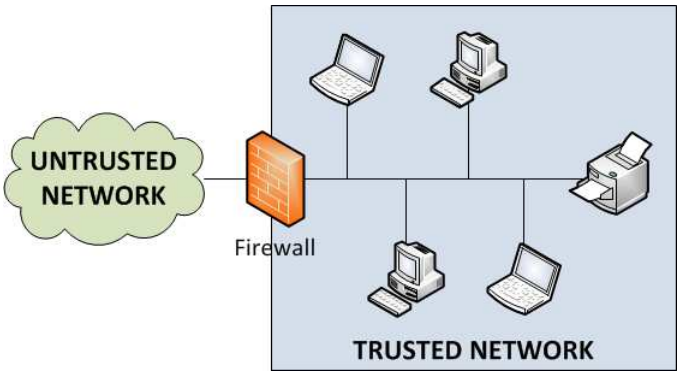

Figure 3. Basic firewall scenario

2.2. Firewall

A firewall is a security component (hardware and/or software) designed to restrict the access between a trusted and an untrusted network (see Figure 3). Note that the term *trusted network* can refer from a single device (e.g., a personal computer) to a complex set of heterogeneous devices (e.g., a company's LAN). As for the term *untrusted network*, it can refer to any type of network; however, it usually refers to the Internet.

The main functions of a firewall are to examine all the packets moving between the trusted and the untrusted network, and to discard all the packets that are not accepted regarding the firewall's security policy. A firewall's security policy consists of a set of filtering rules in which each rule indicates the properties (e.g., source IP address) that packets must have in order to be discarded or accepted. For instance, a typical filtering rule used to prevent the employees of a company from connecting to a particular site is to define a rule indicating that any outgoing packet (a packet going from the trusted to the untrusted network) with a destination IP address equal to the site's IP address must be discarded.

Firewalls can be classified regarding different characteristics:

1. Regarding the type of analysis performed so as to decide whether a packet is discarded or not, firewalls can be classified as follows [4]:

   (a) Static packet filters: in these types of firewalls, the decision of discarding a packet depends only on the value of five of its header fields, namely the source and destination IP addresses, the source and destination port numbers, and the type of transport layer protocol. Consequently, the filtering rules specified in the firewall's security policy use these five header fields to determine which traffic is allowed and which traffic is denied.

   (b) Dynamic packet filters: these types of firewalls are an extension of the static packet filters. They perform the same operations as their predecessors but taking into account connection streams (a connection stream is the set of all the packets with the same source and destination IP addresses, source and destination port numbers, and transport layer protocol). In particular, these firewalls maintain a table of allowed connection streams and use it to check if a particular

packet belongs to an existing connection stream. If it does, no further analysis is performed and the packet is allowed to traverse the firewall. If it does not, the packet is analysed according to the firewall's filtering rules and in case the packet is allowed, the table of connection streams is updated.

(c) Connection filters: these firewalls are an extension of the dynamic packet filters. Apart from the operations that any dynamic packet filter would do, they verify that each TCP packet belongs to a valid TCP connection. In order to do that, they check three fields in the packet's TCP header, namely the SYN and the ACK flags, and the sequence number.

(d) Application filters: these firewalls extend the functionality of connection filters. In addition to the operations that their predecessors perform, these firewalls analyse the various fields contained in each packet's application header. For instance, an application filter can be configured to discard all the packets not complying with the HTTP protocol. The problem of these types of firewalls is that they have to manage information regarding different application-layer protocols and therefore, the time and resources consumed in the detection process may be so high that it may result inefficient.

2. Regarding the type of trusted networks that firewalls protect, they can be classified as follows:

   (a) First-generation firewalls: traditional firewalls, which are usually deployed as dedicated devices, are in charge of protecting networks from unwanted communication activities.

   (b) Web application firewalls (WAFs): a WAF is an enhanced application filter in which the trusted network contains a web server. Apart from the typical operations that any application filter would do, these types of firewalls are specialised in the HTTP protocol and therefore, they analyse the payload of each packet's application layer. With such an enhancement, a WAF can discard, for example, HTTP packets containing harmful HTML code.

   (c) Personal firewalls: a personal firewall is a software components that is executed inside a personal computer so as to protect it from harmful communications.

3. Regarding the proxy capabilities of firewalls, they can be classified as follows:

   (a) Without proxy capabilities: in these types of firewalls, the packets passing all the firewall's filtering rules are directly sent to their destinations.

   (b) With proxy capabilities: in these types of firewalls, the packets passing all the firewall's filtering rules are dropped and new packets with the same intentions are constructed and sent instead.

Depending on the security level that a firewall is meant to provide, firewalls can be deployed in different architectures [5]. Following, the three most usual firewall architectures are described (see Figure 4):

1. Dual-homed host: in this type of architecture, the firewall is deployed inside a dual-homed host, a computer directly connected to the firewall's trusted and untrusted networks. While the dual-homed host offers services to hosts in both networks, the firewall blocks all the IP traffic going from one network to the other one. In case two hosts in different networks were to communicate, the firewall should be provided with proxy capabilities.

