Linköping University | Department of Computer and Information Science
Bachelor thesis, 16 ECTS | Datateknik
202017 | LIU-IDA/LITH-EX-G--2017/085--SE

Network Intrusion and Detection

– An evaluation of SNORT

Nätverksintrång och detektion - En utvärdering av SNORT

Fleming J. Theodor
Wilander Hjalmar

Supervisor : Marcus Bendtsen
Examiner : Nahid Shahmehri

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Students in the 5 year Information Technology program complete a semester-long software development project during their sixth semester (third year). The project is completed in mid-sized groups, and the students implement a mobile application intended to be used in a multi-actor setting, currently a search and rescue scenario. In parallel they study several topics relevant to the technical and ethical considerations in the project. The project culminates by demonstrating a working product and a written report documenting the results of the practical development process including requirements elicitation. During the final stage of the semester, students create small groups and specialise in one topic, resulting in a bachelor thesis. The current report represents the results obtained during this specialisation work. Hence, the thesis should be viewed as part of a larger body of work required to pass the semester, including the conditions and requirements for a bachelor thesis.

Abstract

Network security has become a vital part for computer networks to ensure that they operate as expected. With many of today's services relying on networks it is of great importance that the usage of networks are not being compromised. One way to increase the security of a computer network is to implement a Network Intrusion Detection System (NIDS). This system monitors the traffic sent to, from and within the network. This study investigates how a NIDS called SNORT with different configurations handles common network attacks. The knowledge of how SNORT managed the attacks is used to evaluate and indicate the vulnerability of different SNORT configurations. Different approaches on both how to bypass SNORT and how to detect attacks are described both theoretically, and practically with experiments. This study concludes that a carefully prepared configuration is the factor for SNORT to perform well in network intrusion detection.

Acknowledgments

We would like to thank Marcus Bendtsen for his time and the way he introduced this subject. He inspired us to embark this study with attitude of great curiosity.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Today computer networks, both local and the Internet, is a given for billions of people around the world. With 40% of the world's population having access to the Internet in 2015, a percentage that is increasing every day, tons of information is spread over the Internet [13]. There are also parts of society who relies on the service of the Internet to function as intended such as banks, shops, and communication systems. The Internet comes with huge opportunities and has made a lot of daily tasks more flexible but it does also bring misfortune: stolen information, missing data and several other violations performed by hackers. Hackers are those who seek to exploit weaknesses in computer networks. Therefore it is crucial to protect networks from hackers trying to misuse valuable information and services.

An attempt to break or misuse a system is called an "intrusion". A software application with the purpose of detecting these intrusions is called Intrusion Detection System (IDS). IDSs come in different varieties and this study's focus is on network based IDS (NIDS).A NIDS is placed on a strategic point, or points, within the network to detect malicious traffic from or to all devices on that network. Meanwhile, a host based IDS (HIDS) is placed on a single host [9]. In this study a popular NIDS software called SNORT is tested by monitoring an Apache server that is attacked by common network attacks [15]. SNORT comes without preferences to what it should detect and how, so first SNORT is configured to run with configurations made by the SNORT community. Then SNORT is configured with an extended configuration available to registered SNORT users. And lastly, SNORT is tested with a customized configuration to see if the undetected attacks in this study can be detected.

1.2 Aim

The objective of this study is to evaluate how and if a NIDS called SNORT can be configured to detect malicious network attacks. And, what is required to increase the performance of SNORT considering detection. Since SNORT is an open source program that highly relies on the expertise of the user the intention is to determine how good SNORT is at detecting the intrusion attacks depending on which configurations have been made to the program. First SNORT is tested with configurations available directly at download. Then the SNORT configurations are extended by configurations available to registered SNORT users and tested against the same attacks. Finally, SNORT is tested with a configuration made by the authors of this report. This configuration is tested with the attacks that avoided detection of the previous configurations. This study will also give the reader insight on different common network attacks, both how they are designed and how they are mitigated. It will cover the three main classes of common network attacks that is used in this study, and it will present different types of attacks in these classes as well. Finally weak spots of SNORT is pointed out and what to improve, to encourage development of NIDSs in general. With more systems relying on a network connection the emphasis of a system that exposes incoming attacks increase. Thus, the importance of evaluating current NIDSs and developing NIDSs that detects more attacks increase.

1.3 Research questions

1. Which of the network attacks in this study are detected by SNORT?
2. Which configuration detects which attack?
3. Can SNORT be modified to detect all the executed attacks?
4. Can the attacks be modified to not get detected by SNORT?

1.4 Delimitations

There are many commercial NIDSs available on the market. This study is limited to testing the reliability of an open source NIDS called SNORT. The study limits the executed network attacks to four common and basic network attacks described in Section 2.2.

2 Theory

This chapter explains how general IDSs work and account for the different ways NIDSs detect unauthorized network usage. It also explains how different network attacks work and how they may be modified to evade NIDSs.

2.1 Intrusion detection systems

Intrusion detection is a security technology that aims to determine if a computer system is being attacked. There are a lot of IDSs with diverse designs searching for different types of intrusions; one system might examine only malicious HTTP requests to detect attacks against web servers, while another may consider IP spoofing to monitor dynamic routing protocols. One thing they all have in common is the general definition of "intrusion" as an unauthorized usage, or misuse, of a computer system. IDSs are not only used to detect attacks, they can also contribute with forensic information that might identify the source of an attack. This makes hackers liable for their actions as well as discourages future attacks [21].

One of the major aspects with NIDSs compared with HIDSs is that the only data NIDSs are receiving is raw network traffic while HIDSs are kept updated by the operating system. This allows NIDSs to be good at distinguishing attacks that concern low level manipulations of the network while protecting multiple devices on the same network. One downside that comes with only watching the network traffic is that NIDSs are not fully aware of what is transpiring on a computer system. Hackers can use this knowledge to their advantage by misleading a NIDS to make it believe the computer system accepts one packet but in reality drops it. This is explained in Section 2.3.

A device that is connected to a network generally only inspects packets that are addressed to them. The way NIDSs collect data is by copying packets that are transmitted on the network and inspecting the content, regardless of the packets' destination. There is no distinct way to transmit a packet without it being copied by the NIDS, this makes it very challenging to bypass a NIDS [21].

