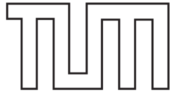
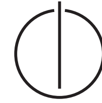
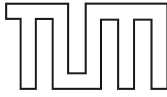
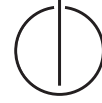



Technische Universität München
Department of Computer Science

Bachelor's Thesis in Informatics

Design of a network security violation detection system

Max Helm

Technische Universität München

Department of Computer Science

Bachelor's Thesis in Informatics

Design of a network security violation detection system

Konzeption eines Systems zur Erkennung von Sicherheitsverletzungen in Netzwerken

Author Max Helm

Supervisor Prof. Dr.-Ing. Georg Carle

Advisor Nadine Herold M.Sc., Dipl.-Inf. Stephan-A. Posselt

Date 15.03.2015

Informatik VIII
Chair for Network Architectures and Services

I confirm that this thesis is my own work and I have documented all sources and material used.

Garching b. München, 15.03.2015

Signature

Abstract

Most devices are connected to the Internet or an intranet. Valuable information can be stored on those devices and gets transferred over these networks. This makes those networks profitable targets for attacks and results in the need of security systems to protect the data stored on network devices and the data traversing those networks. We will take a look at currently available systems that provide security for networks and evaluate their strengths and weaknesses. Based on this we will suggest a concept for a system that takes a hybrid approach. The system relies on the collection of metadata about the network using a set of distributed sensors in order to derive information about the network topology and about network properties. The limited views of the single sensors get correlated into one global view, yielding additional information. This information will be used in conjuncture with a set of rules to identify possible security violations and misconfigurations in the network.

Zusammenfassung

Heutzutage sind die meisten Geräte mit dem Internet oder einem Intranet verbunden. Auf den Netzwerkgeräten sind oft wichtige Daten gespeichert und über die Netzwerke werden wertvolle und vertrauliche Informationen übertragen. Das macht diese Netzwerke ein lohnenswetes Ziel für Angreifer. Deshalb besteht das Bedürfnis nach Sicherheitsmechanismen, welche sowohl die Daten auf den einzelnen Geräten, also auch die Daten welche über das Netzwerk übertragen werden, vor unbefugten Zugriffen zu schützen. Wir werden einige Systeme betrachten, welche Sicherheit für Netzwerke bereitstellen und ihre Stärken und Schwächen analysieren. Basierend darauf werden wir ein System vorschlagen, welches Eigenschaften einiger dieser Systeme verwendet. Es beruht darauf, Metadaten über das Netzwerk durch ein Netz von verteilten Sensoren zu sammeln und aus diesen Metadaten die Netzwerktopologie und bestimmte Netzeigenschaften abzuleiten. Diese Informationen, welche durch die beschränkten Überblicke der einzelnen Sensoren jeweils nur einen beschränkten Überblick über das Netzwerk bieten, werden hierbei in einen Gesamtüberblick über das Netzwerk zusammengefasst. Aus diesem Überblick können zusätzliche Informationen gewonnen werden. Diese Informationen werden zusammen mit einem Regelsatz verwendet um Sicherheitsverletzungen und Fehlkonfigurationen von Entitäten des Netzwerks zu erkennen.

Contents

1 Introduction 1
1.1 Goal . . . 2
1.2 Outline . . . 2

2 Analysis 5
2.1 Network Topology . . . 5
2.1.1 MAC addresses . . . 6
2.1.2 Ports . . . 6
2.1.3 Switch ports . . . 7
2.1.4 ARP cache integrity . . . 8
2.1.5 DHCP servers . . . 9
2.1.6 Rogue servers . . . 10
2.1.7 Load balancing . . . 10
2.1.8 Tunnelling . . . 11
2.1.9 Routing and CAM table . . . 12
2.2 Network Services . . . 12
2.2.1 Reachability . . . 12
2.2.2 DNS . . . 13
2.2.3 Encrypted channels . . . 14
2.3 System Security . . . 14
2.3.1 Service versions . . . 14
2.3.2 Processes . . . 15
2.3.3 Users . . . 15
2.3.4 Certificates . . . 16
2.3.5 Bad USB devices . . . 16
2.4 System Reliability . . . 17
2.4.1 Backup links . . . 17
2.4.2 Backup . . . 18
2.4.3 Storage mediums . . . 18
2.5 Requirements . . . 19

3 Related Work 21

3.1 IDS types . . . 21
3.2 IDS implementations . . . 22
3.2.1 SNORT . . . 22
3.2.2 ESUKOM . . . 23
3.2.3 ONTIDS . . . 23
3.3 Monitoring tools . . . 24
3.3.1 Nagios . . . 24
3.3.2 Nmap . . . 25
3.3.3 Munin . . . 25
3.4 Comparison to requirements . . . 26

4 Design 31
4.1 Components . . . 31
4.2 Graph Model . . . 34
4.3 Devicescanner . . . 36
4.4 Analyzer . . . 38

5 Implementation 41
5.1 Neo4J and Cypher . . . 41
5.2 Database . . . 43
5.3 Devicescanner . . . 44
5.4 Analyzer . . . 47

6 Evaluation 49
6.1 Qualitative evaluation . . . 49
6.2 Quantitative evaluation . . . 50
6.2.1 Test environment . . . 52
6.2.2 Testing tools . . . 52
6.2.3 CPU and memory utilization . . . 52
6.2.4 Network traffic . . . 56
6.2.5 Logging delay . . . 56

7 Conclusion and Outlook 59

Bibliography 61

List of Figures

4.1 UML component diagram of our model 32
4.2 UML sequence diagram of a use case where a security violation is detected 33
4.3 Diagram of the graph model 35
4.4 Diagram of the scanner 37

5.1 Relationship graph between three persons 41
5.2 Cypher representation of a simple graph 43
5.3 Cypher query to find the set of persons Alice likes 43
5.4 Class representation 44
5.5 Find method in the MacAddressRepository 44

6.1 CPU usage of the idle device versus CPU usage with the devicescanner running 53
6.2 RAM usage of the device with the devicescanner running 54
6.3 Example of the data a local devicescanner writes into the database . . . 55
6.4 Bandwidth usage without using a local buffer versus badwidth usage when using a local buffer 56
6.5 Time it took the Analyzer to detect a new port for different scanning intervals 57

List of Tables

2.1 Mapping of each subsection to its own datum 19

3.1 Comparison of the systems discussed in this chapter with regards to the requirements from section 2.5 28
3.2 Comparison of the datums and the related work 29

4.1 Example ports and their properties 33
4.2 Unwanted combinations of the active and config flags 40

5.1 The corresponding relational representation to Fig. 5.1 42
5.2 Performance of the relational- versus the graph-model 42

6.1 Evaluation of the requirements 50
6.2 Comparison of the own solution to the security violations and misconfigurations as defined in chapter 2 51

Chapter 1

Introduction

Today networks are used basically everywhere. Company networks are used to connect people in their workplaces and allow easier communication and the exchange of data. Industrial networks are used in order to connect production processes. Because valuable information is stored on the hosts located in those networks and valuable information gets exchanged over those networks, they are a profitable target for attacks. This leads to an increasing need of securing those networks against intrusions.

For this purpose, there are intrusion detection systems(IDS). They rely mostly on analyzing all the traffic that traverses the network in order to find suspicious or malicious packets. However, with increasing size of the network and increasing amount of traffic traversing the network this process demands an increasing amount of computational power in order to be successful and has its limitations. It cannot detect security problems which are not legible from the network traffic, for example, if a service running on a host using an outdated version, which is susceptible for attacks; this will only be detected as soon as an attack gets launched.

In addition to the intrusion detection systems there are network monitoring tools, which are used to surveil network- and system-resources. They are, in contrast to intrusion detection systems, not used to detect malicious behaviour of certain hosts but rather to detect misconfigurations and failures of important network resources. Miscofigurations can lead to a loss of functionality of the network and an increase of attack vectors.

However, sometimes misconfigurations and attacks cannot be easily distinguished, which causes the detection systems to produce false-positives. This is because there is not enough information available to decide whether it is actually an attack or not. The lack of information is partly cause by the system only having a limited view of the network. For example, a IDS is usually placed at ingress or egress points of a network, where it can only monitor the traffic which passes by it. Multiple instances of IDS can be deployed in order to lower the impact of this, but they don't correlate their traffic analysis with one another.

In order to make better decisions there should be a hybrid system which uses a global view of the network to gain additional information and includes features from both intrusion detection systems and network monitoring tools.

1.1 Goal

The goal of this thesis is to lay down the design of a `Security Violation Detection System` as well as implementing parts of it. The system should be able to detect a series of security violation and misconfigurations that can occur in a network. Security violations occur when an attack against the network is executed or when a property of the network is in a state that opens up an attack vector. A misconfiguration occurs when a certain property of the network is not configured correctly which could lead to parts of the network not being able to function correctly any more.

The system should have a mechanism to obtain a global view of the network, which can feed additional information into the decision making process of whether to raise an alert or not.

The data which is used to detect the security violations and misconfigurations is metadata, collected by active and passive scanning techniques. The metadata can be further classified into the network topology and network properties. The network topology covers which devices are located in the network and how they are interconnected. Network properties contain such things as the names and IP addresses of the devices in the network as well as which services they are running, which certificates they hold or which users are logged into them at the moment. Once this data has been collected, it need to be analyzed in order to detect security violations or misconfigrations. If the analysis yields a violation, an alert will be raised.

We start from the premise that we have a network with an established network topology which is fairly static. An example of what this could be is a company or an industrial network.

1.2 Outline

This thesis is divided into seven chapters. After this brief introduction, the second chapter will cover an exemplary set of security violations and misconfigurations which should be discovered by our system. Furthermore, the requirements which should be met by the resulting system are laid out. In chapter 3, an overview over the different concepts of intrusion detection systems is given, alongside with some systems that are similar to the one proposed in this thesis. They are evaluated in regard to the requirements defined in the previous chapter. Chapter 4 treats the design of the proposed

system. Each component of the system is explained as well as their interactions with one another. In chapter 5 we will discuss how, and using which resources the system has been implemented. This includes a look at the underlying framework as well as the implementation of each component presented in the previous chapter. Chapter 6 deals with the evaluation of the system. It is compared to the requirements defined in chapter 2. In addition a set of measurements will be done using the implementation of the system. The last chapter summarizes the statements of this thesis and gives a brief outlook.

Chapter 2

Analysis

In this section the set of all possible security violations and misconfigurations that we want to be able to detect will be listed. Each subsection will be divided into the following three paragraphs: `Description`, `Security impact` and `Collection method`. In the `Description` it will be explained which system or network resources we want to monitor. In the `Security impact` paragraph it will be elaborated how those resources can be exploited in order to start an attack, how you can detect an attack by monitoring those resources or how they can be misconfigured, resulting in further problems. for the network. In the `Collection method` paragraph it will be explained how the resources we want to monitor can be collected. If the resource collection is done via a command or reading the content of a specific file, it is always Linux specific. For some security violations the collection method will specify the `SNMP` protocol. `SNMP` is not naturally supported and it needs to be installed on every device, which should be monitored this way.

The security violations and misconfigurations are divided into the following four categories: `network topology`, `network services`, `system security` and `system reliability`. In table 2.1 at the end of this chapter there is an overview over all of them.

2.1 Network Topology

Network topology includes everything that is related to how the devices of a network are interconnected. We take a look at access control based on the MAC addresses of devices, the ports and switch ports of devices, the ARP cache, different types of servers, load balancing and tunnelling mechanics and routing tables.

2.1.1 MAC addresses

Description—The first thing we want to be able to monitor is, which MAC-addresses are allowed per subnet. Each device is assigned its own, globally unique MAC address when it gets manufactured. Thus, we have a unique mapping between devices and MAC addresses and therefore this mechanism is useful if we want to be able to restrict access to the network for certain devices. This can be achieved either via a white-listing- or a black-listing-mechanism.

Security impact—The white-listing-mechanism would require the network administrator to enable the access for each device which is trusted. For example, you might only want to allow smartphones of company employees to be allowed to connect to the company network. The blacklisting-mechanism could be used to automatically deny access to devices that have previously displayed unwanted behaviour-patterns, e.g. violated a security policy.