2. Screened host: in this type of architecture, the firewall resides in a router that connects the firewall's untrusted network with one particular host in the trusted network, known as the bastion host. In this way, all the traffic between the trusted and the untrusted network must go through two physically separated barriers, the firewall router and the bastion host. For most purposes, this type of architecture provides better security and usability than dual-homed architectures.

3. Screened subnet: in this type of architecture, the firewall functionality is divided into two routers placed in between the firewall's trusted and untrusted networks. The network formed by these routers is called perimeter network and is used as an extra layer of security, holding and isolating the bastion host. Thanks to this isolation, if an attacker evades the firewall router connected to the untrusted network and compromises the bastion host, the access to the trusted network is limited as there is another firewall router in between the bastion host and the trusted network.

As far as the evaluation of firewalls is concerned, the most widely used technique is penetration testing. Penetration testing evaluates the security of a system (in our case, a firewall) by simulating a series of attacks and analysing their effects. In the literature, some methodologies for firewall penetration testing have been proposed [6, 7]. However, the efficacy of penetration testing is rather limited in the firewall domain, as the attacks included in the testing process are usually very general and do not take into consideration the security requirements of the specific firewall being tested. Other type of techniques in which such requirements are considered are also available in the literature [8, 9]. The idea behind such techniques is to construct a formal model of the firewall security policy and verify its correctness using some mechanical approach, such as for example, a CASE tool [8].

3. Evasion of IDSs

In this section, a series of relevant works published in the field of IDS evasion are presented. For clarity purposes, the works are organized chronologically and following the first IDS classification presented in Section 2.1 (Network IDSs and Host IDSs).

3.1. Evasion of Network IDSs

Four different approaches aimed to evade Network IDSs (NIDSs) are presented following. Note that although the objective of all of the approaches is the same, the techniques that they use are not the same, as they try to exploit different parts of the complex architecture of a NIDS.

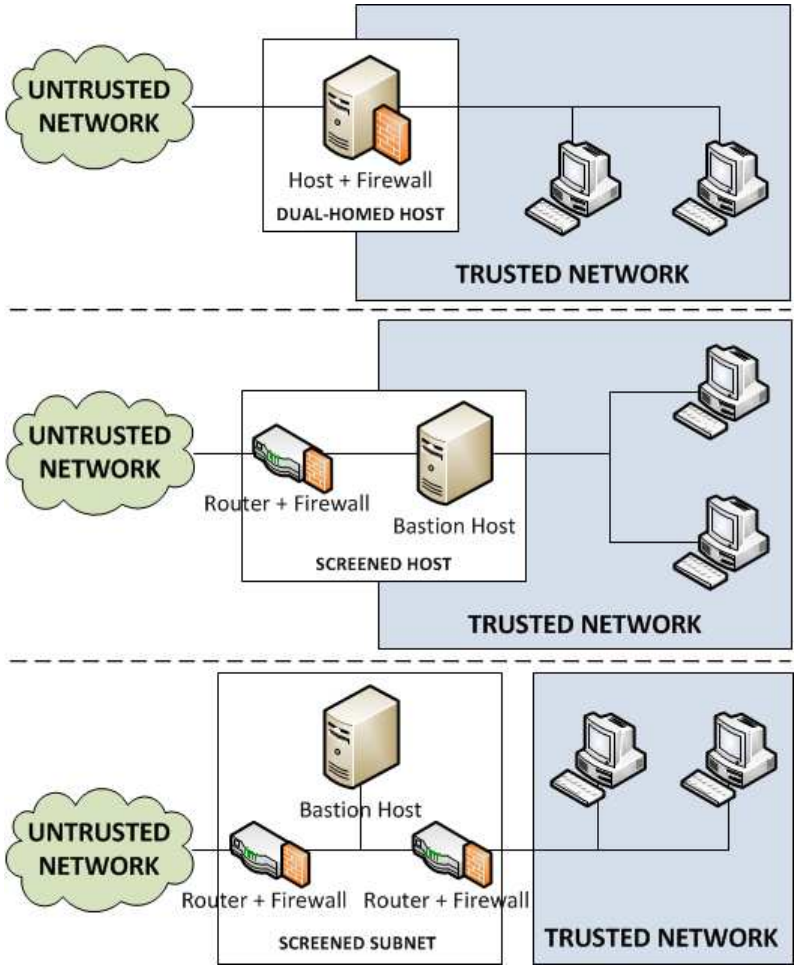

Figure 4. The three most usual firewall architectures

3.1.1. Ptacek and Newsham 1998

In 1998, Ptacek and Newsham proposed the first methods to evade IDSs, particularly NIDSs [10]. In their seminal paper, the authors highlighted that the ambiguities in the IP and TCP protocols, and the implementation problems present in many NIDS could lead different end systems to interpret the same traffic flow differently. In particular, the authors detected two types of situations in which the data flow processed by the NIDS was different from the one processed by the endpoint:

- Packet insertion: this situation occurs when the preprocessor of the IDS accepts a packet that the preprocessor of the endpoint does not (see Figure 5).
- Packet evasion: this situation occurs when the preprocessor of the IDS drops a packet that the preprocessor of the endpoint accepts.