The basics of an IDS

Since there are many different IDSs installed over the world with many different designs, a standard model has been established called the common intrusion detection framework

(CIDF). This model has been developed for allowing different IDSs to interoperate [14]. According to CIDF an IDS is characterized by four components, which are: event generators (E-boxes), analysis engines (A-boxes), storage mechanisms (D-boxes) and countermeasures (C-boxes). These components can be a software package in and of itself or part of a larger system [26]. The objective of the E-box is to let the rest of the system know what events that are occurring. It can be complex events or it can be a low level network protocol instance and does not need to be evidence of an intrusion. The E-box can be seen as the eyes and ears of the IDS; without the input from the E-box the IDS is blind and does not know what to do since it does not receive any data to process. The A-box analyzes the data that the E-box has collected. The data that the E- and A-box produces is stored in the D-box, making it available at a later occasion. If an attack is detected, the C-box can prevent further attacks from occurring. It can do so by executing different actions such as killing processes, shutting down TCP connections, and by modifying router filter lists. If the C-box itself does not have these abilities it can be connected to another program that can accomplish a similar reaction [21].

Different approaches for detection and analyze

The foundation for intrusion detection is that suspicious traffic diverges from normal traffic and can therefore be distinguishable. An IDS can use different techniques, to detect these abnormal activities, that can be divided into three sections: misuse detection, anomaly detection and specification-based detection. Misuse detection uses well known data and patterns from prior attacks to determine if the traffic or operations are malicious. It is considered a good approach for established attacks, but when it comes to new undiscovered or innovative threats it is not efficient. This approach is also sensitive for mistakes when defining these intrusive exploits, if an exploit pattern is not defined correctly the exploit may not be detected [9].

One form of misuse detection is called pattern matching, a common technique used by IDSs. This technique models attack patterns that it uses to match with the content of the packet, the packet header or both. If a match occurs there is a high risk that there is an attack taking place [9].

Another approach of misuse detection is rule-based techniques where the IDS has a set of determined rules and laws. These rules can for example be defined sequences of operations that occur off-hours or activity where someone tries to log in with the wrong password more than three times. If a rule is met the IDS alerts that the system may be exploited [24] [9].

Anomaly detection does the opposite of misuse, building models for how normal activity on a system should behave. It follows one rule, all irregular activities are potential threats. A typical anomaly detection consists of four components: data collection, normal system profile, anomaly detection and response. Normal user activities or traffic data are obtained and stored by the data collection component. Normal system profiles are then created from the data collected by the data collection component. The anomaly detection component compares the normal system profile with the current system's activities to determine how much they diverge and which of these activities should be identified as malicious.

Anomaly detection's main feature is that it can find novel attacks, which is misuse detection's greatest limitation. One of the downsides with anomaly detection is that it in general has a very high false alarm ratio as a result of considering all abnormal activities as possible attacks. Another fact to keep in mind is that the normal system profile is based on data collected over a period of normal operations, attacks that goes undetected meanwhile are prone to be acknowledge as normal activity [9].

Specification-based Detection (SBD) uses behavioral specifications, created by experts, to detect attacks by verifying that the system's executions are done as expected. If the specifications are not met the system might be exploited. If any event or execution sequence in the system is not following the determined specification there is a risk that an attack is taking place. When creating the specifications for SBD it is the experts' competence that determines the system's operating limit. One of the advantages with SBD is that, in theory, it is able to identify unseen attacks that are inside the computer system, but have not yet been exploited.

Although specifications of programs are formed using inductive logic programming, a subfield of machine learning, it still remains complicated to keep track of the correct specification for every program on the monitored system. An example of how SBD could detect a buffer overflow attack is a situation where an attacker tries to overflow a buffer with over sized arguments, the system's specifications are then not fulfilled since the size of the argument is too big. The buffer overflow attack also creates an unexpected execution of a shell and this does not follow the system's specification as well, therefore SBD detects the attack [28] [9].

SNORT

SNORT is an open source network intrusion prevention system that can be configured as a NIDS [6]. SNORT uses misuse detection, that is mentioned in Section 2.1, and the SNORT administrator can implement a set of rules for what SNORT should look for when determining if the incoming packets are malicious or not. Users can write their own rules as well as download rules, written by the SNORT community for example. When packets arrive through the computers network interface controller they first get decoded by a packet decoder. The packet decoder analyzes which protocol is being used for the packet and if the packet deviates from the rules of the given protocol. After the packets have been decoded they are sent to one or more preprocessors, given that they have been implemented by the SNORT user. Preprocessors are plug-ins that are used by SNORT to, in different ways, parse the packets coming from the packet decoder. This could be used for TCP fragmentation reassembly for example. The packets are then sent to the detection engine of SNORT where they are compared to the rules implemented by the administrator. SNORT tries to determine which rules to compare the incoming data against by checking the protocol for the packet and what port it originates from [3]. Then SNORT uses pattern matching to find a unique rule that match a pattern in the packet header or content. The last part in the SNORT cycle, after a rule has been matched with the data, is alerting the user that a rule has been triggered. The alerts and the packets are logged in a file by SNORT. An example of how a rule could be designed:

$$alert\ tcp\ 111.111.1.1\ any\ -> \ 222.222.2.2\ 80\ (msg : \text{“}incoming\ TCP\ packet\text{”};\ sid : 240312) \tag{2.1}$$

This rule alerts if the IP address 111.111.1.1 sends TCP packets from any port to the IP address 222.222.2.2 on port 80, with the message “incoming TCP packet”. The rule option “msg” allows the user to write any message that should be displayed when this rule is met. The “sid” number is the ID of the rule and has to be unique. Another general rule option is “classtype” that is used to categorize more common attacks. Depending on what class a rule contains the priority of the alert varies. Another option is “rev”, which is used to identify improved SNORT rules and allows the rule to be refined and replaced with updated information [2].

2.2 Network attacks

Network attacks impedes the cornerstones of network security, which are: confidentiality, integrity and availability. They can vary from scanning computer ports to taking full control over a computer. Two important concepts when working with network attacks are snooping and spoofing. Snooping is when an entity is browsing through files or system information, or listening to communication, that it is not entitled to. Port scanning is a form of snooping where the attacker is looking for open ports on the victims computer, perhaps to find a way to break into it. This attack is explained in Section 2.2. Spoofing on the other hand is a form of impersonation by one entity of another entity. The victim entity is in trust that it knows who it is communicating with, when in fact someone else is interacting with the victim. This is further explained in 2.2. Spoofing can be used to pretend that data is coming from a valid IP address when it is in fact the attacker who is sending malicious data [5].