This is comparable to the access control that can be configured on routers. And like access control on routers this has the one disadvantage that it can not detect MAC spoofing. MAC spoofing means that you simply pretend to have another MAC address if your real MAC address got rejected. This can be done by either reprogramming the hardware to use another MAC address or by manually sending modified packets over the network interface. For this to work, the interface needs to be in `promiscuous` mode, which means that all traffic can be received and sent over this interface. Normally, packets with a source or destination address other than the one of the interface, get automatically dropped. [1]

Collection method—The MAC addresses of devices in a network can be obtained by using the `nmap` tool.

2.1.2 Ports

Description—We want to monitor which ports are allowed, required and disallowed to be open on a given network entity. The destination and source port numbers are part of the five tuple, which uniquely identifies a connection between two hosts. The five tuple consists of the source IP address and port number, the destination IP address and port number and the protocol used (either TCP for connection oriented data streams or UDP for connectionless data streams). This also means that the connection can be distinctly mapped to an application. A lot of applications use default port numbers over which they communicate.

The default numbers are needed so that you know at which port you need to send a request if you use a certain protocol. For example, you know that a web server which uses the `http` protocol can be reached on port 80. Additionally, it can be done for technical reasons: the DHCP protocol needs to have a fixed port number from which the client communicates, because the server needs to know this in order to be able to reach the client, since in the initial stages of the protocol they communicate over broadcast addresses. The default port numbers are standardized by the Internet Assigned Numbers Authority(IANA) [2]. Therefore, you can easily detect which applications are sending data over the network at the moment.

Security impact—Monitoring which ports of a device are opened at the moment is important for the security of the system, since you might require certain devices to have certain ports open and ,therefore, applications running at all times. For example, a web server needs to have port 80 and/or 443 opened in order to provide `http`- and/or `https`-access for its clients and thus function correctly. Another reason is that you might want to prohibit devices from using certain insecure applications. For example, applications such as `telnet` transmit everything, including passwords, in plain text [3]. Since `telnet` is standardized to use port 23, you might want to prevent devices from sending or listening on port 23.

Collection method—The currently opened ports on a device in a network can be either gathered remotely by using the `nmap` tool or locally by using the `lsof` tool.

2.1.3 Switch ports

Description—We want to monitor the status of all switch ports of a device. A switch port is a physical interface that is used to establish connections between devices. This is done using an Ethernet cable. Switch ports which are not needed should be disabled. In addition we want to know which MAC address is connected to a switch port.

Security impact—The monitoring of the switch ports is done because switch ports that are not used at the moment might as well be disabled to prevent them from being misused to gain unauthorized access to the network in which the device is located. For example, if you connect your device to a switch which is connected to a network, you gain immediately access. This might not be the desired behaviour towards all people that have access to the locality.

By monitoring the MAC address connected to a switch port, it can be detected if a connection gets unplugged and a new device connects itself onto this switch port.

Collection method—The status of the switch ports of a device can be monitored and altered by using the `SNMP` protocol and requesting the `ifOperStatus` object with `OID` 1.3.6.1.2.1.2.2.1.8. It can have a total of seven different states, including `up` and `down` indicating that packets are allowed to traverse the switch port or not [4].

To obtain the MAC addresses connected to the ports of a switch, you can use the `SNMP` protocol and request the `dot1dTpFdbPort` object with `OID` 1.3.6.1.2.1.17.4.3.1.2.

2.1.4 ARP cache integrity

Description—The next thing we want to ensure is the integrity of the ARP caches across the network. The ARP cache holds mappings between MAC- and IP-addresses. If a host wants to send a packet to a specific IP-address, he needs to know the corresponding MAC-address, since it needs to be specified in the Ethernet-frame. This is required for a successful transmission of the packet over the wire. Normally a host would get this information every time he needs to send a new packet via an ARP-request, which is sent to the broadcast MAC and returns the MAC corresponding of the target IP. To avoid sending a ARP-request for every packet, the ARP cache is used. Here are all mappings between MAC and IP stored, which were learned in the past 15 minutes.

Security impact—The integrity of the ARP caches is crucial to the security of the network since a corrupted ARP-cache can be an indicator for a recently launched man-in-the-middle-attack using a technique called ARP spoofing. In an ARP spoofing attack the attacker propagates false ARP-reply packets into the network, mapping his MAC to the IP-addresses of his victims. This leads to the fact that the attacker will receive every packet destined for one of his victims. He can log those packets and then he proceeds to forward them to their real destination. Therefore, the attack is not detected by either the sender or receiver of the packets.

To detect this malicious behaviour we need to store the ARP tables of all hosts in the subnet, which we can then use to check if certain IP-addresses are mapped to more than one MAC-address. This occurs when the MAC and IP address of the victim device is stored in the ARP cache of one or more other devices and at the same time the attacker propagates his MAC address as destination for the victims IP address. Unfortunately this way of checking for inconsistencies in the ARP caches can also lead to false-positive alarms since some inconsistencies can have legitimate reasons, which will be explained later.

The ARP cache is not static. New entries get added and old ones get removed. There are two factors based on which it gets decided whether a entry gets removed or not. Firstly, if the size of the ARP cache exceeds a certain limit defined in `/proc/sys/net/ipv4/neigh/default/gc_thresh2`, which defaults to 512 entries, entries get removed if

the threshold is still broken after five seconds. This is the soft limit. There is also a hard limit at which entries get immediately removed (usually 1024). The second factor is time. After an entry is added, it is considered to be valid for a certain time frame which is randomly determined to lay between half- and one-and-a-half-times the default value (30 seconds) defined in `/proc/sys/net/ipv4/neigh/default/base_reachable_time_ms`. The timespan gets renewed if the device receives packets, which confirm the entry. Else the entry is removed after the interval times out [5].

With this knowledge, inconsistencies can occur, for example, if `Device A` requests a new DHCP lease, which means that it requests to get a new IP address from a DHCP server, the old address of `Device A` can be assigned to a new `Device B`. Since the ARP cache only refreshes in a fixed interval this can lead to inconsistencies, because the MAC address of `Device A` can still be mapped to its old IP address in the ARP cache of a device that did not communicate with `Device A` since the changes. While at the same time in the ARP cache of another device the old IP address can already be mapped to the MAC address of `Device B`.

Collection method—The contents of the ARP cache can be obtained locally by using the `arp -n` command or remotely by using `SNMP` and requesting the `ipNetToPhysicalTable` object with `OID` 1.3.6.1.2.1.4.35 [6].

2.1.5 DHCP servers

Description—We want to be able to detect if there are any unauthorized DHCP-servers running in the subnet. A DHCP server is, among other things, responsible for assigning IP addresses to all hosts connected to the network. The protocol procedure is as follows: a host that wants to connect to the network sends a `DHCP discover` message to the broadcast IP (255.255.255.255). The message contains the MAC address of the host. A DHCP server which receives this message sends a `DHCP offer` message to the broadcast IP as well, containing an IP addresses from which the host is able to choose. The host only reacts to `DHCP offer` messages that contain his MAC address. Now the host chooses one of the offered IPs and sends a `DHCP request` message to the DHCP server. The DHCP server normally replies with a `DHCP ack` message to confirm the assignment of the IP [7].

Security impact—If a rogue DHCP-server is running on a host inside the network, this can have consequences for the reachability of hosts in the network. Since hosts in an network are not configured which DHCP-server they should use, they cannot distinguish a rogue DHCP-server from a legitimate one. They simply accept the first `DHCP offer` message they receive. The rogue DHCP-server could, for example, provide a wrong subnet masks to specific hosts which will result in them not being able to

reach anyone in the network. In addition to assigning IP addresses, a DHCP server can be responsible for assigning a DNS or NTP server to hosts. In this case a rouge DHCP server could assign a malicious DNS or NTP server to the hosts as opposed to the "official" ones of the network [7].

Collection method—The DHCP servers in a network can be determined by the `dhcping` tool. It sends `DHCP request` or `DHCP inform` packets to the broadcast IP of the network in order to trigger responses from every DHCP server in the network. The responses get parsed and the IP address extracted [8].

The differentiation between legitimate and illegitimate servers needs to be configured by hand. Since you do not know which DHCP server a client is using without inspecting the packets exchanged over the network you could, for example, limit the number of DHCP servers to a specific number and if an additional server above this number appears, it would be flagged as illegitimate.

2.1.6 Rogue servers

Description—In addition to the check for rogue DHCP-servers we want to be able to detect rogue NTP-, DNS- and update-servers in the network.

Security impact—This is done for the same reason as the DHCP server monitoring, since they could send false information to their clients. For example, if you receive distorted information about the current time from your NTP server, protocols which rely on an accurate system time can encounter problems. One example would be `Kerberos` which enables hosts to authenticate themselves to one another over an insecure network. The Kerberos protocol allows for a maximum discrepancy of five minutes between the client and server time. If this discrepancy is surpassed, the authentication process cannot be completed [9]. A second problem occurs if you want to check if a certificate is expired or not. If you have a deprecated system time, you could falsely accept already expired certificates.

Collection method—Which NTP server a device uses is normally defined in the `/etc/ntp.conf` file. This can be collected and compared against the NTP server that should be used. The location of the DNS server is usually stored in `/etc/resolv.conf`.

2.1.7 Load balancing

Description—Another thing we want to monitor is the load balancing between two entities that provide the same service for the network that are redundantly deployed in

order to distribute the load onto them and thus increase the amount of requests that can be processed. Load balancing can be done, for example, by the DNS server, which evenly distributes the name resolution onto the different IP addresses [10] or by the front end of an application distributing the redirection of requests onto different back ends [11].

Security impact—If one of the entities has to process a significantly larger amount of the incoming requests, this is an indication that the load distributer used is not working (correctly).

Collection method—The bandwidth usage for an interface of a device can be obtained remotely by using `SNMP` and requesting the `ifInOctets`, `ifOutOctets` and `ifSpeed` objects. Calculation can be done according to [12].

2.1.8 Tunnelling

Description—We want to detect if there are devices which use a tunnel opened between two subnets in order to gain access to resources on the other subnet.

Security impact—This should be monitored because by creating an unauthorized tunnel you could bypass a firewall or spread malicious software that is running on your local device onto other, more critical, devices. An additional risk arises if you have split-tunnelling enabled. Split-tunnelling allows you to connect to a remote network via a tunnel and at the same time access the internet via your normal network connection instead of routing all data over the tunnel. This is done in order to prevent the use of unnecessarily long routes and thus reduce the latency [13]. If you are connected to a shared network, for example, a wireless network that is used by multiple users while you use split-tunnelling there is the opportunity for an attacker to compromise your machine over that connection and then gain access to the remote network over your tunnelled connection [14].

Collection method—In order to detect tunnelling, flow data is needed. Flow data is basically metadata about traffic: a flow is information about a sequence of packets exchanged between a source and destination [15]. Flow data can be obtained by using `flow-tools` [16].

2.1.9 Routing and CAM table

Description—For the rare case that the network uses static routing instead of a dynamic routing algorithm we want to be able to check the entries of the routing- as well as the CAM-table. An example application of static routing is if you have a small network, where the static routes are easy to configure, and you don't want to waste the links bandwidths by exchanging dynamic routing information over it. The routing table is the table used by the router to decide the route on which to route a packet on layer three of the ISO/OSI model. The CAM table is the table in a switch based on which the switch decides where to forward a packet to on layer two.

Security impact—The monitoring is done in order to detect a poisoning of the routing as well as the CAM table. Poisoning means that false routes are inserted into the table. This can be done for several reasons: you might, for example, want to shut down the functionality of the network by producing routing loops or inserting invalid IP addresses. A second reason could be that you want to execute a man-in-the-middle attack by routing all traffic over yourself and then forward it to the actual destination.

Collection method—The routing table of a device can be obtained by using the `SNMP` protocol and requesting the `ipRouteTable` object with `OID` 1.3.6.1.2.1.4.21. The CAM table of a switch can be obtained by using `SNMP` and requesting the `dot1dBasePortIfIndex` object with `OID` 1.3.6.1.2.1.17.1.4.1.2.

2.2 Network Services

This section covers everything that is related to the interaction between normal devices and entities that provide a service for the network. We will take a look at the reachability of service providers in the network as well as the usage of DNS servers and encrypted communication.

2.2.1 Reachability

Description—We want to monitor if all services, which are provided by hosts located in the network, that should be reachable from outside the network are reachable from outside. For example, a web server should be reachable from outside the network. Additionally, we want to monitor a list of all services which are not allowed to be accessed from outside the network. An example for this would be a DNS server which should not be reachable from outside the network it is responsible for.