The main packet characteristics that, according to [10], are prone to be exploited to succeed in evading NIDSs are presented following:

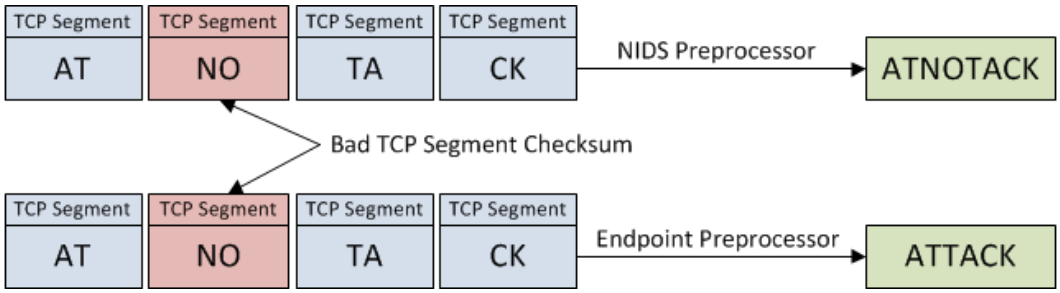

Figure 5. Example of packet insertion

- IP and TCP headers: an attacker can tamper with the IP and TCP headers of packets or make use of the ambiguity of IP and TCP options in order to achieve packet insertions and evasions. A typical example of IP header field that can be used for packet insertion is the TTL (Time To Live) field. Every time an IP packet is forwarded by an intermediate router or hop, the value of its TTL is decreased. Since IP implementations discard packets with a TTL value equal to or lower than 0, if an attacker knows the network architecture (i.e., the location of the NIDS and the location of the victim host), it can set a TTL value big enough to reach the NIDS but small enough not to reach the endpoint. An example of TCP header field which can also lead to packet insertion is the checksum field. A bad checksum value causes the packet to be dropped in most endpoint implementations; however, due to flaws in NIDS implementations, some of them do not verify the correctness of the checksum value. As shown in Figure 5, such a lack of verification enables an invalid packet to reach and deceive the detection engine of a NIDS. In the particular case presented in Figure 5, if the NIDS is signature-based and it is looking for the pattern "ATTACK" to generate an alarm, the insertion of an invalid packet with the payload "NO" prevents the detection engine from generating the alarm, which is obviously a security problem taking into account that the payload arriving at the endpoint is actually "ATTACK".

- TCP connections: maintaining information about open connections is crucial for a NIDS to properly handle packets (e.g., to reassemble IP fragments). Knowing when to record a particular connection state is essential to avoid memory overloads and consequently, DoS (Denial of Service) attacks. However, if the state of a connection is not properly tracked, problems like packet desynchronization can occur (e.g., setting the SYN flag in packets belonging to an already established connection). Another problem that NIDSs have to face is to determine when to stop recording the state of a connection. Normally, the end of a connection is triggered by an RST or FIN packet. However, an attacker may decide not to send such a packet in order to force a connection to remain open forever. An attacker may also manipulate an RST or FIN packet so that it arrives at the NIDS but not at the endpoint, causing the connection to be closed in the NIDS but remain open forever in the endpoint.

- Packet fragmentation, reassembly and overlapping: since IP packets can be fragmented and they can arrive at the destination out of order and duplicated, NIDSs should reorder, reassemble and handle duplicate packets before processing them. Due

to efficiency reasons, some NIDSs do not perform any of these operations and therefore, they are vulnerable to multiple types of attacks. Another typical problem arises when the last packet of an IP message (such a packet has the MF (More Fragments) flag unset) never arrives at the NIDS, which is bound to cause a memory flooding unless some special mechanism is implemented. Other more complex problems are related to the need of NIDS to mimic the behaviour of an endpoint. If the NIDS receives a packet with the same sequence number of a previous one, it must know which the policy of the endpoint is in order to substitute the previous or drop the new one. Regarding TCP, a more complicated problem occurs with the window size value, as it have to accept the same size that the endpoint does.

In order to defend NIDS against these evasive methods, several techniques have been proposed so far. Most of these techniques are based on the idea of modifying the traffic arriving at the NIDS so that it fulfils some specific format. In 2000, Handley et al. introduced the concept of traffic normalizers [11]. A traffic normalizer is a tool in charge of removing the possible ambiguities in the traffic arriving at the NIDS. Since some of the evasive techniques are based on packet fragmentation and reassembly, the state of each connection together with some meta-data must be stored and processed by the normalizer. In 2008, Vutukuru et al. proposed a more efficient normalizer using packet hashes and connection tables to provide an overview of the unacknowledged and out-of-order packets in each connection [12]. In 2004, Watson et. al proposed a system for generating well-formed TCP data from any TCP traffic [13]. The goal of such a system was to ensure that NIDSs and endpoints interpreted TCP traffic in the exact same way. The system was based on state machines, which were used to indicate the states that any TCP connection had to step in, modifying or dropping the packets that did not follow a valid state sequence.