Port scanning

Port scanning is a technique used to search for open ports or other available services and to decide if these ports/services are vulnerable to exploits. It is useful to detect port scans as they are often executed before an attack that can cause harm is carried out. Attackers can use port scanning to gain valuable information about the targeted computer, such as what ports are open for connection, what operating system the computer is running, what types of packet filters/firewalls are used and more [18]. A program called Nmap implements different port scanning techniques, where the default and most popular technique is TCP SYN scan. This method for port scanning is to only establish half open TCP connections to the target by sending only SYN segments to different ports on the target machine as if you wanted to establish a real connection. If the port is open and listening to possible connections the targeted computer will reply with a SYN/ACK message and if the port is not listening for connections the attacker will receive a RST (reset) flag. If no response is received after several retransmissions the port is marked as filtered, which means that the port could be open but something, like a firewall, is preventing the probes from reaching the port. When receiving a reply from the target the attacking host breaks the connection, because it has already received the information it needs.

Another technique for port scanning is when the scanning host tries to establish full TCP connections on chosen ports on the target machine. While a port on the target machine is listening a connection will be established, otherwise the target will deny the attempted connection. The main advantage of the first approach is that if the target logs the TCP packets, it is more likely to log full TCP connections than SYN connection attempts. That would cause target logging mechanisms not to detect the second approach. Both these techniques are implemented in Nmap, which is used for port scanning in this study [19] [29].

Denial of Service and Distributed Denial of Service

Denial of Service (DoS) attacks aim to disable or downgrade network services by exhausting resources on a network or host. The resources to be exhausted may be network bandwidth, routers' packet forwarding capacities, memory and computing power on servers or operating systems data structures. This attack can be used to shutdown a website by opening a TCP connection to the web server and overload it with meaningless TCP packets. There have been cases where DoS attackers bring down online businesses to later offer protection in exchange for money. It is not always possible for an attacker to overload the target's resources only using a single computer because of the attacker's limited computational and network resources. Although, an attacker could overload the target by initiating an attack using several computers to gain more resources, called a Distributed Denial of Service (DDoS) attack. A DDoS attack consists of multiple DoS attacks launched from a large number of hosts on the network. These hosts, that may not be aware that they are being used or controlled by an attacker, are called zombies or bots. The fact that DDoS attacks are well organized and can send traffic from many different hosts can cause them to bring down very powerful servers that have fast connections. DDoS attacks are more complex and harder to prevent than DoS attacks. Not only because it is hard to distinguish between the traffic generated by attacking computers and the traffic generated by normal users, but also because many hosts can generate more attacking traffic than one host. NIDSs are vulnerable to DoS and DDoS attacks since they are by nature "fail open", meaning that the NIDS does not kill the network connection even if the NIDS detects an attack [10] [9].

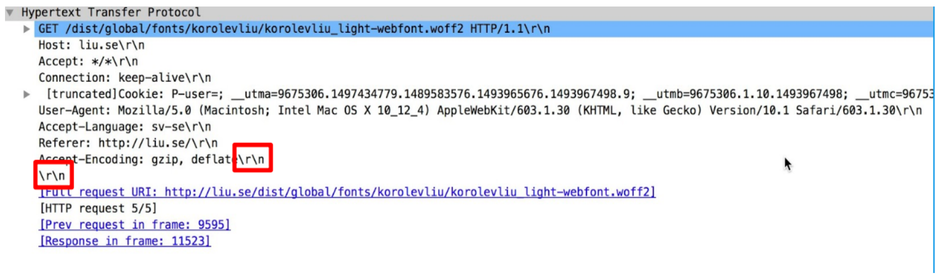
TCP SYN flood

TCP SYN flood is a basic DoS attack. In order for a client to achieve a TCP connection with a server both have to accomplish a three-way handshake. The three-way handshake consists of the client sending a requesting SYN message to the server, which replies with a SYN/ACK message acknowledging the client that the SYN message has arrived. To complete the connection establishment the client responds with an ACK to the server. When the connection is established service specific data can be exchanged between the two of them. The TCP SYN flood attack aims to generate half-open TCP connections by spoofing source IPs in SYN messages or ignoring SYN/ACKs so the server never gets the final ACK message, therefore leaving the connection half-open. The server needs to allocate memory to store half-open connections while waiting for the final ACK to arrive. This memory is often limited which leads to the memory space being filled and no more connections can be established, preventing legitimate requests from being accomplished. Half-open connections eventually expire due to timeouts, but never the less, zombies can send spoofed TCP SYN messages at a much higher rate than the expiration rate [10] [9].

Slowloris

A HTTP GET request utilizes CRLF (carriage return linefeed) to indicate a line break or, when used two times in a row as show in Figure 2.1, a blank line which indicates the completion of a request. An attacker can take advantage of this procedure by not sending the finishing blank line resulting in an incomplete HTTP request [8] [17]. Slowloris is a type of DoS attack that feeds of these incomplete HTTP requests. The attacker sends incomplete HTTP requests to a server which makes the server wait for the request to be completed before breaking the connection. If the server makes use of timeout counters, to break a connection that is taking too long, Slowloris sends another request before the timeout has reached the end to keep the connection alive. Slowloris sends enough HTTP requests to hold all connections to a web server. It sends these requests at a long term and regular interval to keep the connections occupied, resulting in a DoS [17]. A huge difference to other DoS attacks is that while other DoS attacks relies on high bandwidth by flooding the network to the targeted machine, Slowloris can run with low bandwidth utilization and on a single computer. This is because Slowloris only affects the targeted web server by sending only a few hundred requests at a regular and long term interval. So while the web server is inaccessible, other network services on the targeted machine stays intact.