Security impact—If a server that should be reachable from outside the network cannot be reached this way this could be an indication for a DOS attack or a misconfiguration of the server itself or any device on the route between the ingress point of the network and the server.

If hosts, which do not provide any service for clients outside their own network, are reachable from outside the network this can open up some attack vectors. For example, if a DNS-server answers queries which come from outside the network it resides in, it would enable an attacker to misuse this DNS-server for a DNS-amplification attack which is a specific Distributed-Denial-Of-Service(DDoS) attack. The attacker sends very small DNS queries with a spoofed source IP-address, which corresponds to the IP-address of the victim, to the DNS-server. A suitable DNS-query can be as small as 60 byte. The responses to these queries can reach a volume of up to the maximum DNS-response size of 512 byte. This would be an amplification of the attack traffic by a factor of eight. Newer DNS implementations, however, support even bigger response sizes of up to 4000 byte, which would result in an amplification by a factor of over 60. Since the target of the attack is most likely situated outside the network of the vulnerable DNS-server, this attack can only be carried out because the DNS-server responds to queries coming from a source IP-address located outside the network.

Collection method—To obtain the data required, you need to control a host located outside the network. If you manage to achieve this, you can simply ping each host specified in the corresponding lists of hosts that should be reachable or unreachable and compare the results.

2.2.2 DNS

Description—We want to be able to monitor if every client in the realm of the same DNS server receives the same answers to the same DNS queries.

Security impact—This is important to be able to detect the selective dispensing of spoofed DNS answers. However, a big problem here is the load balancing mechanism deployed by many big websites. In order to process the vast amount of requests they receive, they have different servers deployed with different IP-addresses, which all resolve to the same host name.

Collection method—Using the `host` command you are able to retrieve all mappings between IP addresses and domain names from the view point of the device on which it gets executed.

2.2.3 Encrypted channels

Description—We want to monitor if a service is only used by clients that use encrypted communication,

Security impact—This is a mechanism to ensure that sensitive data, protected by the usage of encrypted protocols, can not be sniffed from the network.

Collection method—Checking if each client uses encrypted communication can only be done by monitoring the traffic or by checking the ports to which the client is connected, if it is a port which is bound to a service known for providing unencrypted communication like port 23 or 80.

2.3 System Security

This section covers everything that is related to the security of the local device. It covers the security of running services and processes, the usage of certificates and so called `BadUSB` devices.

2.3.1 Service versions

Description—We want to be able to monitor the services running on a device. A service is a process that is constantly running and is reachable over a network port. We want to be able to monitor if every running service on a device is on the latest version. In addition we want to be able to control which services need to run on certain devices and which services are not allowed to run on certain devices.

Security impact—Having a service on the most recent version is important since older versions can contain security holes, which get fixed by updates to a newer version. Controlling which services run on a device is of importance to ensure the functionality as well as the security of the device. For example, you might want to have it as a necessity that you have `apache` or a similar service running on your web server to ensure functionality. On the other hand you might want to forbid certain services to run on your web server. Because a web server is an attractive resource to attack, insecure services which are executed with high privileges pose an unnecessary risk if they are not genuinely needed. An attacker can gain access to the web server by exploiting weaknesses in these services. For example, if `FTP` is running and accepting outbound connections, it could be misused to upload malicious code, which could infect the web server itself or connected systems [17].

Collection method—The running services can be either locally detected by using the `ps` command or remotely by using the `namp` tool, which compares the open ports of a device against a database of over 2000 services. This method is not completely accurate, though. Services can be operated on different ports than their default ports [18].

2.3.2 Processes

Description—We want to monitor and restrict the processes which are running on a device. We want to restrict the running processes based on a white- or black-listing-mechanism. White-listing can be used on a device where very few processes are required to run anyway. The black-listing can be used to deny certain processes the right to execute on more complex devices where creating a white-list would be too time consuming.

Security impact—Similar to the security impact discussed in 2.1.2, controlling which processes are mandatory to run is done in order to ensure functionality. Controlling which processes are not allowed to run is done in order to enhance the security of the system.

Collection method—The currently running processes on a device can be obtained by using the `ps` command.

2.3.3 Users

Description—We want to control how many and which users are logged in on a critical device at a time. For example, you might want to restrict access to one user at a time on some critical device in a industrial network such as a central pumping system.

Security impact—In addition you might want a user account only to be used to log into one device at a time, since being logged in on multiple devices at a time might be an indication for the fact that another person is using the login credentials in order to gain access to a device on a privilege level not intended for that person. Of course this could also be caused by using, for example, `ssh` to log in to a remote device. However, the use of remote login for normal users might be undesired anyway, since logging into a remote device might only be needed by the administrator of the network.

Collection method—The currently logged in users can be obtained by using the `who` command.

2.3.4 Certificates

Description—We want to be able to check the certificates of the devices in the network. A certificate is used to bind a public key to a person or an organization. Public keys are used to encrypt messages, which then can only be deciphered by using the corresponding private key. By binding the public key to a person or organization, it is ensured that you can be certain you are talking to the right person without exchanging the public keys manually and verifying the identity of the key holder. For this purpose a certificate contains information about the public key as well as the person or institution.

Certificates are being issued by `Certification Authorities(CAs)` and are only valid for a certain time span. CAs are organized in a tree-like structure. Each CA owns a CA certificate. The root CAs can issue intermediate CA certificates to CAs on a lower level. The issuer of a certificate is always included in the certificate. A company can obtain a certificate from a CA to provide authenticated and encrypted access to their web server. For this whole authentication mechanism to work, it is necessary that the person using it is trusting certain CAs. Because each certificate has a field containing its issuer, a "chain of trust" can be established from each valid certificate to a CA you trust. [19]

Security impact—A security requirement is that every device holds a certificate in order to authenticate itself when using encrypted connections. Furthermore, no device should hold an expired or an invalid certificate. In addition you are able to define a custom list of CAs that you want to trust and check if the chain of trust of every certificate contains one of these CAs. Furthermore, if you have a server providing an encrypted service you might want to define minimum requirements for certain properties, like for example, the key lengths of the certificates used. Another thing you might want to check is the cipher suites provided by the server or if the server is vulnerable to a specific exploit, for example, heartbleed.

Collection method—Certificates are normally located in `/etc/ssl/certs/`. They are stored with `base64` encoding, which can be decoded into a human readable form by using `openssl x509 -in <certificate> -text -noout`. If you want to obtain certificates remotely, this can be done by using the `SSLyze` tool. In addition to providing the basic information contained in a certificate, it checks the host and the SSL connection for common exploits [20].

2.3.5 Bad USB devices

Description—We want to monitor so called `BadUSB` devices. A BadUSB device is created if the firmware of a USB device such as a USB-stick, USB-card-reader or USB-web-cam is reprogrammed. Normally this cannot be detected since you do not have access to the

partition on which the firmware resides. The firmware is altered in such a way that the USB device impersonates another device such as a keyboard or even a network card.

Security impact—By impersonating a network card you can act as a network interface. You can then reply to DHCP queries with an answer that contains only the address of a DNS server on the internet which you control. No default gateway is specified. Therefore, all traffic gets routed over the normal internet connection except the DNS queries which will be sent to the rogue DNS server. This enables DNS redirection attacks, where you match a host name, e.g. `www.google.de`, to an IP of a server that you control. This server then impersonates the originally requested website. You are then able to steal, for example, login credentials.

If you manage to impersonate a keyboard you are able to execute commands without the user noticing. This can be used to install malicious software on the system or upload important files to a FTP server. Even if you need root privileges to execute your command you can easily get them in combination with a keylogger. The keylogger can be loaded by adding it to the `LD_PRELOAD` environment variable. Then you simply need to lock the screen, for example, by using the `gnome-screensaver-command -l` command, which is executable without root privileges, log the input when the user enters the password and then you are able to start a `sudo` session. Root privileges are needed to infect other USB devices. [21, 22]

Collection method—The `lsusb` command can be used to retrieve information about all USB buses of a device as well as the devices which are connected to each of the buses. Each bus and each device connected to a bus is assigned its own unique number. Furthermore, each type of device has a unique tuple consisting of its `InterfaceClass` and its `InterfaceProtocol`. For example, a keyboard always has an `InterfaceClass` of 3 and an `InterfaceProtocol` of 1.

2.4 System Reliability

This section covers everything related to preventing a local system from losing data. It includes the usage of backup links and redundant virtual machines and storage mediums.

2.4.1 Backup links

Description—The next thing we want to monitor is the availability of redundant or backup links between critical entities in the network.

Security impact—You might want two different routes between two entities that rely upon one another in order to provide their service correctly. Because if a component of one of the paths fails, you will still be able to provide the service over the backup route. For example, you might want to have two different routes from a surveillance camera to the server where the video feed is stored.

Collection method—If the two entities are directly connected to switches or routers, we can detect the switch port to which they are connected via their MAC addresses using `SNMP` as described in section 2.1.3. From this, it can be gathered if there is more than one path.

2.4.2 Backup

Description—We want to be able to check if a service, a virtual machine or storage device that is deployed redundantly is running on different physical devices.

Security impact—This is important for the system reliability because if a device which contains all instances of a redundantly deployed service or virtual machine fails, the service itself or the services provided by the virtual machine will be unavailable. If a device fails, which contains two redundant instances of a storage medium, all data is lost. In such cases the redundancy is useless and should therefore be set up in another way.

Collection method—You can obtain all virtual machines running on a device by using the `virsh list` tool. The information obtained contains the name of the virtual machine that was given to it upon creation as well as the status which can, among other states, be either `running` or `shut off` [23].

Details about the hard disks of a device can be obtained by using the `lsblk` command.

2.4.3 Storage mediums

Description—We want to check the status of hard drives or other physical storage mediums, meaning that we want to be able to detect and report hard disks which already failed or are likely to fail in the near future.

Security impact—If a hard disk fails, obviously all data will be lost. Or if it is deployed redundantly, the redundancy is not present any more or at least attenuated. Diagnosing the health of hard drives can reduce the risks.

Network topology	
A1	MAC addresses
A2	Ports
A3	Switch ports
A4	ARP cache integrity
A5	DHCP servers
A6	Rogue servers
A7	Load balancing
A8	Tunnelling
A9	Routing and CAM table
Network services	
A10	Reachability
A11	DNS
A12	Encrypted channels

System security	
A13	Service versions
A14	Processes
A15	Users
A16	Certificates
A17	Bad USB devices
System reliability	
A18	Backup links
A19	Backup
A20	Storage mediums

Table 2.1: Mapping of each subsection to its own datum

Collection method—Basic diagnosis of a hard drive can be done by using the `smartctl -H <disk>` tool. It displays the health of the hard disk using two states, either `PASSED` or `FAILED`. `PASSED` indicates that there are no errors, `FAILED` indicates that the disk is likely to fail within 24 hours [24, 25].

2.5 Requirements

There are various requirements to be made to the system that should be able to detect the security violations and misconfigurations listed in the previous sections.

R1: High level of abstraction – First, the abstraction level of the system should be high in order to allow an easier configuration of the rules, which are used to detect the desired set of security violations and misconfigurations of the network. A high level of abstraction means in this context that you want to have concepts in place which go beyond the level of packet inspection. The system should, for example, know the concept of a network, a device or a user. And that a device is located in a network and a user can be logged into a device. The data collected by the system should be put into those concepts. Based on this, it will be easier for the administrator of the network to

define rules which represent a set of network states and if the network is currently not in one of those states, an alarm can be raised.

R2: Event triggered – The next requirement is being event triggered. Event triggered means that the logging of a security violation or misconfiguration is triggered as soon as it occurs. This is important for a variety of reasons. First, an alert needs to be raised in temporal vicinity of the occurrence of the event. This is because for an alert to have any value you need to trigger a response that deals with the event associated to the alarm. And the sooner a reaction is triggered, the more likely it is to succeed. The triggering of a response is not in the scope of this thesis, it still needs to be considered, though. Secondly, if the capturing wasn't done event triggered, possible violations could occur between two logging instances and therefore not be recorded.