Some other solutions that do not modify the traffic arriving at the NIDS have also been proposed. In 2002, Shankar and Paxon proposed a system in charge of informing the NIDS about the network topology and the interpretation policy of the endpoint being monitored [14]. With such information, the NIDS could adapt its configuration so as to mimic the behaviour of the endpoint. For example, Snort [1] adopts this technique in its IP processor (frag3). In 2006, Varguese et al. presented the idea of dividing an entire signature of the NIDS into single smaller pieces [15]. A fast path finds matches with any of these pieces, and if it detects any match of them with the packet being analysed, a slower path inspects it deeper. Finally, In 2009, Antichi et al. proposed the use of Bloom Filters [16] to perform signature matching over a set of unassembled packets [17]. This technique improves the efficiency of NIDSs and allows all intrusions to be detected. However, the number of false positives is also very large.

3.1.2. Vigna et al. 2004

In 2004, Vigna et al. presented a method to evaluate the response of different signature-based NIDSs against evasion attacks [18]. 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 provoke 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.

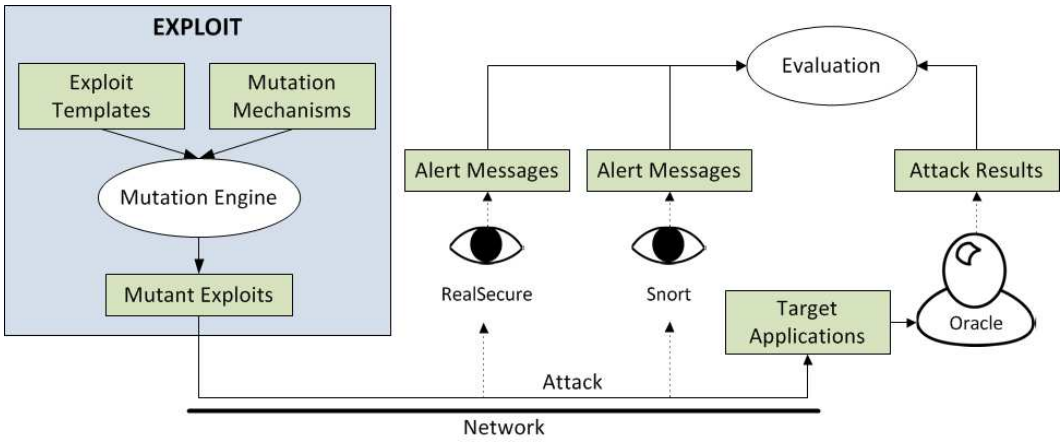

Figure 6. Framework designed to mutate exploits capable of evading Snort and RealSecure

The authors built a framework that, given a series of mutation mechanisms and a set of exploit templates, combined both sets to deterministically generate a set of mutant exploits. Then, these mutant exploits were analysed by the external oracle in order to verify that the changes were valid (such a verification was made by checking that the application of the exploits to the target applications were successful). Finally, the mutant exploits were presented to the analysed NIDSs, Snort and RealSecure, where it was verified whether the mutant exploits could actually evade the detection or not. Figure 6 shows the schema of the proposed framework.

It is important to remark that the set of mutations mechanisms included, amongst others, the mechanisms presented by Ptacek and Newsham in 1998 (see Section 3.1.1): packet fragmentation, protocol flow modifications, etc. They also 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.

3.1.3. Fogla et al. 2006

The main characteristic of a worm is the self-replicating capability among different victims. In particular, 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. However, polymorphic worms are not classed as normal behaviour and therefore, they cannot evade anomaly-based NIDS. In 2006, Fogla et al. [19] extended the idea of polymorphic worms and proposed the use of Polymorphic Blending Attacks (PBAs), a technique that allows changing a worm so that it can blend in with the normal behaviour of a network.

A PBA is composed of three parts: the attack vector, used to exploit the desired vulnerability in the target host; the attack body, encrypted with some simple reversible substitution algorithm; and the polymorphic decryptor, in charge of decrypting the malicious code and transferring the control to it. The main steps involved in the generation of a PBA are de-