Slowloris is not a TCP DoS attack, since it establishes full TCP connections to the server. However it is similar to a SYN flood attack over the HTTP protocol, as the HTTP requests are only partial. There are also some stealth features with Slowloris. If the target server uses log files for error messages these will not be updated until after the request is completed, that is, after the attack is completed. Also Slowloris attacks are not commonly detected by NIDSs since the attack does not send any malformed requests. The fact that Slowloris only affects the targeted web server also makes it less likely to be detected by the victim [11].



```
▼ Hypertext Transfer Protocol
  ▶ GET /dist/global/fonts/korolevliu/korolevliu_light-webfont.woff2 HTTP/1.1\r\n
    Host: liu.se\r\n
    Accept: */*\r\n
    Connection: keep-alive\r\n
  ▶  [truncated]Cookie: P-user=; __utma=9675306.1497434779.1489583576.1493965676.1493967498.9; __utmb=9675306.1.10.1493967498; __utmc=96753
    User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_4) AppleWebKit/603.1.30 (KHTML, like Gecko) Version/10.1 Safari/603.1.30\r\n
    Accept-Language: sv-se\r\n
    Referer: http://liu.se/\r\n
    Accept-Encoding: gzip, deflate\r\n
    \r\n
    [Full request URI: http://liu.se/dist/global/fonts/korolevliu/korolevliu_light-webfont.woff2]
    [HTTP request 5/5]
    [Prev request in frame: 9595]
    [Response in frame: 11523]
```

Figure 2.1: HTTP request displaying the CRLF (here, in Wireshark, represented as \r\n)

ICMP smurf flood

Internet Control Message Protocol (ICMP) is a protocol used for gateway or destination hosts to communicate with the source host. ICMP uses the basic support of the IP as it is an integral part of the IP and must be implemented by every IP module. ICMP messages are sent via the IP header. The purpose of ICMP is to provide feedback about problems in the communication environment, or to determine if a computer connected to the network is responding. A source host sends an ICMP echo request to the destination host, which responds with an ICMP echo reply [20]. An attacker can send ICMP requests to many different computers using the victims IP as the source address of these ICMP messages. These computers then send echo replies to the victim causing a congestion at either links or routers on the path to the victim. ICMP smurf flood can easily be detected by observing the traffic heading for a system. A large number of echo replies heads for the victim host even though the victim host has not sent any echo requests [10] [9].

DHCP starvation attack

The Dynamic Host Configuration Protocol (DHCP) has two main features: a protocol for delivering host-specific configurations from a DHCP server to a host, and a mechanism for allocating network addresses to a host. The DHCP acts by the client to server model, where a DHCP server acts as a host providing the client with network configuration parameters and allocating a network address to the client. When allocating a network address to a client the client starts with sending out a DHCPDISCOVER to locate available servers. The server replies with a DHCPOFFER that offers the client certain configuration parameters and a network address. The client then sends a DHCPREQUEST to the server with the wanted network configurations, declining the other servers' DHCPOFFERs. Lastly the server sends a DHCPACK with the network configurations and the network address [7].

To perform a DHCP starvation attack the attacker sniffs the network for a DHCPREQUEST from the victim. From this request the attacker can acquire the MAC address of the victim. The DHCP server then sends a DHCPACK to the victim. After this the victim broadcasts an ARP_REQ to check if any other client on the network is using the IP address offered by the server. Since the victim does not yet have an IP address it sends the ARP_REQ from IP 0.0.0.0. The attacker can then sniff the network for packages sent from IP 0.0.0.0 and send a false ARP_REP replying to the victim with its own MAC address as source MAC. When the victim receives the false ARP_REP from the attacker it interprets this as if the IP address is occupied by another client, the attacker. The victim then starts over the process of acquiring an IP address and the attacker starts over the process of denying the victim an IP address causing a denial of service [12].

Man-in-the-middle attack

A man-in-the-middle (MITM) attack is when an attacker puts itself between two hosts and intercept their communication without them knowing about it. A MITM attack can be executed against a wide range of protocols. As an example consider a client communicating with a web server that is using the Transport Layer Security (TLS) protocol. The majority of web servers and browsers uses TLS as security protocol to setup private end-to-end connections between client and server. TLS uses a public key exchange between the client and server to encrypt the data being sent. Assume client A wants to connect to server B. The MITM attacker uses snooping to intercept the public key exchange between A and B. When the attacker gets A's public key he can then send his own key to A pretending to be server B. The attacker then starts a secure TLS connection to the server and transmits A's data to B, pretending it is A who sends the data. Now both A and B are sending their information to the attacker who controls what is sent between them, while they believe they are communicating directly with each other. Although strong encryption and authentication mechanisms are the best way to prevent an MITM attack, one could detect snooping and spoofing, used in MITM attacks, with a NIDS [9] [27].

ARP spoofing

The Address Resolution Protocol (ARP) is a protocol situated in the network layer. ARP is used by the IP to find the destination node's MAC address via the destination node's IP address. Consider if host A wants to communicate with host B, then A sends an ARP request to B and B responds with B's MAC address in an ARP reply to A. When the right address is found the host stores the corresponding MAC address in an ARP cache for quicker access the next time A wants to communicate with B [4].

The problem with ARP is that an ARP request is sent to all hosts on the local network that the requester is connected to and that responses are not tracked to a request. This means that an attacker could send ARP replies to a host without the host ever sending an ARP request and map their own IP to the actually targeted hosts MAC address. The victim can not determine if these are legitimate ARP responses and sends packets to the attackers MAC address. To continue spoofing an IP address the attacker can flood the victim with forged ARP responses overwriting legitimate ARP responses in the victims ARP cache. This is called ARP cache poisoning.

ARP spoofing can be used in a MITM attack, consider host A trying to communicate with host B. Attacking host C poisons host A's ARP cache making it believe B's IP matches C's MAC address. Host C also poisons B's ARP cache making it look like A's IP is connected to C's MAC address. Now the attacking host C intercept all packets sent between A and B and can do harm in different ways. The attacking host can extract information that is sent between A and B. It can block all of their communication with each other or send their packets to different locations [16].

2.3 Evading NIDSs

Insertion and evasion attacks can be used to fool a NIDSs into either miss packets heading for the end-system or distract it with meaningless packets that the end-system is rejecting anyway. The latter is called an insertion attack. An insertion attack exploits NIDSs' use of pattern matching by adding more packets to the data stream making it harder to evaluate. Evasion attacks on the other hand are executed in a way that makes the NIDS reject incoming packets that are accepted by the system monitored by the NIDS. Evasion attacks makes the NIDS read a different stream of data than the end-system. In both cases the attacker exploits the fact that a stream of data can be divided into many different TCP segments, at the transport layer, or IP fragments, at the network layer. These packets may be received in a different order from when they were sent and it is up to the receiving end-system and NIDS to reassemble the TCP segments or IP fragments in the right order.

Insertion attacks makes use of reassembly in which they add packets to the stream. This can cause the NIDS to reassemble the packets in the wrong order, overwrite other packets in the stream or just add content to the stream. In all cases it leads the NIDS to interpret the data incorrectly. Evasion attacks instead causes the NIDS to miss parts of the packet stream. Missing packets can lead to the NIDS missing the context of the data, which might be harmful for the end-system [21].