R3: Low false positives rate – For a system that detects problems in a network, it is very important that there are as few wrong alerts as possible, because the alerts normally need to be processed. This can either be done by the system administrator by hand, which gets more time consuming and, therefore, expensive with an increasing rate of alerts. Or responses can be triggered automatically, which needs more resources with increasing alert rate, and in addition a response that is triggered because of a false alert might harm the system. Therefore, it is important to have a low false positives rate.

R4: Global view – The next requirement is for the system to have a global view of the network. This means that the mechanism that decides whether to raise an alert or not needs to have access to all data that gets collected in the network. From this data it can construct a global view, which correlates all the limited views that the distributed sensors, which collected the data, have. From this correlated view the mechanism might get additional information.

R5: Performance – The last requirement is to be performant with increasing network traffic. Therefore, it is useful to rely as little on the analysis of network traffic as possible. There are two disadvantages to a system that relies heavily on the examination of the complete traffic that traverses the network. Firstly, the more traffic accumulates, the more computational power is needed to analyze all of it. Secondly, the system is susceptible to attacks that attempt to overload the system by generating huge amounts of traffic. In order to always analyze the most recent traffic, it needs to drop packets in between. In this time frame the actual attack can be launched, in the hope that the system will drop the packets belonging to the attack from inspection. The information otherwise gained by analyzing every network packet should be partly gained by having a global view on the network.

Chapter 3

Related Work

To begin our evaluation of the related work, we will take a look at the different types of `Intrusion Detection Systems(IDS)` that are currently available. We will then evaluate some IDS- and monitoring system implementations showing their respective advantages and disadvantages. Generally, IDS are used to detect suspicious or malicious behaviour of hosts, whereas monitoring systems are used to monitor network and host resources. The systems discussed will then be assessed in respect to the requirements defined in the previous chapter in table 3.1. Lastly there will be an overview over which of the security violations and misconfigurations as defined in table 2.1 each of the presented systems covers. This is depicted in table 3.2.

3.1 IDS types

There are two big types by which you can classify an IDS. An IDS is either a `Host-` or `Network-IDS`. In addition there are hybrid solutions which incorporate parts of both types. A Host-IDS is responsible for detecting intrusions only on the host it is running on. The detection is done by collecting data from the host-machine itself such as the running file system, system-calls and operating-system- or application-logs. Because of this it can easily detect attacks by an insider like the local installation of malicious software or the access to restricted files. A drawback of this system, however, is that it needs to be installed on every system it should monitor. For big networks this can be a pretty significant problem. A Network-IDS on the other hand is responsible not only for one host but for the complete (sub-)network. It is mostly deployed on ingress or egress points of a network. A Network-IDS sniffs the data it needs from the network traffic. Here for it uses mostly the headers of network- and transport-layer packets. To maintain a high detection rate, every packet that traverses the Network-IDS needs to be analysed. [26, 27]

You then can differentiate each type further into either a `signature-` or `anomaly-based` IDS. In a signature-based IDS there is a previously initialized database with known attack patterns. Each packet gets compared to this database and raises an alarm if there is a match. The advantages of this type are that you can reliably detect known attacks and that there are no false-positive alarms. The downsides are that you need to keep an up-to-date database and you can only detect previously known attacks. To circumvent these disadvantages there are the anomaly-based IDS. They do not rely on a signature database. They model the network with normal behaviour. Then they compare the current state of the network with that of the model. An alarm is raised if the network state differs from the one of the modelled network by a specific amount. The advantage here is that, in theory, you are able to detect previously unknown attacks, but at the same time you increase the rate of false-positive alarms raised.

Furthermore, you can differentiate between the IDS being active or passive. A passive IDS is only responsible for detecting an attack. This can be done in real-time or long after the attack has occurred. An active IDS on the other hand is responsible for detecting and preventing an intrusion attempt. Therefore, active IDS are also known as `Intrusion Prevention Systems(IPS)`.

We will not consider anomaly based IDS for this evaluation, since we have the assumption that we have a known and fairly static network. In this scenario it is easier to use a rule based approach, since the deployment of an anomaly based system would take more time, since it needs an initial learning phase. During this phase the network is exposed to possible attacks. In addition anomaly based IDS have traditionally a high false-positives rate.

3.2 IDS implementations

In this section we will take a look at three different implementations of IDS and IDS-like systems.

3.2.1 SNORT

An implementation of a signature-based network IDS is `SNORT`. SNORT is one of the most popular IDS available. [28] It relies completely on inspecting packets and comparing them to attack signatures. There are several drawbacks of this implementation which come with the type of the IDS.

Firstly, if you want to define rules based on which traffic should get analysed, they can become fairly long and complex. Secondly, SNORT can not deal with encryption. As soon as it detects encrypted traffic, since it is obviously not able to decrypt it, it simply ignores it completely for performance reasons. Thirdly, it scales badly with

increasing network traffic, which means the more packets traverse your network, the more computational power you'll need. This is caused by the need to compare every packet against the rules and then every packet that matches a rule to the complete attack signature database. Additionally, with increasing traffic the system needs more memory to store states, which are needed to match packets to their TCP connection.

Another weakness is that it is vulnerable to attacks on the observed protocol, which means that you can, for example, overload SNORT with a DoS attack against the network to an extend where SNORT has to drop packets in order to process the latest packets. Then you can try to launch an attack against a specific entity and hope that your packets get dropped from inspection and the attack can execute undetected. The last point is that small modifications of known attacks might lead to not being detected since SNORT compares every packet which is selected for inspection strictly with all known attack signatures. [29]

3.2.2 ESUKOM

Another related work, which is not strictly speaking an IDS but an access control system is called `ESUKOM` [30]. However, it relies heavily on metadata and is, therefore, quite interesting in this context. It is designed as an access control mechanism for mobile devices in a corporate network. Before it grants or denies access it collects and analyzes certain metadata about the mobile device.

For example, which applications are installed at the moment, which permissions these applications have and what their version is. In addition the version and type of the running operating system is checked. They even collect data from sensors, such as the GPS location or the acceleration of the mobile device. The data gets collected and transferred to a centralized server, called `policy decision point`, using the `IF-MAP` protocol. Here the decision concerning the access is reached based on some predefined rules. The decision is forwarded to the `policy enforcement point` which can, for example, be a switch or a router. Here, the access is granted or denied. [31]

3.2.3 ONTIDS

The last related work we will discuss is `ONTIDS`. ONTIDS is an alert correlation framework, which attempts to combine alerts from traditional IDS with metadata about the alert itself. An example would be when the alarm occurred, the configuration of the host where it occurred, the role and privileges of the user who triggered it, the location of the device that triggered it or the criticality of the entity affected. This is done in order to provide some context on attacks to the administrator of the system or to the system that processes the alerts and initiates the responses.

The combination of alerts with the collected metadata is done via ontologies, which are comparable to what we use in our system. There are four different types of ontologies: attacks, alerts, contexts and vulnerabilities. An `attack` is an attack executed against the monitored network. It can have one of multiple dimensions, e.g. be a worm or a trojan. It can trigger an `alert`, which is either generated by a host IDS or by a network IDS. The alert happens in a `context`, which is the current state of the attacked entity. The state is reflected in the collected metdata as explained earlier. A context in turn has a specific `vulnerability`, which can be, for example, phishing or social engineering. The context, to close the cycle, can be exploited by an `attack`.

In conclusion, ONTIDS basically adds two more dimensions to the classification and detection of attacks done by IDS, which normally only consist of the `alert` and `attack` ontologies. [32]

3.3 Monitoring tools

In this section we will take a look at four different monitoring tools. They are responsible for collecting data from hosts in the network and the network itself. In contrast to IDS they do not throw alarms based on detecting a known attack signature in the network traffic. If they are configured to raise alarms, they mostly do this based on a certain value of a data set they collected passing a certain threshold.

3.3.1 Nagios

Nagios is a network monitoring system. It consists of a list of predefined plug-ins and can be extended by writing custom plug-ins in a lot of popular programming languages. A deployed Nagios system usually consists of a central host, which is responsible for collecting all monitoring data. It is connected to a database where the data gets stored. The collection of the data is either done by the central host, which is requesting or probing for data from hosts in the network by using the `FTP`, `UDP` or `SNMP` protocol; Or the data is collected by so called `Nagios Remote Plugin Executors(NRPE)`. NRPEs are plug-ins which need to be installed on each device which should be monitored. They collect the data locally and for each property they should monitor, they send a status back to the central host. The status can take one of four states: `OK`, `WARNING`, `CRITICAL` or `UNKNOWN`. More detailed measurement results can be added as additional data, they aren't involved in the process of creating alerts, though. The execution of plug-ins is mostly done in preconfigured time intervals. There are only two cases where plug-in executions are event triggered. Firstly, when a property in the database which is associated with a device changes its value. Then the plug-in which is installed on the associated device is polled for new data. Secondly, when a Nagios intern checks needs

a certain property of a device [33].

Alerts are generated based on the status values of the properties: if they stay in an undesired state for a certain amount of time an alert will be dispatched. Alerts can trigger events such as sending the information to the administrator or triggering responses. Two disadvantages of Nagios are: Firstly, that it doesn't distinguish between different types of hosts, for example, between routers, switches and firewalls. Secondly, the status of a property gets determined by looking at hard-coded thresholds or ranges. It lacks the option to determine those states dynamically based on other properties. [34]

3.3.2 Nmap

`Nmap` is short for `Network Mapper`. It is a tool which can be used for network discovery and security auditing. It sends packets into the network in order to retrieve information about the hosts in the network. It can be used to either target a whole network or a single host. The information that can be retrieved contains such things as the IP and MAC address, the services running on the host and the operating system used. Certain information such as the IP and MAC address and open ports can be directly extracted from the packets Nmap receives as answers to his requests. Other information such as the services running can be obtained by comparing the open ports to a database mapping port numbers to services. The operating system is obtained by comparing certain informations about the host called `fingerprints` to a database. The fingerprints are obtained by, for example, measuring the responses of a host to different types of TCP/IP packets. Since each operating system uses other response mechanisms and timings, this is often successful. [18]

However, Nmap is not a tool which supports automatic monitoring of a network or the logging of its output. It can be used by a network administrator to manually detect problems and vulnerabilities in a network or it must be used in conjuncture with another application which provides the functionality to automate the monitoring.

3.3.3 Munin

Munin is a network monitoring tool which collects information from hosts in a network and illustrates them using graphs. Munin has a client server architecture. The server is a database which stores all the data collected by the clients. It is not a snapshot of the network at the current moment. It contains a history of the network and its properties. The granularity of the data decreases proportionality to its age. For example, the default granularity of data with an age of one day is five minutes. If the age reaches one week, the granularity goes down to 30 minutes.

The clients are the devices which should be monitored. The data is being gathered

on the client by so called plug-ins. The server polls the clients in regular intervals for new data. This polling triggers the execution of the plug-ins. The default value for the polling interval is five minutes. If the value of a property, collected by a plug-in, reaches a certain threshold a warning can be generated. [35]

3.4 Comparison to requirements

In this section we will compare the implementations discussed in the last section with regards to the requirements defined in chapter 2.

SNORT fulfils R2 and R3, since it processes packets in real time and signature based IDS generally have a low false positives rate. It does not fulfil R1, R4 and R5 however. R1 is not fulfilled since SNORT works only on the network-package-level. R4 is not fulfilled because different SNORT instances can be deployed at multiple locations in the network. This way you can cover the whole network, but each instance is only able to see and analyze the traffic which passes its network segment. R5 is not fulfilled because SNORT obviously relies heavily on inspecting the network traffic.

ESUKOM fulfils every requirement. It is, however, used for a different purpose than what our goal is, since it does not monitor the state of the network. It is used exclusively for access control of mobile devices.

ONTIDS fulfils R1 through R4. R1 is fulfilled because ONTIDS knows the concept of, for example, a host, a network and a user. R2 is fulfilled because it uses data from other IDS solutions, which should be accessible in real time, and correlates it with data from other sensor that can be gathered in real time such as the user privilege level. R3 is fulfilled since this was the goal of ONTIDS and it only applies the additional knowledge gained in order to decrease the number of alarms raised. R4 is fulfilled because all data from all IDS and sensors is gathered and processed at a centralized `Alert Correlation Unit`. R5 is not fulfilled since it relies on network IDS to collect data and only uses metadata, that can be gathered without inspecting network traffic, as additional parameters in the decision process.