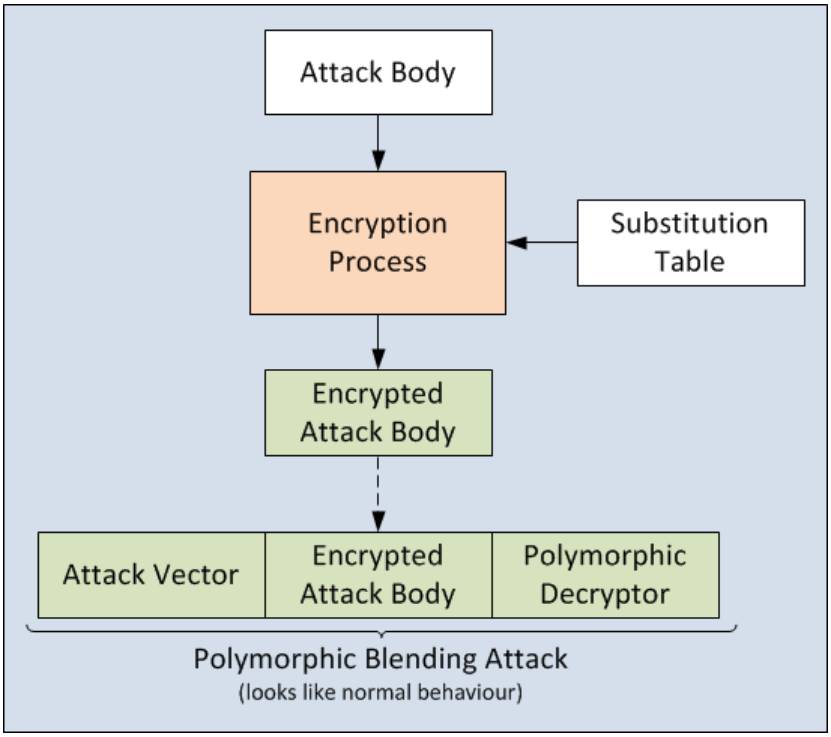

Figure 7. Generation of a Polymorphic Blending Attack (PBA)

scribed following (see Figure 7):

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 behaviour, guaranteeing that the statistical properties specified in the normal profile are satisfied (note that finding an optimal substitution table is a very complex task, according to [20]).

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 [20], 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 either a satisfiability (SAT) problem (if the NIDS had been represented as a FSA) or Integer Linear Programming (ILP) problem (if the NIDS had been represented as a sFSA), two types of problems for which polynomial-time algorithms already existed.

3.1.4. Pastrana et al. 2011

In [21], we presented a new approach to look for flaws and weaknesses in NIDS. As the behaviour of a NIDS is usually very complex (even obscure if its source code is proprietary), our approach tries to model it by means of Genetic Programming (GP). GP is a paradigm that evolves programs (also known as individuals) represented by trees, where intermediate nodes are functions and leafs are terminals. The evolution is performed by crossing and mutating the best individuals of the current population and repeating the process until a certain condition is held.

In the case at stake, the result of the GP evolution is a simple model that behave as similarly as possible to the original NIDS. Since the resulting model is simpler that the original NIDS, it can be used to finding evasions that otherwise would be very difficult to find. In the paper, evasions are performed over a NIDS built specifically for that purpose. Following, an overall description of the methodology used is presented (see Figure 8):

1. A controlled network infrastructure is used to generate system traces, both normal and abnormal ("1" in Figure 8).

2. Raw traffic along with its respective output are combined and processed in order to generate labelled traces that represent the relationships between the inputs and the outputs of the NIDS and therefore, its behaviour. These traces are exposed to data mining procedures in order to extract and select the most relevant features to model the NIDS by means of GP.

3. The labelled traces are inputted to GP so as to generate the models that will classify the network traffic very similarly to the original NIDS. In order to optimize the process, some GP parameters must be correctly defined, namely the mutation and crossover rates, the selection method and the most important one, the fitness function, in charge of deciding which of two individuals is better ("2" in Figure 8).

4. The resulting GP models are used to look for evasions. The way in which this search is performed is not defined. In future works, we will try to automatize the process ("3" in Figure 8).

5. If an evasion over the models is found, it is then tried over the original NIDS, evaluating so if it actually succeeds ("4" in Figure 8).

The main advantage of this methodology is that it can be adapted to any IDS. The only requirement that the IDS must fulfil is having the relationships between its inputs and its

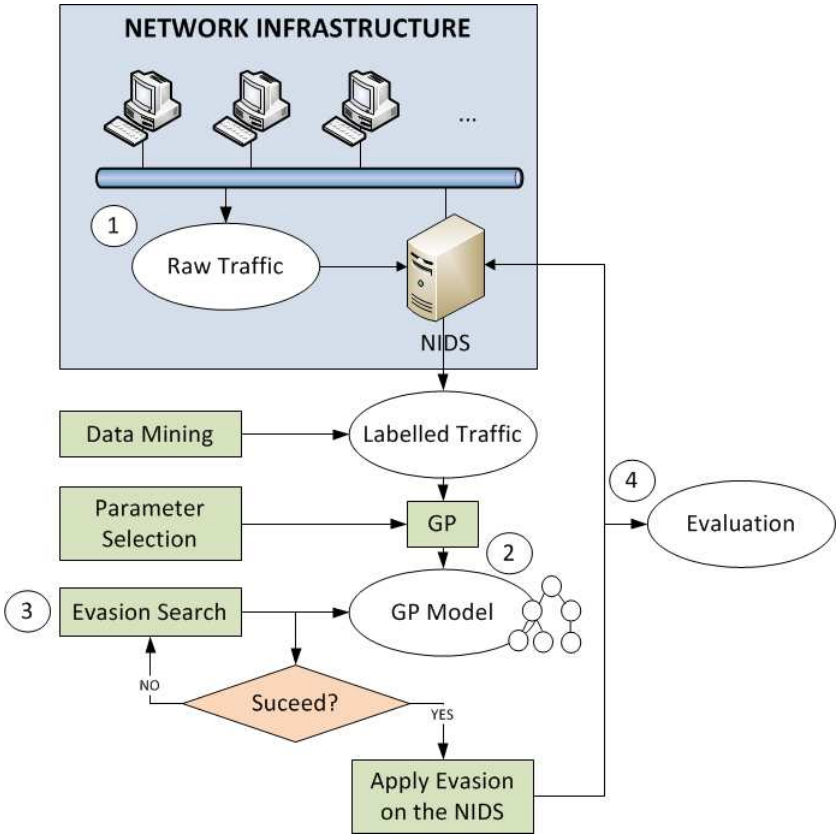

Figure 8. Methodology using Genetic Programming (GP) to search for vulnerabilities in NIDSs

outputs. Note that it is possible to use more than one technique (different from GP) to generate the models.