3 Method

This chapter explains how the experimental environment was set up and how the network attacks were conducted. It also explains how SNORT was implemented and which rules were used for this study.

3.1 Experimental environment

An Apache web server was set up on a computer that ran Ubuntu to act as the victim for all the network attacks that were conducted in this study. In this paper this server is referred to as the web server. It served as a simple user interface where you could submit your first name and last name. If the server was denied of service you were not able to submit your name. This was used as a check to tell if the web server was working during an attack or not. The web server was connected to the Internet through a gateway router. On the same network, also connected to the Internet through the same gateway router, another computer was used to act as the attacker. All attacks were executed from this computer. The experimental environment is illustrated in Figure 3.1.

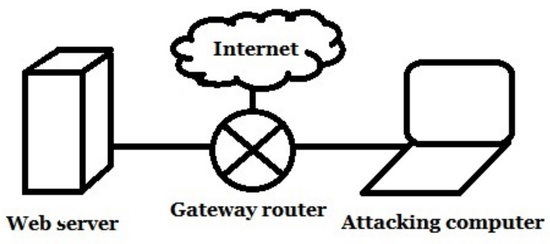


Figure 3.1: Illustrated environment network

3.2 Implementation of SNORT

On the same computer as the web server SNORT was installed and configured to sniff packets sent to and from the server. The server's IP address was configured as the home network and all other IP addresses were seen as coming from an external network. When installing SNORT on the web server the preprocessors build-essential, DAQ, bison flex, libpcap, libpcre3, libdumbnet, sfPortscan, frag3, stream5 and zlib1g were installed. These were used to capture network traffic, support regular expressions, compress data and compile SNORT. The rules used by SNORT were the community rule set as well as the SNORT registered rule set, both provided by snort.org. When none of the rule sets detected an attack, custom rules from stand-alone rule writers were added. For example, rules were acquired from forums and sometimes modified by the authors. Anyone is able to download the community rule set from SNORT's website, but to download the registered rule set you need to be registered at snort.org.

3.3 Executed port scan

The port scan attack was done by using the program Nmap. First it was used to find live hosts within a range of specified IP addresses. Nmap was then used to scan the web server for ports and which operating system it was running on with a TCP SYN scan.

3.4 Executed TCP SYN flood attack

A program to execute DoS attacks, hping3, was used on the attacking computer. Three different TCP SYN flood attacks were launched with hping3 against the web server. The first attack was the most basic, it sent as many TCP SYN packets as fast as possible to the web server on port 0 which is Hping3's default port to send traffic on. The second attack sent TCP SYN packets with the same frequency and on the same port as the first attack but the source IP address was randomized. The third attack was identical with the second attack except it sent packets on port 80.

3.5 Executed Slowloris attack

A Slowloris script written in Perl were used on the attacking computer. When the Slowloris script was run it opened 10000 connections on port 80 to the web server, with a 5 second TCP connection timeout and a 100 second retry timeout.

3.6 Executed MITM attack

On the attacking computer IP forwarding was enabled and Dsniff was installed, containing a tool for ARP spoofing called Arpspoof. The attacker used Arpspoof to send ARP replies to both the web server and the gateway router. This tricked the web server into believing that the attacking computer is the gateway router, and in the same way it tricked the gateway router into believing the attacking computer is the web server. All packets sent between the web server and gateway server were redirected through the attacking computer, making it able for the attacker to see all packets sent between them. A program named Drifnet was used to display the pictures that were sent between the web server and the gateway router. Another Dsniff tool named Urlsnarf was used to display all the requested URLs sniffed from HTTP traffic between the web server and the gateway router.

The configuration for how the attacking computer handled the time to live (TTL) for packets sent through it was modified twice with attempts to avoid detection by SNORT. The first modification did not reduce the TTL by one for packets sent through the attacking computer. And, the second modification added one to the TTL, instead of reducing it by one, for packets sent through the attacking computer.

4 Results

4.1 Results

The result chapter presents if and how the conducted network attacks affected the web server. And, whether or not SNORT detected the attacks, and if so, with which rule set.

Port scan

The first run with Nmap scanned for available hosts on the network. SNORT did not detect this scan with neither the community rule set nor the registered rule set. The second run scanned the web server for ports and operating system with success. It displayed the ports as well as which operating system the web server was running on. SNORT did not detect the port scan with the community rule set. SNORT detected the second scan with the registered rule set thanks to the preprocessor sfPortscan, this preprocessor is designed, but not limited, to detect TCP portscans from Nmap and is included in the registered rule set [22]. Therefore there is no specific SNORT rule responsible for the detection.

TCP SYN flood attack

The TCP SYN flood attacks did slow the web server down a bit but they were not able to fully deny its service. SNORT did not detect the attacks with the community or the registered rule set. Though, it did detect all TCP SYN flood attacks with the custom rule 4.1. This rule alerts if more than 70 SYN packets are sent from the external network, from any port, to the home network on any port within 10 seconds. The flags keyword set to S indicates that SNORT is looking for SYN packets. In an attempt to avoid detection by SNORT the source IP address was randomized. SNORT still alerted with the same rule as previously. Also, when the TCP SYN flood attack was modified to send packets on a different port it still got detected with the same rule.

$$\begin{aligned}&alert\ tcp\ \$EXTERNAL\_NET\ any -> \ \$HOME\_NET\ any\ (msg: \text{“}Possible\\ &TCP\ DoS\text{”};\ flags:\ S;\ flow:\ stateless;\ detection\_filter:\ track\ by\_dst,\\ &count\ 70,\ seconds\ 10; sid: 199203)\end{aligned} \tag{4.1}$$

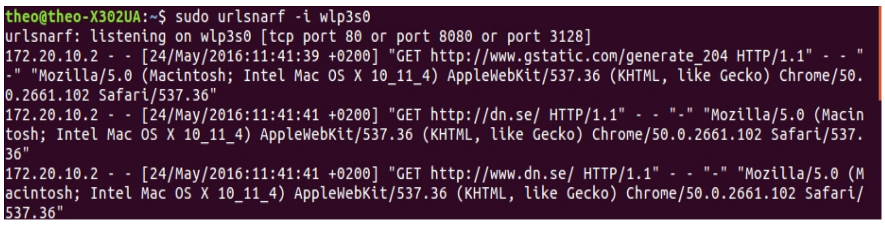
Slowloris attack

The Slowloris attack denied all service of the web server. Since all sockets on port 80 were occupied new connections were not able to be established. SNORT did not detect that a Slowloris attack was engaged with the community rule set. SNORT detected the Slowloris attack with the registered rule set. The content keyword in the rule that detected the Slowloris attack matches the pattern defined after the keyword written in text and/or binary [25]. As seen in rule 4.2 it looks for specific content and the absence of two consecutive CRLF indicates an incomplete HTTP header, which is detected with this content keyword.