Nagios fulfils all requirements but R4 and R2. R1 is fulfilled because it has high level concepts implemented like a host or a user. R3 is fulfilled because there is a concept of a `hard state` and a `soft state`. A property is in a soft state when it only changed for a short period of time. It is in a hard state if the change of the property is persistent. Alerts are only raised if a property is in a undesired hard state. R5 is fulfilled because Nagios only actively inserts packets into the network and listens for their answers, and Nagios collects data about hosts with plug-ins. R2 is only partly fulfilled because the checks done by plug-ins are mostly executed in predefined time intervals. Nagios only partly fulfils R4 because even though it has a central host on which the data can be

evaluated, determining which status a certain property has is done by the local device on which the data is gathered. This applies if you use NRPEs, which run locally.

Nmap fulfils R1, R3 and R5. R1 is fulfilled since Nmap can extract things like the operating system, running services and their versions as well as classifying devices into categories such as router, switch or firewall. R3 is fulfilled because Nmap only sends packets into the network and parses the responses. The only analysis that is done by Nmap are things like service and operating system detection. Since this is done by comparing signatures to a database, it is very accurate. If it is not sure if the result is correct, it adds a reference that the results may be less accurate. Those results can be ignored in order to have a low false-positives rate. R5 is fulfilled because mostly only the packet headers need to be analyzed. Additionally Nmap is not contantly running and, therefore, only needs to monitor packets for a certain time frame each time a scanning process started (until the results are received or an internal time out occurs). R2 not fulfilled since Nmap needs to be executed manually except if it is embedded in another application which decides when it should be triggered. R4 is only partly fulfilled because Nmap can only collect data from the point of view of the location of the host on which it gets executed in the network.

Munin fulfils R1, R4 and R5. R1 is fulfilled because it collects informations such as devices and processes running on devices. R4 is fulfilled because it has a client server architecture, where the clients can be deployed at arbitrary locations throughout the network. The server then collects the data from the clients and can construct a global view from this data. R5 is fulfilled because data collection is done in time intervals not based on traffic. R3 is only partially fulfilled because Munin only considers the thresholds of single values when generating warnings. There is no correlation done between different values. R2 is not fulfilled because the server polls for data from the clients every five minutes by default. This can be changed to a smaller interval, but the granularity of the data store in the database won't be changed by this, because if the polling interval is smaller than five minutes, Munin will calculate the average of all values for a time span of five minutes and only store the average.

	SNORT	ESUKOM	ONTIDS	Nagios	Nmap	Munin
R1	-	+	○	+	-	+
R2	+	+	+	○	-	-
R3	+	+	+	+	+	○
R4	-	+	+	○	○	+
R5	-	+	-	+	+	+

Table 3.1: Comparison of the systems discussed in this chapter with regards to the requirements from section 2.5

	SNORT	ESUKOM	ONTIDS	Nagios	Nmap	Munin
Network topology						
A1	+	+	+	+	+	+
A2	+	+	+	+	+	+
A3	-	-	-	+	-	+
A4	-	-	-	+	-	+
A5	+	-	○	-	+	○
A6	+	-	○	-	-	○
A7	+	-	-	+	-	+
A8	+	-	+	+	-	+
A9	-	-	-	-	-	+
Network services						
A10	-	-	-	+	+	+
A11	+	-	+	-	-	-
A12	+	-	+	○	-	○
System security						
A13	-	+	+	+	+	+
A14	-	+	+	+	○	+
A15	-	-	+	+	-	+
A16	-	-	-	+	-	-
A17	-	-	-	-	-	-
System reliability						
A18	-	-	-	+	-	+
A19	-	-	-	+	-	+
A20	-	-	-	+	-	+

Table 3.2: Comparison of the datums and the related work

Chapter 4

Design

In this chapter the design of the proposed model will be outlined. This includes an overview of the complete system as well as closer inspection of each of its components and their interactions with each other.

4.1 Components

The model we propose consists of the components displayed in Fig. 4.1. At the center of our model there is a `Database`. The data filling the database is collected by the `Devicescanners`. The database exchanges data with the `Analyzer`, which is responsible for checking the data against a set of pre-configured rules and raise an alert in case one of the rules gets violated.

We distinguish between the `Global Devicescanner` and the `Local Devicescanner`. The `Local Devicescanner` is deployed on every device registered in the network that is able to run the devicescanner and that should be monitored. The local devicescanner is responsible for collecting data which is only accessible from within the device the data is compiled on, like which processes are running on the device or which users are currently logged into the device. Deploying the local devicescanner on every device is a pretty invasive behaviour and additionally not every device, that should be monitored, might be able to run the devicescanner because of restrictions on processing power or memory capacity. Therefore, we try to have as much of the scanning as possible done by the global devicescanner.

In contrast to the local devicescanner, the global devicescanner can be deployed at multiple and arbitrary positions in the network. The work of the global devicescanner is twofold. It is responsible for the collection of data from the network itself as well as data about devices in the network, which is accessible from outside a device. Collecting data from the network itself contains active as well as passive scans. An active network

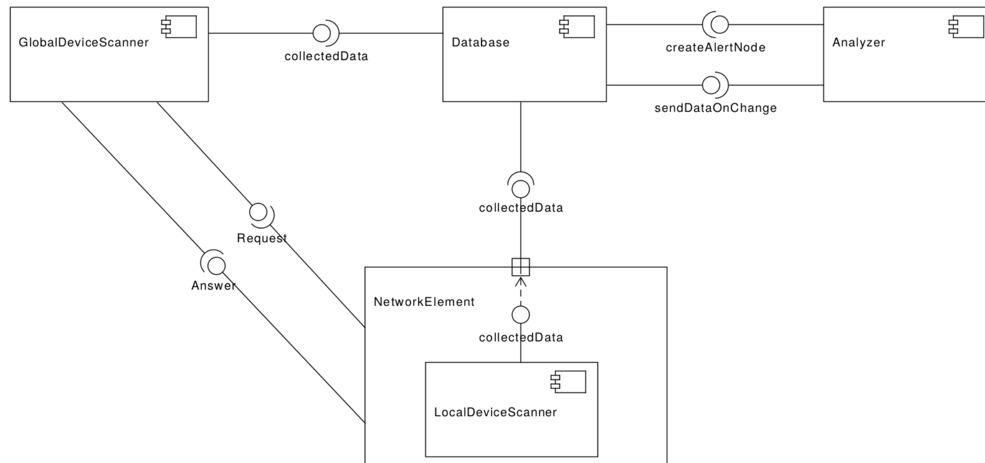


Figure 4.1: UML component diagram of our model

scan would be, for example, sending `DHCP-discvery` packets into the network to learn which DHCP servers are active in the network. An example for a passive scan would be the recording of ARP requests exchanged between devices. The collection of data from devices can, for example, be done by using `SNMP(Simple Network Management Protocol)`. You can send SNMP request to a device, which if it supports SNMP, responds with an SNMP answer containing the requested data. There is a predefined list of requests [36], for example, you could send a request for the content of the ARP cache of the device.

The exact sequence of interactions between these components is depicted in Fig. 4.2. Here we see that the `Local Devicescanner` first collects a set of data. In parallel the `Global Devicescanner` sends a request to a network device, for example, over SNMP for some data. The network device answers with a response which contains the requested data. Then each of the two scanners sends the data it collected to the `Database`.

The `Database` in turn writes the changes to its dataset. Then it forwards the portions of data which were affected by the changes to the `Analyzer`. The `Analyzer` parses the data and checks it against a previously installed rule set. If it discovers the violation of one of the rules, it creates a corresponding alert node in the `Database`. The alert node contains all relevant information of the security violation such as the type of violation, the target and the attacker.

An example would be: the devicescanners collect data about the network and the devices in the network, among other things which ports are currently open. They then send this data to the database. For the IP address of a web server the scan resulted in the fact that only tcp port 80 is opened at the moment. In the database however, the corresponding

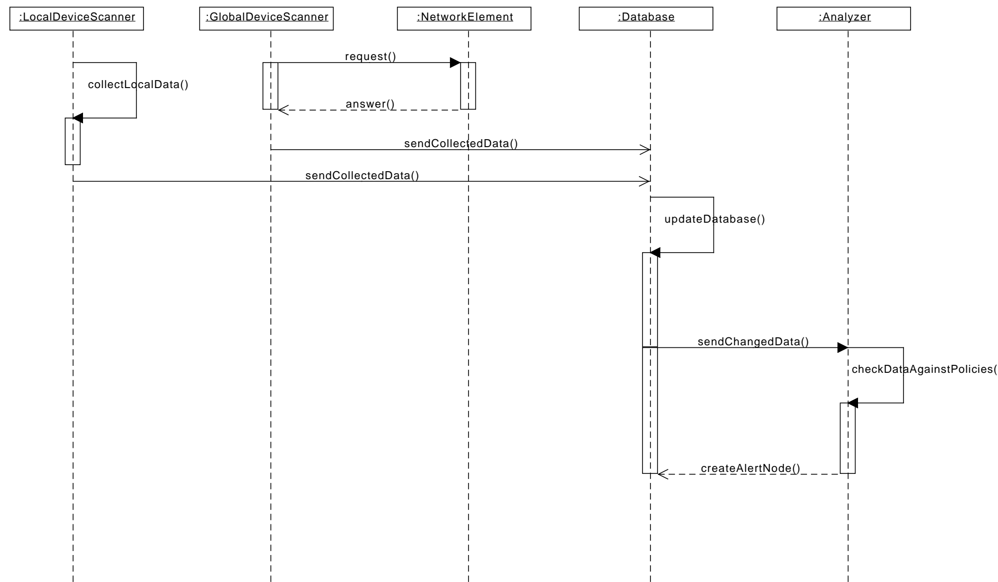


Figure 4.2: UML sequence diagram of a use case where a security violation is detected

Port	Protocol	active	config
80	TCP	true	required
443	TCP	true	required

Table 4.1: Example ports and their properties

IP address node has two ports registered on it as depicted in table 4.1. The scans resulted in port 80 being opened and the database states the same. Therefore, nothing will be changed. However, according to the scans Port 443 is closed. Therefore, the `active` property of port 443 will be set to false. Because changes occurred in the database, the affected data will be sent to the analyzer. There should be a rule installed that states that if the `active` attribute is set to false and the `config` attribute is set to required, there is a security violation or misconfiguration somewhere. As a result the analyzer will create an alert node in the database which states that port 443 on the web server is closed, which it should not be.

The collection of data by the two devicescanner types will be time-triggered for the active modules. This means that there is a pre-configured interval at which they start a new scanning process and report the results to the database. A more precise way of doing this would be to start the scanning event-triggered instead of time-triggered. This means that an event such as the opening of a new port starts a port scan or a new entry in the ARP cache starts a scanning of the ARP cache. This is better since you do not miss out on any information because then a port can not be opened and closed in

between two scanning intervals. However, it is also more difficult to implement and more resource consuming in operation. The passive modules of the devicescanners will be running constantly.

4.2 Graph Model

As seen in Fig. 4.3 we decided to model the network and all its components as a graph. Generally, the nodes are used to model entities of the network and the edges to display the relationships between those nodes. The two only exceptions are the `Flow-` and the `Connects`-node which are representing relationship classes (e.g. a relationship modelled as a node).

In the following paragraphs after each node there will be listed a set of security violations or misconfigurations, as specified in chapter 2, in parenthesis. They map the security violations to the nodes that are required to detect them.

To begin the description of the graph model we will take a look at all the nodes and edges representing the network topology, starting with the `Device`(A5, A6) node. A `Device` is a physical entity of the network. Each `Device` has a set of `NetworkInterfaces` attached to it. Each of those `NetworkInterfaces` has a `MacAddress`(A1, A4) registered on it, which in turn corresponds to one or more `IpAddresses`(A4, A9). Each `IpAddress` has a set of open `Ports`(A2) registered on it. The connectivity between `Devices` is modelled via the `Layer2Network` and `Layer3Network`(A8, A4) nodes, which are connected to the `NetworkInterface` and `IpAddress` respectively. Those two networks in turn are both merging into one layer independent `Network`. Each `Layer2Network` may be connected to another `Layer2Network` via a `Device` of the network. The same holds for `Layer3Networks`. In case of two `Layer2Networks` the connecting `Device` would be a switch, in case of a `Layer3Network` it would be a router. In addition each of the three network types can be a subnet of a bigger network via the `partOf` relation onto itself. Network traffic is modelled by the `Flow`(A7, A12, A18) relationship, which connects two `Port` instances, one being the flow source and one the flow destination of the traffic.