The paper also presents a proof of concept of the methodology. It uses a simple NIDS built specifically for that purpose and the dataset LBNL[1], which contains both normal traces and traces corresponding to a port scanning attack. Before using the methodology, the NIDS was able to detect most of the port scanning traces, but after using it, we succeeded in evading detection.

3.2. Evasion of Host IDS

Since Host IDSs (HIDSs) look for intrusions inside computer hosts, the evasion techniques applied to them are mainly focused on modifying system calls, memory registers, the stack, etc. The first evasion technique for HIDSs appeared in 2002 and since then, an important evolution process has occurred. Following, four relevant works illustrating such a process are presented.

[1] http://www.icir.org/enterprise-tracing/

3.2.1. Tan et al. 2002

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 [22]. In particular, they showed how to evade the Stide HIDS [23].

The 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. For the sake of illustration, the paper presented two attacks against two Linux programs, namely *passwd* and *traceroute*. First of all, the authors showed how Stide detected the attacks when they were executed in a victim host. Then, they explained how to modify the attacks so as to evade detection. Finally, they executed the attacks again and verified that they actually bypassed detection.

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. For instance, in the case of the attack against the *passwd* program, the original attack was to execute the sequence of system calls *setuid-setgid*. The authors realized that the same attack could be accomplished with the sequence *chmod-exit*, whose length is the same. However, as Stide's database of normal behaviour contained the sequence *chmod-utime-close-munmap*, there was a chance to combine such a sequence with the one of the attack (i.e., *chmod-exit*) and succeed with the attack. In particular, the authors realized that Stide did not take into account the argument of system calls and thus, they tried their attack with the sequence *chmod-utime-close-munmap-exit*, making sure that the arguments of the *munmap* call were tuned in such a way that the state of the system was not modified. With such a sequence, if Stide used a detection window size of two, no alarms would be generated as no malicious sequence is equal to or smaller than two.

Finally, the authors proposed to automatically look for all the allowed sequences that, if correctly tuned, do nothing to the system and make malicious sequences larger. They presented some experiments to automatically generate such sequences and exposed how they were inserted into malicious sequences of system calls traces without disrupting the wicked intention.

3.2.2. Wagner et al. 2002

In 2002, Wagner et al. presented six different ways to evade anomaly-based HIDSs [24] (note that they did not tackle evading signature-based IDSs because they considered such a task a "child's play"). Before describing the six ways to evade anomaly-based HIDSs, two assumptions need to be established:

1. The behaviour of the HIDS is known by the attacker, including the database of normal behaviours. This assumption is straightforward in the case of open source IDS, as one can use the public training algorithm with real data in order to obtain the model of normal behaviour. In the case of proprietary systems, some reverse engineering approach may be used (e.g., the one explained in Section 3.1.4).

2. The attacker can gain the control of an application without being detected. This can be due to flaws in the implementation or by forcing the application to generate wrong results.

Once the assumptions were established, the authors described the six methods to evade detection:

1. Slip under the radar: as anomaly-based IDSs look for intrusion evidences by examining system call sequences, the authors state that an attack can remain undetected if no system calls are made. This type of attack can be achieved by forcing an application to compute incorrect results. However, the damage that such an attack can cause is quite limited.

2. Be patient: the authors stated the limitation that at some point in time a HIDS may accept a malicious sequence of system calls, an attacker can wait this time to launch her attack. However, this scenario poses an important problem: after the malicious sequence is executed, the returned address of the application will produce abnormal trace and thus, an alarm may be generated. To solve such a problem, the authors propose to make the application crash, pretending the abnormal trace to be a bug rather than an attack.

3. Be patient, but make your own luck: this method is an improvement of the previous one. In it, the attacker replaces the entire application with a clone manipulated in such a way that the returned address produces no abnormal trace.

4. Replace system call parameters: changing the parameters of a valid system call may allow an attack to be executed undetected. This situation happened because most of the HIDS in 2002 did not examine these parameters when searching for anomalies.

5. Insert no-ops: the idea behind this method is the same as the one explained in Section 3.2.1. Since some HIDSs take into account the length of anomalous sequences to generate an alarm, detection may be evaded if operations without an action in the system are inserted in the sequences.

6. Generate equivalent attacks: generating variations of a particular attack (e.g., by reordering the system calls or replacing some of them with equivalent ones) may help the attack to bypass detection.

3.2.3. Kruegel et al. 2005

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.

In [25], Kruegel et al. 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.

To conclude, it is important to remark that the technique proposed by Kruegel et al. was innovative because no previous work had previously proposed to analyse and modify binary code so as to evade detection.