$$alert\ tcp\ \$EXTERNAL\_NET\ any -> \$HOME\_NET\ 80\ (msg : \text{“}SLOW - LORIS\text{”};\ flow : established,\ to\_server;\ dsize : 8;\ ;content : \text{”}X - a|3a20|b|0d0a|\text{”};\ depth : 8;\ sid : 1337) \tag{4.2}$$

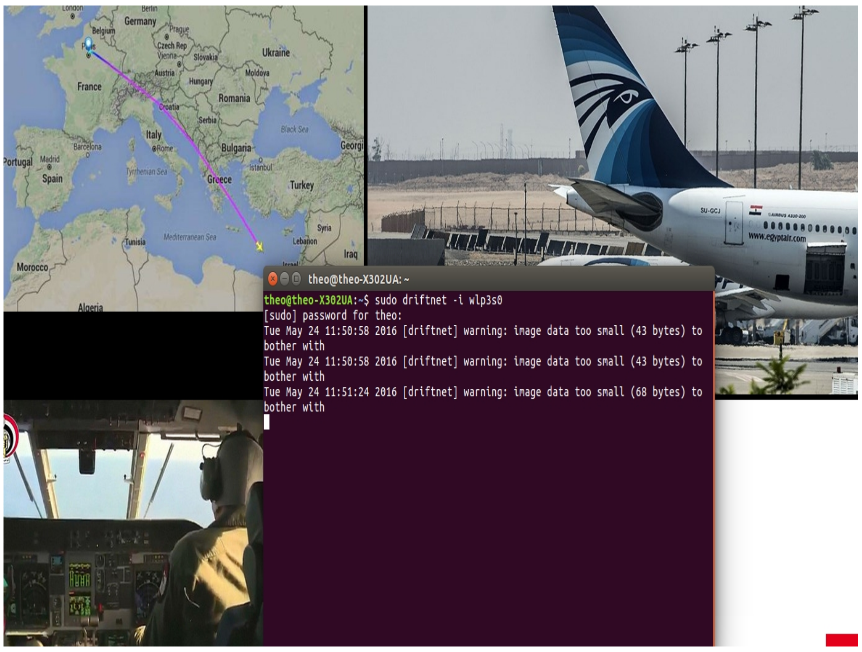
MITM attack

The MITM attack successfully poisoned the ARP cache of the victims, the web server and the gateway router. All packets between the web server and the gateway router were redirected through the attacking computer. The attacker was able to extract information from them, for example: requested URLs, as seen in Figure 4.1, and images, as seen in Figure 4.2. SNORT



```
theo@theo-X302UA:~$ sudo urlsnarf -i wlp3s0
urlsnarf: listening on wlp3s0 [tcp port 80 or port 8080 or port 3128]
172.20.10.2 - - [24/May/2016:11:41:39 +0200] "GET http://www.gstatic.com/generate_204 HTTP/1.1" - - "
-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.
0.2661.102 Safari/537.36"
172.20.10.2 - - [24/May/2016:11:41:41 +0200] "GET http://dn.se/ HTTP/1.1" - - "-" "Mozilla/5.0 (Macin
tosh; Intel Mac OS X 10_11_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.
36"
172.20.10.2 - - [24/May/2016:11:41:41 +0200] "GET http://www.dn.se/ HTTP/1.1" - - "-" "Mozilla/5.0 (M
acintosh; Intel Mac OS X 10_11_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/
537.36"
```

Figure 4.1: Extracted URLs from the traffic between the web server and the gateway router



```
theo@theo-X302UA:~$ sudo driftnet -i wlp3s0
[sudo] password for theo:
Tue May 24 11:50:58 2016 [driftnet] warning: image data too small (43 bytes) to
bother with
Tue May 24 11:50:58 2016 [driftnet] warning: image data too small (43 bytes) to
bother with
Tue May 24 11:51:24 2016 [driftnet] warning: image data too small (68 bytes) to
bother with
```

Figure 4.2: Extracted images from the traffic between the web server and the gateway router

did not detect the MITM attack with neither the community rules nor the registered rules. It detected the attack with the added custom rule 4.3. This rule alerts when a shorter route to the gateway device is available but is not used. This is because all packets are sent through the attacking computer before it reaches the gateway router and thus a longer route is used. The itype keyword notifies that the ICMP message is of type 5 and icode that it is of code 1. Code 1 under ICMP type 5 means that the datagrams for the host have been redirected. The ICMP message therefore advises the web server that there is a shorter route to the destination [20].

$$alert\ icmp\ \$EXTERNAL\_NET\ any -> \ \$HOME\_NET\ any\ (msg : \text{“}ICMP\ redirect\ host\text{”};\ icode : 1;\ itype : 5;\ classtype : bad - unknown;\ sid : 472;\ ) \tag{4.3}$$

Both attempts with different TTL configurations were still detected by SNORT with the same rule (4.3) as in the first execution of the MITM attack.

Summary

None of the network attacks were detected when SNORT was configured with the community rule set. When SNORT was configured with the registered rule set it detected two, the TCP SYN port scan and the Slowloris attack. We were able to create custom rules that detected all of the attacks that neither the community nor registered rule set did not detect. The results are presented in Table 4.1.

	Community	Registered	Custom
Host scan	No	No	Not tested
TCP SYN scan	No	Yes	Not tested
TCP SYN flood	No	No	Yes
TCP SYN flood, randomized source IP	No	No	Yes
TCP SYN on port 80	No	No	Yes
Slowloris	No	Yes	Not tested
MITM	No	No	Yes
MITM with modified TTL	No	No	Yes

Table 4.1: Result of network attacks

5 Discussion

In this chapter we discuss the results and if they were expected or not. We will also discuss the methods we have used and what could have been done differently.