The remaining nodes are used to specify properties of the local device. On a `Device` there is a `Binary`(A13, A14) installed and a `Instance`(A13, A14) running. The `Binary` is executed by the `Instance`. The `Binary` is the program which is being executed at the moment. An `Instance` provides a specific `Service`(A13). The service is a more abstract concept like a mail-service whereas the `Instance` is a specific instance of that type that is deployed on the device. Each `Instance` listens on a specific port. To stay with the example of the mail-service this would be a SMTP or POP3 instance listening on port 25 or 110. A `Device` has also an `ArpCache`(A4) which in turn has some `ArpEntries`(A4), which consist of a IP- and MAC-address tuple. A `Device` has also a `UsbBus`(A17) with some additional properties which indicate the type of device which is connected to

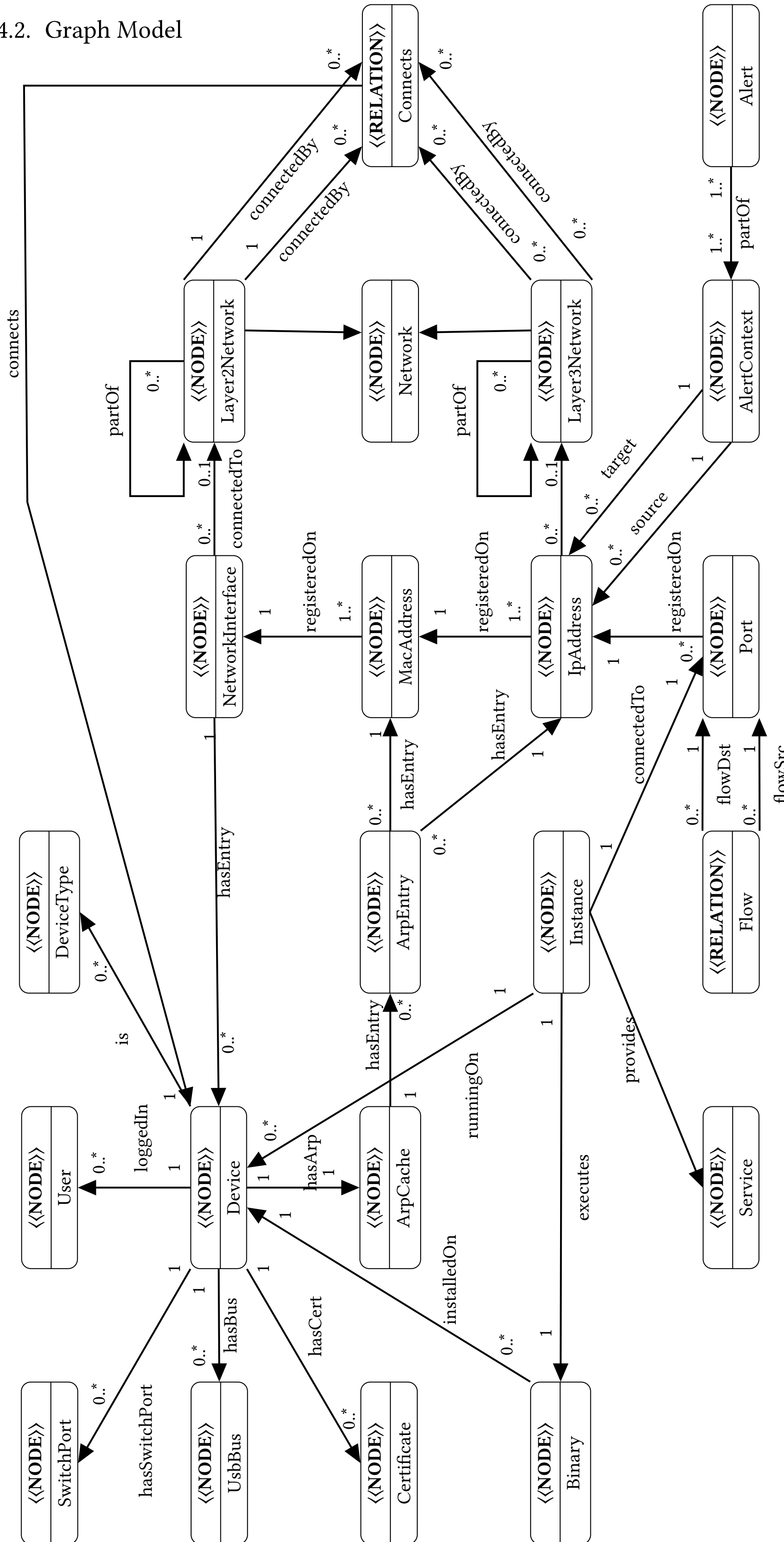


Figure 4.3: Diagram of the graph model

this USB port, for example, a keyboard or a web cam. The `User` node of a device specifies the users which are currently logged into the device. The `DeviceType`(A5, A6) specifies which type of device this is, for example, a router or a switch. Finally the `Certificate`(A16) node lists all the certificates available on the device.

Some of the nodes and relationships will have two additional properties called `active` and `config`. The `active` property is a boolean value which is set to true when the corresponding node or relationship is currently active and set to false when it is inactive. The `config` property is an enum consisting of the following four states: `unknown`, `required`, `allowed` and `not allowed`. The `required` state means that it is mandatory that the node is active or, in other words, that the active property is set to true. The `not allowed` state means that the node is not allowed to be active. The `allowed` property means that the node is allowed to be active as well as inactive. The `unknown` state means that there is no configuration available. The difference between the `allowed` and `unknown` state is that, if a node in the `allowed` state gets set from active to inactive, the node remains in the database since it is part of the configuration. If however, a node in the `unknown` state gets set from active to inactive, it will be removed from the database by the devicescanner.

Those two properties are used to easily determine whether a node or relationship is regarded as a threat or not. This simplifies the work the analyzer has to do significantly, since it does not have to have a rule-set for every type of node and relationship. This way the analyzer only needs to check if the active-required tuple forms an acceptable combination.

4.3 Devicescanner

Fig. 4.4 depicts the details of the devicescanner. It is applicable to the local as well as the global one. You can see the database an the connection to it from Fig. 4.1 at the top. The devicescanner itself consists of a scheduler, a configuration and an arbitrary amount of modules.

The scheduler is responsible for launching the different modules at different times and in different intervals. There are active and passive modules. The scheduler is only responsible for launching the active modules, since the passive ones are always running. The times and intervals for each active module are stored in the configuration, which will be accessed by the scheduler. On top of that it is set in the configuration which modules will be launched at all.

The configuration for the time is needed so that not every module is executed at the same time, since this might overload the device, or at least affect the performance of the device at execution-time of the modules. The configuration for the intervals is needed

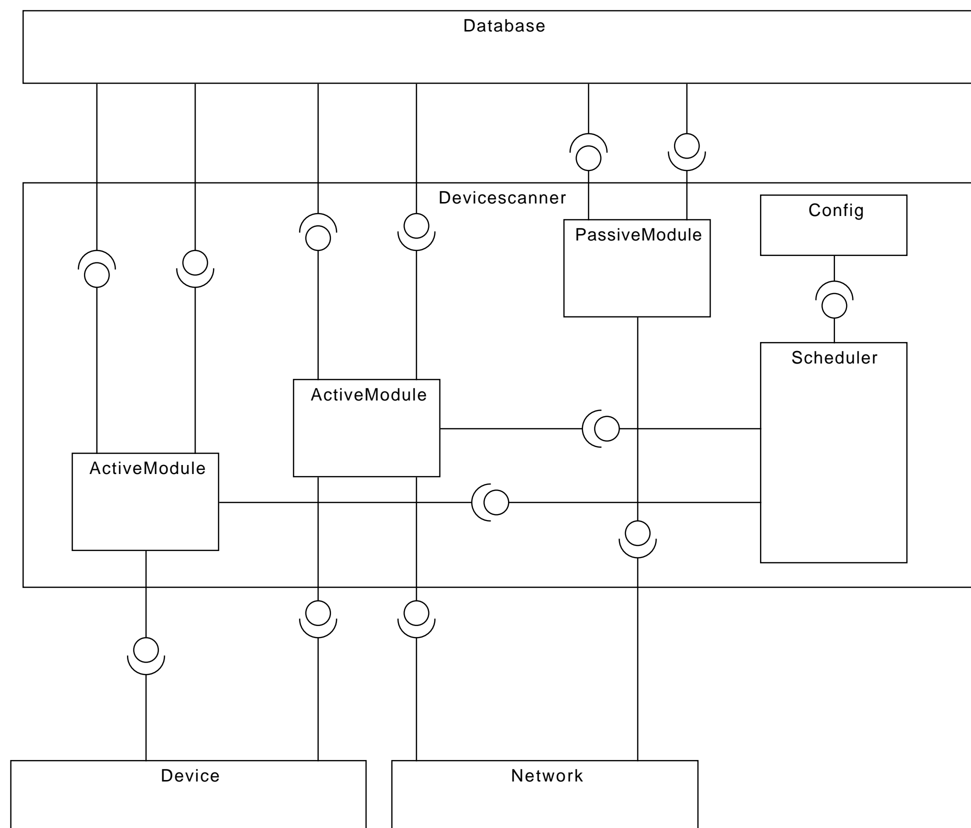


Figure 4.4: Diagram of the scanner

since some data sets need to be collected more often than others. For example, you need to check for open ports fairly often, whereas the content of the ARP cache remains relatively static for small intervals and you, therefore, only need to check it in larger intervals. Another reason for the configuration is that you are able to enable and disable modules depending on which kind of a device you are running on and which data you want to consider.

The modules itself are each responsible for collecting a specific part of the data which will be sent to the database later on. This specific data is one node of the graph model and all its outgoing relationships. Therefore, they need to access the medium from which they can retrieve this information. For the local devicescanner this would be the device itself, for example, using log files. For the global devicescanner this would be the device as well as the network wire. So, for example, one module will be responsible for collecting the entries of the ARP cache, while another one will be responsible for retrieving the currently opened ports. After they collected the data, they will directly pass it on to the database.

To make this more efficient, each module has its own local cache. This cache contains the results of the last scan. It gets updated every time a scanning process has finished. Before the module sends any data to the database it checks if this data is contained in the cache. If it is, you can avoid wasting the bandwidth of the network since the database will tell you that this data is already stored there anyway.

The two properties, `active` and `config` from the previous section are partly being set by the devicescanners. The `active` property is being set by the devicescanners at runtime, by the global as well as the local one. The `config` property needs to be set up manually beforehand, for example, by the system administrator. The only exception is the `unknown` state of the `config` property, which is the default value and which will be set by the devicescanner if it detects a new node, for which there is no configuration available in the database.

4.4 Analyzer

The analyzer will, as explained above, get any data that got changed from the database. It will then apply a set of rules, based on which type of node or relationship from the graph model was changed, to this data. Based on the outcome of the application of those rules, it might generate an alert node in the database. The rules will be written in the query language of the database to minimize the data that needs to be transferred between the database and the analyzer. This is because you might need a quite large portion of the graph in order to apply your rules properly. For example, if there was a change in the ARP cache of one device in the network, and you want to check for inconsistencies in the ARP caches across the network, you would need to get the ARP

cache of every single device in the network transferred to the analyzer. Using queries as rules, you only need to send one query over the network.

There will be a set of predefined rules installed in the Analyzer, but you will also be able to write your own rules based on your needs.

The most important basic rule that is already installed is looking at the `active` and `config` flags for each node and raises an alert if an unwanted combination occurs. There are three combinations that are undesirable. Firstly, if the `active` flag is set to `true` and the `config` flag is set to `notAllowed`. In this case the node was preconfigured as not being allowed to be active, but it is nonetheless. The second case occurs when the `active` flag is set to `true` and the `config` flag to `unknown`. In this case a scanner detected a node for which there is no configuration available. This doesn't need to, but could indicate a security violation or misconfiguration. The last case occurs if the `active` flag is set to `false` and the `config` flag is set to `required`. In this case there is definitely something wrong because the node has been preconfigured to be required to be always active, it is not, tough. All possible unwanted combinations are listed in table 4.2.

Additional rules are required in order to cover unique attributes of different networks and systems. Those rules can already be defined and you only need to adjust some values of properties based on your needs.