3.2.4. Kayacik et al. 2011

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 [26]. In order to generate such mutations, Genetic Programming (GP) was used (see Section 3.1.4 for more information about GP). 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.

In order to evaluate the attack success, they studied an attack composed of a sequence of 3 defined system calls and arguments. Concretely, as they wanted to add a new user with root privileges, this sequence was:

1. open('/etc/passwd')
2. write('toor::0:0:root:/root:/bin/bash')
3. close('/etc/passwd')

If this sequence was found in the set of system calls of the individual, its fitness was increased.

In their experiments, the authors selected four vulnerable applications, ran them under normal circumstances, recorded their system calls, selected the twenty most frequent ones and inputted them to the GP algorithm. As for the anomaly and delay rates used in the experiments, they were generated by five different IDSs: the Stide HIDS, the Process Homeostasis (PH), the Process Homeostasis with schema mask (PHsm), a Markov model-based detector and an auto-associative neural network. 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 behaviour of the IDS. The results of such a comparison show that, although using the white-box approach produces 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 this 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 a IDS but wants to evade another one.

4. Evasion of Firewalls

Nowadays, the only general mechanism to attain firewall evasion is tunneling, a technique that allows encapsulating one protocol into another. By means of tunneling, an adversary can bypass a firewall and execute a protocol that the firewall blocks (e.g., the BitTorrent protocol) by wrapping it into a protocol that the firewall accepts (e.g., the HTTP protocol). In order for the firewall to detect such an evasion, it must analyze the payload of the application layer and make sure that its data corresponds to regular data of the specified protocol. Since such an analysis is time-consuming, many firewalls fail to perform it and thus, tunneling becomes an effective mechanism to evade firewalls.

Apart from tunneling, the only way to currently evade a firewall is to find and exploit particular vulnerabilities. Firewall vulnerabilities can be caused by different types of errors, being the most common those listed following [27]:

1. Validation error: this error happens when environmental data (e.g., a source IP address) is used without verifying its correctness.
2. Authorization error: this error occurs when a protected operation is invoked without properly verifying the authority of the invoker.
3. Serialization error: this error happens when system operations with asynchronous behaviours permit an adversary to attack the protected system.
4. Aliasing error: this error occurs when an object that has been already validated is modified unexpectedly and hence, its state can no longer be assumed correct.
5. Boundary checking error: this error happens when some boundary or constraint in the firewall (e.g., the maximum size of a buffer) is not checked.
6. Domain error: this error occurs when there exists a security hole that permits information leakage between two protection environments.
7. Design error: this error happens when the origin of a firewall's vulnerability can be traced back to the design phase of the firewall.

Depending on the vulnerabilities that a particular firewall has, an attacker may achieve different types of effects, being the most common ones summarized next:

1. Execution of code: illegitimate code is executed in the firewall or in some device in the protected network.
2. Change of target resource: some resource in the firewall (e.g., a filtering rule table) or in the protected network (e.g., a host) is illegitimately modified.
3. Access to target resource: an attacker gains illegitimate access to some resource in the firewall or in the protected network.
4. Denial of service: an attacker disrupts some service offered by the firewall (e.g., the packet forwarding service) or by some device in the protected network (e.g., an FTP service).

For the sake of illustration, three vulnerabilities found repeatedly in multiple firewalls are briefly described following (note that each of the three vulnerabilities results in an illegitimate access to some resource in the firewall or in the protected network):

1. Some firewalls fail to discard external packets claiming to come from the internal network. This vulnerability is caused by an origin validation error. For this error to be fixed, firewalls must check the interface through which a packet arrives; otherwise, determining if a packet has a spoofed source IP address becomes impossible.

2. The firewall included in some Windows systems (e.g., Windows 2000 Server) accepts all the incoming traffic targeted at the TCP/UDP port 88 (this port is reserved for Kerberos [28], a security protocol that some Windows systems use to authenticate IPsec sessions). This vulnerability is caused by a design error and its solution requires firewalls not to assume that the traffic passing through well-known ports is legitimate and valid.

3. Some firewalls fail to discard external packets claiming to come from an IANA-reserved or private address (e.g., 127.x.x.x). This vulnerability is also caused by a design error and fixing it requires firewalls not to assume that the IP addresses specified in a packet belong to some specific ranges.

To conclude this section, it is worth to remark that an interesting work describing an attempt to evade the so-called "Great Firewall of China" is presented in [29]. In such a work, the authors try to achieve two goals: gaining access to a Web site blocked by the firewall and provoking a denial of service in the firewall's packet forwarding service.

5. Conclusions

In this chapter, we have presented and analysed some of the most relevant methods to evade IDSs and firewalls. In the case of IDSs, the analysis shows that since the publication of the first technical report in 1998 showing how to evade NIDSs, there has been a worrying evolution of the techniques and methods aimed to evade these systems. For instance, nowadays attackers can use sophisticated artificial intelligence techniques along with high performance systems to evade detection.

In the case of firewalls, the analysis shows that there exists no general methodology to bypass the different types of firewalls. Despite that, we have presented the most common vulnerabilities that an attacker can exploit so as to evade a firewall. Furthermore, by way of illustration, we have introduced a few real vulnerabilities that can be found in some state-of-the-art firewalls.