5.1 Results

The first scan that searched for available hosts is not an attack and was not expected to trigger any rule. We chose to included that scan since it is often the first part when executing a port scan, but we did not create a custom rule to detect it. When performing the port scans with the TCP SYN scan-technique we were not sure what to expect from SNORT. On one hand, as mentioned in Section 2.2, an IDS is less likely to log SYN connection attempts and should therefore not detect the port scan. Although, port scanning can be used before launching an actual attack and should in our opinion be detected by an IDS. The TCP SYN scan was detected by the preprocessor sfPortscan that has its own inbuilt rules and configurations. These rules are not available for the user and we could therefore not analyze any rules for the port scan. We expect this preprocessor to detect all TCP SYN scans executed with Nmap, since it is designed to do so and there are no known false positives or false negatives according to snort.org for this case [23].

The first TCP SYN flood attack was created with hping3's default settings. The second and third attack was slightly modified to see if they would go undetected by SNORT.

We found it surprising that SNORT did not detect any of the TCP SYN flood attacks with neither the community rule set nor the registered rule set. Since the TCP SYN flood attack is such a common attack we expected it to be detected. All three TCP SYN flood attacks were however detected with the custom rule 4.1 as we expected. Since, this rule looks at how many packets are sent per time interval, we set the number of packets to an abnormal amount for our server in such a short time interval. You can change the number of packets and the time range to match your network's traffic. But, to use this rule it is important to know how much traffic your network normally receives so it does not alert when ordinary amount of SYN packets are received. This attack was modified to attack different ports to see if it could bypass SNORT, but it did not make any difference.

The Slowloris attack was not detected with the community rule set provided by snort.org. But, when SNORT was configured with the registered rule set it detected the attack. As expected from the theory in Section 2.2 the rule was triggered and alerted after the attack had been performed because of Slowloris never completing any of its requests. So even if a server is under attack the administrator would not be alerted of such circumstances until after the attack is finished. This DoS attack is powerful if not prevented. It does not need a lot of resources to be successful, as the other DoS attack executed in this study needs. And, the requests that it sends looks like normal requests which makes it hard to distinguish the attack from normal requests. The problem with SNORT versus DoS attacks is, as mentioned in 2.2, that it is hard to distinguish the packets used in DoS attacks from packets used in normal network activity. And, as we could see from the case with the Slowloris attack, that when SNORT detected the attack, it was already too late. Also, even if an attack is detected, the victim will have a hard time preventing the outcome of the attack. Because in most cases it will come down to which side is having the most resources.

Something unforeseen was that SNORT did not detect the MITM attack neither with the community rule set nor with the registered rule set. Since it is a common network attack as we brought up in Section 1.1, we thought at least one of the rule sets should trigger an alert. We were not sure how SNORT would detect such an attack, but found it alarming that it did not.

What is interesting though is that if the MITM attack is started before SNORT and keeps running after SNORT also is started, SNORT will not detect the attack. This is because SNORT is started when the route already is longer, due to the redirecting through the attacking computer, and therefore does not detect the attack. The custom rule that detected the MITM attack is not really designed to detect MITM attacks. It triggers when ICMP says there is a longer route, which is not equal to an ongoing MITM attack. It could also be triggered if there is in fact a shorter route for the packets, which would lead to false positives as a result from SNORT. However, during our experiments with the rule, the alert was only triggered when the MITM attack was active which generated true positives.

We thought that the modification of the TTL on packets redirected through the attacking computer would avoid detection, but it did not. This made us believe that SNORT does not only, or not at all, look at the TTL to decide if the distance of the route has changed. The results of our modified MITM attack were not expected, but it shows that SNORT can handle modified attacks as well.

We did not expect the community rule set to not detect a single attack executed in this study. We think one reason for this could be that the community rule set lacks a lot of important rules in an attempt by snort.org to obtain more registered user. But, even though the registered rule set contained more rules it only detected two of the attacks in this study.

There is one more rule set that snort.org provides, but this requires you to buy a subscription at snort.org. We think that this rule set would detect more of the attacks executed in this study because it contains more rules.

Our interpretation is that good performance of SNORT goes hand in hand with the competence of the user and the rules that he/she implements. We believe SNORT can be a very good IDS if you understand network attacks and know what to look for. Although, you do not have to be a security expert to use SNORT, there are still companies that can help you install and maintain SNORT or that makes use of SNORT in their own security products. Also, we found a lot of documentation for SNORT and came across many helpful users on the Internet.

5.2 Network security in a work environment

We interviewed Tranås municipality's IT-security department regarding their network security to get a better understanding of how IDSs can be used in a working environment. Although they used a wide variety of security measurements they only used IDSs for their external web servers, but they believed that they will have to implement internal IDSs in the future because of increasing threats. The IDS they use is provided by a Swedish security company called Clavister. Clavister's IDS uses the same approach as SNORT namely misuse detection and Clavister also provides Tranås municipality's IT-security department with sigantures for what to detect. Clavister provides signatures for all the categories of attacks that we have studied in this report [1].

Clavister themselves mentions the presence of open source IDSs, but encourage the use of their product as writing your own signatures can be a complicated task. That they encourage their own product is pretty expected from a company selling a product, but we agree to their opinion that writing your own signatures can be difficult depending on what to detect. The interview gave us an example where IDSs are used in a working environment. Even though our interviewee did not use SNORT the IDS solution they used is similar [1].

5.3 Method

The validity of our experiments is a bit compromised due to different approaches of conducting the same type of attacks. For example, an MITM attack can be performed with other techniques than the techniques we adopted. Same goes for the DoS attacks, even if the malware used to perform the attacks are built on the same principles as other malwares achieving the same result, the underlying code could be different. So it is not totally certain that SNORT would always detect these attacks, though how SNORT detected them agrees with the theory the attacks are based on. The reliability of the method is however still consistent. Using our method for configuring SNORT and the attacks with the same experimental environment will have the same outcome. Although the setup leaves little room for modifications as these could alter the outcome, for example not installing one of the preprocessors or deleting a rule in SNORT could cause SNORT to miss an attack.

One aim in this study was to find out if it was possible to modify the conducted attacks to bypass SNORT. Unfortunately, we were only able to modify two of the conducted attacks. One attack was conducted by using a downloaded program and we did not have access to the source code which made it hard to modify it. Another attack was conducted with a script written in Perl and we did not have enough knowledge of how to change this script to bypass SNORT.