An example of this would be a rule specifying the maximum amount of DHCP servers per subnet. You then only need to adjust the number of servers based on your needs.

Another example could be that communication with devices in a specific subnet or the whole network is only allowed if both parties hold a valid certificate that meets certain requirements. For example, only certain certificate signature algorithms, public key algorithms or issuers are allowed. This rule makes sure that each client authenticates itself, reducing the risks of a man-in-the-middle attack on entities in the network, and it introduces certain minimum requirements for signature and encryption algorithms. For this rule to work you need to check that every `Device` node whose `Port` nodes are connected with a `Flow` node have a `Certificate` node that meets your requirements.

In addition to this, other arbitrary rules can be defined in order to add functionality to the Analyzer.

Active	Config	Status
true	notAllowed	The node is active even though it is forbidden by the configuration.
true	unknown	There is no configuration available. Could possibly be a violation.
false	required	The node is inactive even though it is required to be active.

Table 4.2: Unwanted combinations of the active and config flags

Chapter 5

Implementation

In this chapter we take a look at the implementation done in the scope of this thesis. It covers which frameworks were used and how each component defined in chapter 4 has been implemented. First we will take a look at Neo4J, which was chosen as the database solution and its query language, Cypher. Then we will take a look at how the interaction with the database is done and finally we will look at how the Devicescanners and the Analyzer was implemented.

5.1 Neo4J and Cypher

As a database solution we chose `Neo4J`, a graph database. A graph database, in contrast to a relational database, contains like the name suggests graphs not tables. Such a graph consists of nodes and relationships between nodes. The nodes and relationships can have properties and the relationships are directed. An example of an easy graph is shown in Fig. 5.1. The graph represents the like-relationship between three persons. The nodes each represent a person with a name-property, which is set respectively to Alice,Bob and Carol. The question arises why this is supposed to be better than

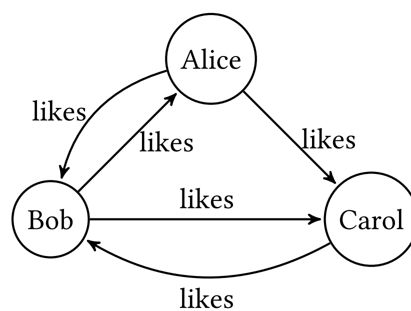


Figure 5.1: Relationship graph between three persons

PersonID	Name
0	Alice
1	Bob
2	Carol

Likes	
Person1	Person2
0	1
0	2
1	0
1	2
2	1

Table 5.1: The corresponding relational representation to Fig. 5.1

Depth	Relational	Graph
2	0.016	0.01
3	30.267	0.168
4	1543.505	1.359
5	Unfinished	2.132

Table 5.2: Performance of the relational- versus the graph-model

the traditional way of storing it in a relational database. In Table 5.1 the graph from Fig. 5.1 is depicted the way it is stored in a relational database. There are two tables required. The left one assigns a unique index to each person. The right table defines the like-relationship between two persons.

If you want to figure out who likes Alice, this is easily done in both the relational- as well as the graph-model. In the graph-model you simply need to find all paths which have length one (consisting of two nodes and one relationship) and Alice as the end node of an edge and return the start nodes of those paths. If you want to find out all persons that like another person, which in turn likes Alice, it gets more complicated in the relational model. You already need three computationally expensive `JOIN` operations to find the result. In the graph database you only need to go find all paths with length two which have Alice as an end node and return the starting node of this path. The performance of the likes relationship up to a depth of five, on a graph- as well as a relational-model, have been evaluated in [37]. The dataset used has a size of one million people with an average of `50 likes` relations per person. Table 5.2 shows the results. As you can see a graph database is the clearly superior choice if you need to do operations even only over three relationships.

The query language used to pose requests to the database is called `Cypher`. Cypher is easy

```
(a)-[:likes]->(b)
```

Figure 5.2: Cypher representation of a simple graph

```
MATCH (a)-[:likes]->(b)
WHERE a.name="Alice"
RETURN b
```

Figure 5.3: Cypher query to find the set of persons Alice likes

to learn and to read, since it uses an ASCII-art like syntax. The Cypher representation of a simple graph consisting only of two nodes connected by a directed relationship of the type `likes` is shown in Fig. 5.2. A query which returns us all people that Alice likes from the graph in Fig. 5.1 looks like shown in Fig. 5.3. You might notice that the syntax, except for the ASCII-art, is very similar to the syntax of SQL. The MATCH command specifies what the path we are searching for looks like. In this case it searches for two nodes connected by a `likes` relationship. The WHERE clause specifies further details about the path, in this case That the property `name` of the starting node should be Alice. The RETURN statement returns the end nodes of all the paths that matched the WHERE clause. Thus this query should return us Bob and Carol, since there exists a path in the graph on which Alice is the starting node and Bob the end node, as well as a path where Alice is the starting node and Carol the end node. [37] The data will be written to the database by using the `Spring Data Framework`, which is generating Cypher queries from Java code and handels connections with a Neo4J database. However, Cypher syntax is important for the Analyzer later on, which is using native Cypher queries.

5.2 Database

Each node of the graph model is mapped to a separate Java class with the corresponding name. They all extend the `VitsiGraphNode`-class. This class specifies basic properties that are true for every node of the graph model. This includes the ID used by the graph database, the UUID of the node as well as the `active` and `config` flags. Each class that represents a node specifies the properties of this node as well as all relationships it has to another node. For example: the `MacAddress` class has one property specifying the name of the node, which corresponds to the string representation of the hardware address. In addition it has two relationships: an outgoing one to a `NetworkInterface` instance and an incoming one from a `IpAddress` instance. Fig. 5.4 shows an abstracted graphical representation of the concept.

Each class has an interface associated with it, which is called `repository`. The repository is responsible for deriving cypher queries from Java methods. The interface extends

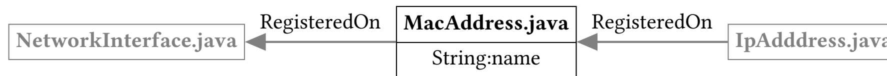


Figure 5.4: Class representation

```
MacAddress findByName(String name);
```

Figure 5.5: Find method in the MacAddressRepository

the `GraphRepository` class (parameterized with the class of the node). From this class it inherits its ability to derive cypher queries and a lot of standard CRUD(create read update delete) operations. You are now able to specify methods to, for example, search the database. You simply need to provide the return type of the method as well as its parameters. The repository will then derive a cypher query that searches for nodes of the return type of the method which match the provided parameters. [38] For example, in the `MacAddressRepository` interface there is the method from Fig. 5.5

This method will return a node of type `MacAddress` where the property `name` matches the String passed as an argument.

5.3 Devicescanner

The devicescanner is implemented as a Java class. As described in chapter 4 each devicescanner is partitioned into different modules. Each of those module is implemented in a separate Java class as well. The modules will be executed in different intervals, which are defined in an external configuration file. For this purpose, the devicescanner class obtains a `Cached Thread Pool`. This can be used to schedule the different modules to be executed in different intervals. You simply need to pass an instance of the class `Runnable` and the interval at which it should be executed as well as the delay before the initial execution. The devicescanner class then defines a separate `Runnable` for every module that should be executed. It then creates an `Executor` for the thread pool and schedules each `Runnable` with its corresponding interval and its initial delay.

Each module implements three methods used to extract the data and write it to the database. The first method is the `read` method. This method is in charge of extracting and returning the data for which the module in question is responsible. The second method is called `insertIntoDb`. It is responsible of writing the data that has been collected by the `read` method into the database. The third method is called `cleanDb`. It is responsible for removing any outdated information from the database.

The `read` method is called every time the module is used to scan the device. Since most modules rely on executing a bash command in order to obtain the data they require, the

read method is normally structured as follows. At first an instance of the class Runtime is created. This is used to execute the command. The parsing of the output of the result is either done directly on the commandline itself by using pipes and commands such as cut , grep and sed, or it is done using Java functionality after the output of the command has been read in. Next an Object of the required type is constructed and initiated with the values obtained by the executed bash command.

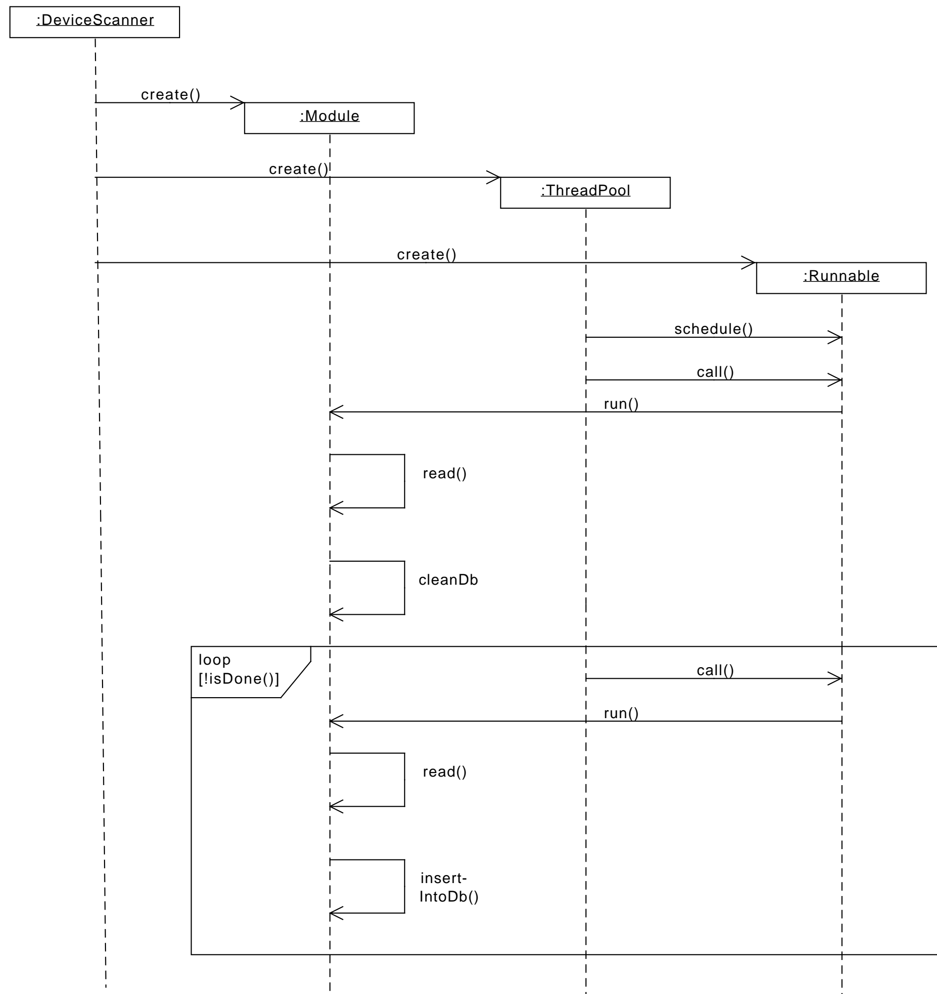
In contrast to the read method, the cleanDb method is only called the first time the module performs a scan. It is responsible for deleting any outdated information left in the database. This is only the case if the devicescanner got restarted and something in the data collected by this module changed during the down time of the scanner. More concretely, this method pulls every node and relationship from the database for which its module is responsible. It then compares this data to the results returned by the read method and deletes every node and relationship which is present in the database but not in the scan, and therefore, is outdated.

The insertIntoDb method is called on every consecutive scanning process this module performs. Like mentioned earlier it is responsible for inserting the data collected by the read method into the database.

To minimize the network traffic between this method and the database a local buffer is used. The buffer contains the results of the last scanning process. On the first iteration this buffer is empty. Therefore, the method simply attempts to write everything from the scan into the database. On each consecutive iteration the buffer will be filled, though. When this is the case, the method compares the contents of the buffer to the contents of the scan. It only writes the difference to the database (the relative complement between scan S and buffer B: $S \setminus B$). After the insertion into the database is done, it will delete every node and relationship that is in the buffer but not in the scan ($B \setminus S$) from the database. This is done, because this is data that changed between the previous scan and this one and is therefore outdated.