It is clear that as technology and security barriers become more complex and sophisticated, the knowledge and techniques used to find security vulnerabilities to bypass detection evolve. Consequently, in line with the improvement of security countermeasures, and in order to have advantage over attackers, it is essential to explore and analyse the different mechanisms that allow these barriers to be evaded. In the particular case of SIEMs, due to its great complexity and heterogeneous architecture, it is crucial to secure each of the countermeasures taking part in the detection process.

Acknowledgements

This work has been partially funded by the Regional Government of Madrid, Spain, under the project EVADIR: A Methodology for Evasion Attacks on Network Intrusion Detection Systems.

References

[1] Roesch, M. (1999) Snort: Lightweight intrusion detection for networks. *Proceedings of the 13th Systems Administration Conference*, Seattle, WA, USA, November, pp. 229–238. USENIX.

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

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

[4] Bhagchandka, D. (2003) Classification of firewalls and proxies. Technical report. University of Texas, Department of Computer Science, Texas, USA.

[5] Abramowitz, M. and Stegun, I. A. (1995) Firewall design. In Russel, D. (ed.), *Building Internet Firewalls*. O'Reilly and Associates, Inc., San Francisco, CA, USA.

[6] Moyer, P. R. and Schultz", E. (1996) A systematic methodology for firewall penetration testing. *Network Security*, **1996**, 11–18.

[7] Haeni, R. E. (1997) Firewall penetration testing. Technical report. The George Washington University Cyberspace Policy Institute, Washington, DC, USA.

[8] Jürjens, J. and Wimmel, G. (2001) Specification-based testing of firewalls. *Perspectives of System Informatics*, Lecture Notes in Computer Science, **2244**, pp. 308–316. Springer, Novosibirsk, Russia.

[9] Senn, D., Basin, D., and Caronni, G. (2005) Firewall conformance testing. *Testing of Communicating Systems*, Lecture Notes in Computer Science, **3502**, pp. 348–348. springer, Montreal,Canada.

[10] Ptacek, T. H. and Newsham, T. N. (1998) Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report. Secure Networks, Inc., Syracuse, NY, USA.

[11] Handley, M., Kreibich, C., and Paxson, V. (2000) Network intrusion detection: Evasion, traffic normalization and end-to-end protocol semantics. *USENIX Security Symposium*, Denver, Colorado, USA, August, pp. 115–131. USENIX.

[12] Vutukuru, M., Balakrishnan, H., and Paxson, V. (2008) Efficient and robust tcp stream normalization. *IEEE Symposium on Security and Privacy*, Oakland, California, USA, May, pp. 96–110. IEEE.

[13] 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**, 261–273.

[14] Shankar, U. (2003) Active mapping: Resisting nids evasion without altering traffic. *Security and Privacy*, Oakland, California, USA, may, pp. 44–61. IEEE.

[15] Varghese, G., Fingerhut, J. A., and Bonomi, F. (2006) Detecting evasion attacks at high speeds without reassembly. *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, Pisa, Italy, September, pp. 327–338. ACM.

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

[17] 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.

[18] Vigna, G., Robertson, W., and Balzarotti, D. (2004) Testing network-based intrusion detection signatures using mutant exploits. *Proceedings of the 11th ACM Conference on Computer and Communications Security*, Washington, DC, USA, October 21. ACM.

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

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

[21] Pastrana, S., Orfila, A., and Ribagorda, A. (2011) A functional framework to evade network ids. *Hawaii International Conference on System Sciences*, Koloa,Hawaii,USA, January, pp. 1–10. IEEE.

[22] Tan, K., Killourhy, K., and Maxion, R. (2002) Undermining an anomaly-based intrusion detection system using common exploits. *Proceedings of the 5th International Conference on Recent Advances in Intrusion Detection*, Zurich, Switzerland, October, pp. 54–73. Springer-Verlag.

[23] Forrest, S., Hofmeyr, S., Somayaji, A., and Longstaff, T. (1996) A sense of self for unix processes. *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May, pp. 120–128. IEEE.

[24] Wagner, D. and Soto, P. (2002) Mimicry attacks on host-based intrusion detection systems. *Proceedings of the 9th ACM Conference on Computer and Communications Security*, Washington, DC, USA, November, pp. 255–264. ACM.

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

[26] 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**, 243–266.

[27] Kamara, S., Fahmy, S., Schultz, E., Kerschbaum, F., and Frantzen, M. (2003) Analysis of vulnerabilities in internet firewalls. *Computers and Security*, **22**, 214–232.

[28] Neuman, B. and Ts'o, T. (1994) Kerberos: an authentication service for computer networks. *IEEE Communications Magazine*, **32**, 33–38.

[29] Clayton, R., Murdoch, S. J., and Watson, R. N. M. (2006) Ignoring the great firewall of china. In Danezis, G. and Golle, P. (eds.), *Proceedings of the Sixth Workshop on Privacy Enhancing Technologies (PET 2006)*, Cambridge, UK, June, pp. 20–35. Springer.