One problem when evaluating NIDSs in general is that we would have to test other NIDSs on the market and infinite different configurations of SNORT. The fact that we used only one type of NIDS prevented us from evaluating different detection approaches. This made us unable to point out weak spots of NIDSs using other detection approaches than misuse detection. This limited our way to achieve the aim of this study, since we wanted to point out weak spots of NIDSs and what to improve for further development of NIDSs and not only the type we used.

5.4 The work in a wider context

It is illegal to expose an entity with the exploits we have conducted without their approval. This study can be perceived as a threat towards private persons or organizations. It is important to understand that these attacks are only created to evaluate the NIDS we have implemented and will not be used for any other use. We point out the weaknesses we have found in an attempt to encourage improvement in NIDSs and not to give hackers an advantage.

6 Conclusion

SNORT presented few weak spots during the attacks in this study. Since SNORT is so dependent on the user, we consider that the greatest weakness could be the incompetence of said user. Which intrusions SNORT detects is entirely up to the rules of SNORT. This generates another weakness of SNORT, that it is based on misuse detection which, as mentioned in Section 2.1, has problems dealing with novel attacks. SNORT has to know about the structure of the attack before it can detect it.

The rule sets obtained from snort.org detected two of the networks attacks conducted in this study. The rest of the attacks were detected by custom rules. SNORT performed well against all attacks in this study even if we had to implement custom rules. But, you cannot expect to be protected from network attacks with neither of the free rule sets.

We reckon that with more knowledge of how to write SNORT rules more attacks can be detected by SNORT. The same goes for the hackers point of view. It is possible to modify a network attack that has been detected to bypass SNORT. If the attacker is experienced it can create exploits that does not alert any rule. It can be seen as a battle between the attackers and the SNORT users. An advantage the SNORT users have is that SNORT provides them with free tools and a helpful community that seek to create a secure network environment.

Our suggestion to improve NIDSs is to combine different detection and analyze techniques to obtain more ways to detect malicious traffic. By merging the best parts of the different methods a NIDS could use the strengths of SNORT as well as the strengths of anomaly detection to also detect previously unknown attacks as an example.

SNORT is a good choice to increase the security of a network, but keep in mind that you are never completely safe from network attacks and a continuous development of new rules and techniques are encouraged. Staying updated with knowledge about how new and present attacks operate is also a good measure for increasing network security.

Bibliography

[1] Clavister AB. *Intrusion Detection and Prevention*. 2011. URL: http://www.simet.com.tr/documents/product/679DA31A-FABF-8A9B-7D07C9F0761FE0EB.pdf.

[2] Raven Alder and Jay Beale. *Snort 2.1*. Syngress Publishing, 2004.

[3] Neil Archibald, Gilbert Ramirez, and Noam Rathaus. *Nessus, Snort & Ethereal Power Tools*. Syngress Publishing, 2005.

[4] Yusuf Bhaiji. *Network Security Technologies and Solutions*. CCIE Professional Development, 2008.

[5] Matt Bishop. *Introduction to Computer Security*. Pearson Education, Inc., 2005.

[6] Cisco. *What is Snort?* 2016. URL: https://snort.org/faq/what-is-snort.

[7] Ralph Droms. *Dynamic Host Configuration Protocol*. Tech. rep. RFC editor, 1997.

[8] Roy Fielding, James Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. *Internet Control Message Protocol*. Tech. rep. RFC editor, 1999.

[9] Ali A. Ghorbani, Wei Lu, and Mahbod Tavallaee. *Network Intrusion Detection and Prevention*. Springer, 2010.

[10] Qijun Gu and Peng Liu. "Denial of Service Attacks". In: John Wiley & Sons, 2008.

[11] Robert Hansen and John Kinsella. *Slowloris HTTP DoS*. 2013. URL: https://web.archive.org/web/20150315054838/http://ha.ckers.org/slowloris/.

[12] Neminath Hubballi and Nikhil Tripathi. "A closer look into DHCP starvation attack in wireless networks". In: (2017). URL: http://www.sciencedirect.com/science/article/pii/S0167404816301262.

[13] InternetLiveStats.com. *Internet Users*. 2016. URL: http://www.internetlivestats.com/internet-users/.

[14] Clifford Kahn, Phillip A. Porras, Stuart Staniford-Chen, and Brian Tung. *A Common Intrusion Detection Framework*. Tech. rep. IEEE, 1998.

[15] Resul Dasc, Abubakar Karabade and Gurkan Tuna. *Common Network Attack Types and Defense Mechanisms*. Tech. rep. Department of Software Engineering, 2015.

[16] Andrew Lockhart. *Network Security Hacks*. O'Reilly Media, 2004.

[17] Ian Muscat. *How To Mitigate Slow HTTP DoS Attacks in Apache HTTP Server*. 2013. URL: `https://www.acunetix.com/blog/articles/slow-http-dos-attacks-mitigate-apache-http-server/`.

[18] nmap.org. *Nmap Reference Guide*. 2017. URL: `https://nmap.org/book/man.html#man-description`.

[19] nmap.org. *Port Scanning Techniques*. 2017. URL: `https://nmap.org/book/man-port-scanning-techniques.html`.

[20] Jon Postel. *Internet Control Message Protocol*. Tech. rep. RFC editor, 1981.

[21] Thomas H. Ptack and Timothy N. Newsham. *Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection*. Tech. rep. Secure Networks Inc., 1998.

[22] Daniel Roelker, Marc Norton, and Jeremy Hewlett. *README.sfportscan*. 2004. URL: `https://www.snort.org/faq/readme-sfportscan`.

[23] Daniel Roelker, Marc Norton, Jeremy Hewlett, and Nigel Houghton. *Sid 122-1*. 2017. URL: `https://www.snort.org/rule_docs/122-1`.

[24] Ravi S. Sandhu and Pierangela Samarati. *Authentication and Access Control and Intrusion Detection*. Tech. rep. IEEE Communications, 1997.

[25] Snort Team. *SNORT Users Manual*. 2016.

[26] Brian Tung. *Common Intrusion Detection Framework*. 1999. URL: `http://gost.isi.edu/cidf/`.

[27] Sean Turner. *Transport layer security*. Tech. rep. IEEE, 2014.

[28] Prem Uppuluri and R. Sekar. *Experiences with Specification-Based Intrusion Detection*. Springer, 2001.

[29] Marco de Vivo, Eddy Carrasco, Germinal Isern, and Gabriela O. de Vivo. “A Review of Port Scanning Techniques”. In: (1999). URL: `http://doi.acm.org/10.1145/505733.505737`.