However, in order to comply with the concept of the active and config flags introduced earlier, the process of insertion into the database and the deletion from it has to follow a few additional rules. If you want to insert a node into the database and it is already present, you need to check if the active flag is set to true. If it is set to false you need to set it to true. If you want to delete a node from the database, you only remove the node completely if the config flag is set to unknown. If it is set to any other value, you do not remove the node. You only set the active flag to false. The complete process described above is depicted in Fig. 5.3.

The local devicescanner can be executed as an unprivileged user. For scanning the local ports, however, it needs root privileges. If you do not want to run the entire devicescanner as root, you have two options. You can either set the setuid bit for the lsof tool which is used for the scanning with the following command: sudo chmod


:DeviceScanner
create()
:Module
create()
:ThreadPool
create()
:Runnable
schedule()
call()
run()
read()
cleanDb
loop
[!isDone()]
call()
run()
read()
insert-
IntoDb()

u+s $(which lsof). Alternatively you can add an exception to the sudoers file, which allows to execute sudo commands without password by adding the following line to the /etc/sudoers file: ALL ALL = NOPASSWD : /usr/bin/lsof.

The same is true for the global devicescanner if you want to scan the network for DHCP servers. Here you need to give root privileges to the dhcping tool, because it needs to bind itself to port 68 and send out packets over this port (sending packets from every "well-known-port", which is each port with a port number of less than 1024, needs root privileges).

5.4 Analyzer

The analyzer is, like discussed earlier, responsible for analyzing every change made to the database and decide if a security or configuration policy has been violated. The policies are implemented as a set of cypher queries, which get executed against the database. Like the devicescanner, the analyzer is implemented as a separate Java class. It consists of the genCypher and checkChanges methods and an inner class called Noftifier.

The Notifier class extends the Thread class. When the analyzer gets executed, a Thread is created from a Notifier instance. This thread gets started and runs as long as it is not externally interrupted.

The Notifier class is responsible for polling the database for changes and if a change occurs, triggering a reaction. To realize the polling of the database it makes use of a modified version of the ChangeFeed module from the GraphAware framework. [39] For the changefeed module to work, there needs to be a compiled jar file of the grapahaware framework as well as the used changefeed module in the plugins directory of the Neo4J installation. Every time the a change is made to the data in the database, this change is logged by the changefeed module. On the first start of the Neo4J database, it creates a GA_ChangeFeed node in the database. This node has a moduleId field which holds the id by which it can be referenced later, when the database gets polled for changes. Every time any data in the database gets changed, the changefeed module will create a separate GA_ChangeSet node that holds those changes. Those nodes are chronologically linked to one another. In addition the GA_ChangeFeed module has relationships to the first and the last node in this chain. The earlier created thread is polling the database and gets notified as soon as a change occurs. If a change occurs, it calls the checkChanges method and passes to it all changes that occurred.

The checkChanges method is responsible for deciding which cypher queries get executed if a certain change in the database occurred. For that purpose every cypher query has a label assigned to it. Those labels correspond to the labels of the nodes in the database. Every cypher query can have a specific label or the label all. Cypher queries which

possess the label `all` will be executed for every change that occurs. Cypher queries with another label will only be executed if their label matches the label of the node that got changed.

As soon as a cypher query is selected for execution the `genCypher` method is called. The arguments passed to it are the cypher query that should be executed and the ID of the node that got changed in the database and triggered the execution of the query. The `genCypher` method returns the Cypher query with the ID of the node inserted as a parameter. This is done in order to avoid needing to query every node of the same label as the node that got changed. The adding of the ID is optional though, and it is decided based on the query if it is inserted or not. This is because some queries might need to execute over the whole database anyway in order to get a meaningful result. For example if an ARP entry got added and you want to check the consistency of all ARP caches across the network, you need to check the new entry against every other entry.

Chapter 6

Evaluation

In this chapter we will evaluate our system qualitatively and quantitatively. The qualitative evaluation consists of a comparison to the requirements as well as the security violations and misconfigurations defined in 2. The quantitative evaluation will measure certain properties of the implemented version as described in chapter 5.

6.1 Qualitative evaluation

R1: High level of abstraction – The first requirement is met by our system because of the graph model we use. There are a lot of high level concepts, such as the concept of a device, a network, a user, a certificate and a process. The information collected by the scanners is stored in the database encapsulated in these concepts: The Analyzer can then define rules on those concepts, which makes the creation of a rule more easy. This displays a higher abstraction level than only using packet inspection.

R2: Event triggered – This requirement is met for the devicescanners because you can trigger all scans based on events as well as based on predefined intervals. You can for example if you want to scan for open ports, constantly monitor the output of `lsof` or `netcat`, instead of polling it every couple of seconds, and as soon as a change occurs you can trigger a method to insert the changes into the database. The only disadvantage is that you need more processing power. The requirement is also met for the Analyzer because the Changefeed, which is used to retrieve changes made to the database, is constantly polling the database and as soon as a new node or relationship gets added or the value of a property gets changed it triggers the execution of the rules.

R3: Low false positives rate – The system can be used in conjuncture with a network intrusion detection system, which analyzes the traffic traversing the network. Because of the additional information, that are gained by having a global view of the network

	SNORT	ESUKOM	ONTIDS	Own solution
R1	-	+	○	+
R2	+	+	+	+
R3	+	+	+	○
R4	-	+	+	+
R5	-	+	-	+

Table 6.1: Evaluation of the requirements

topology and the network properties, the false positives rate of the IDS can be reduced by providing this data to it. The false positives rate of the system itself is directly dependent on the rules defined in the Analyzer. Because of the high level of abstraction that the rules display, it is easier to configure them correctly and , thus, it is less likely that they throw wrong alerts because of a slightly incorrect configuration.

R4: Global view – The requirement for a global view is met by our system because we have the different instances of the devicescanners which are deployed throughout the network. They collect the data and send it to the centralized Neo4J database. Therefore, we have the global view in the database. From there the Analyzer, which is responsible raising alerts, gets its information and can therefore make its decisions with all the available information from the global view.

R5: Performance – The requirement for as little traffic analysis as possible is met because monitoring of traffic is only done by some modules of the global devicescanner. However, in contrast to the constant monitoring of all traversing traffic, they send out packets into the network and then monitor the traffic until they receive an answer and then stop until they send out the next packet. Constant and passive monitoring of the traffic is not done. It might, however, be used as an extension if you want to verify the data you get from the local devicescanners but even then you don't need to analyze every packet in-depth. Monitoring the packet headers might suffice here.

The comparison to the security violations as misconfigurations as defined in chapter 2 is summarized in table 6.2.

6.2 Quantitative evaluation

The quantitative evaluation will cover the following points: the CPU and main memory consumption of the local devicescanner, the production of network traffic between the

	SNORT	ESUKOM	ONTIDS	Nagios	Nmap	Munin	Own solution
Network topology							
A1	+	+	+	+	+	+	+
A2	+	+	+	+	+	+	+
A3	-	-	-	+	-	+	+
A4	-	-	-	+	-	+	+
A5	+	-	○	-	+	○	+
A6	+	-	○	-	-	○	+
A7	+	-	-	+	-	+	+
A8	+	-	+	+	-	+	+
A9	-	-	-	-	-	+	+
Network services							
A10	-	-	-	+	+	+	+
A11	+	-	+	-	-	-	+
A12	+	-	+	○	-	○	+
System security							
A13	-	+	+	+	+	+	+
A14	-	+	+	+	○	+	+
A15	-	-	+	+	-	+	+
A16	-	-	-	+	-	-	+
A17	-	-	-	-	-	-	+
System reliability							
A18	-	-	-	+	-	+	+
A19	-	-	-	+	-	+	+
A20	-	-	-	+	-	+	+

Table 6.2: Comparison of the own solution to the security violations and misconfigurations as defined in chapter 2

devicescanners and the database with and without the usage of the local buffer and the logging delay which happens between the occurrence of an event and its detection by the analyzer.

6.2.1 Test environment

All tests have been performed with the database, the analyzer and the devicescanner located on the same device. This way there are less obstructions of the measurements by other traffic. The system specifics are as follows: CPU: `Intel Core i7-2630QM CPU @ 2.00GHz × 8`, memory: `6024 MB`, operating system: `Ubuntu 14.10 64-bit`.

6.2.2 Testing tools

For the testing we used `Conky` as well as `Collectl`. They are both programs used to measure system resources. Conky was used for the measurements of the CPU and main memory utilization. Collectl was used for the measurements of the network traffic. In addition, a custom script was used in conjuncture with a `Junit` test to measure the logging delay.

6.2.3 CPU and memory utilization

Since the devicescanner is deployed on every device and should always run in the background, it is important that it does not interfere with other processes that are running on the device as well as I/O operations. Therefore it is important that the devicescanner only uses up a small amount of the computational power and the main memory(RAM) of the device. Primarily this only concerns the local devicescanner, because the global one can be deployed on dedicated devices if need be.

Conky was used to monitor the CPU and memory utilization. There were two stages of monitoring: First the idle device was monitored and then the device was monitored while the devicescanner was running. Conky was configured to log the data in an interval of 0.1 seconds. Each of the two stages was executed five times for a duration of three minutes. The results depicted in table 6.1 and 6.2 show the mean of these five runs at each point in time. The devicescanner was running with seven active modules, running at scanning intervals of 600, 35, 30, 25, 20, 5 and 5 seconds respectively.

The idle device uses up about 3.5 percent of the power of the CPU. When the devicescanner is running, the device uses an average of about 9.4 percent of the CPU, with occasional spikes to about 15 percent. Comparing this to the idle device yields an average CPU usage of the devicescanner of 5.9 percent. The big spike at the beginning

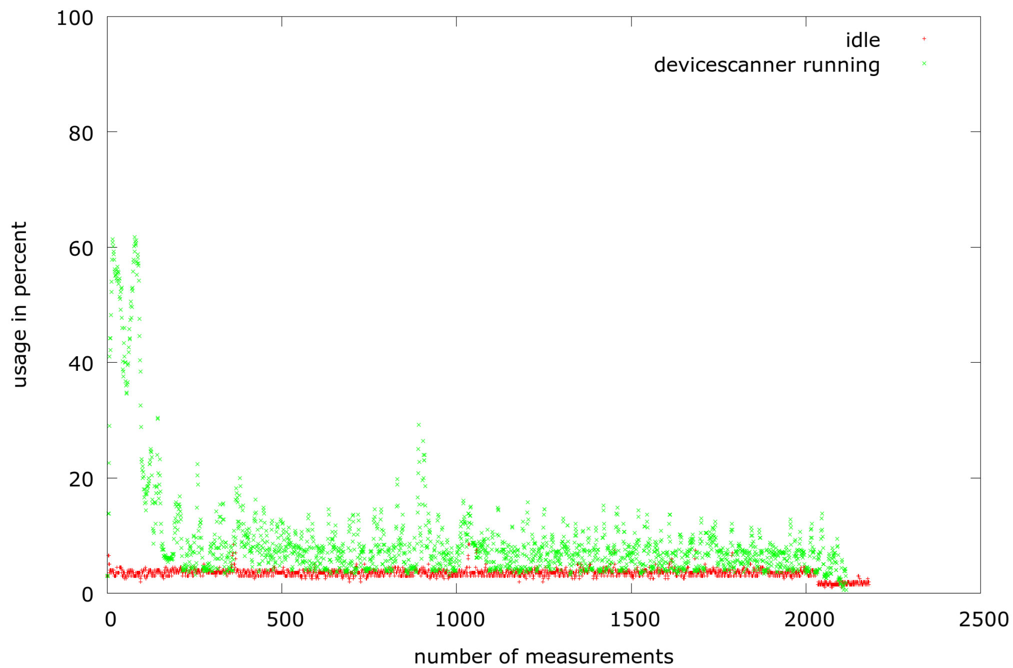


Figure 6.1: CPU usage of the idle device versus CPU usage with the devicescanner running

comes from the fact that the devicescanner has been restarted for ever measurement. Normally this should be only the case on system start up.

On average, the memory consumption of the devicescanner lies at about 540 MB.

Figure 6.3 depicts an example of what the local devicescanner had written into the database after a test run. We can see the local device (yellow node) which has two real and one virtual interface. Each interface has a MAC address registered on it, each with their respective IPv4 and IPv6 addresses . Each IPv4 address resides in its own network and has a set of open ports. Furthermore, the device has an ARP cache with one entry, one logged in user and three external devices connected to it via USB.

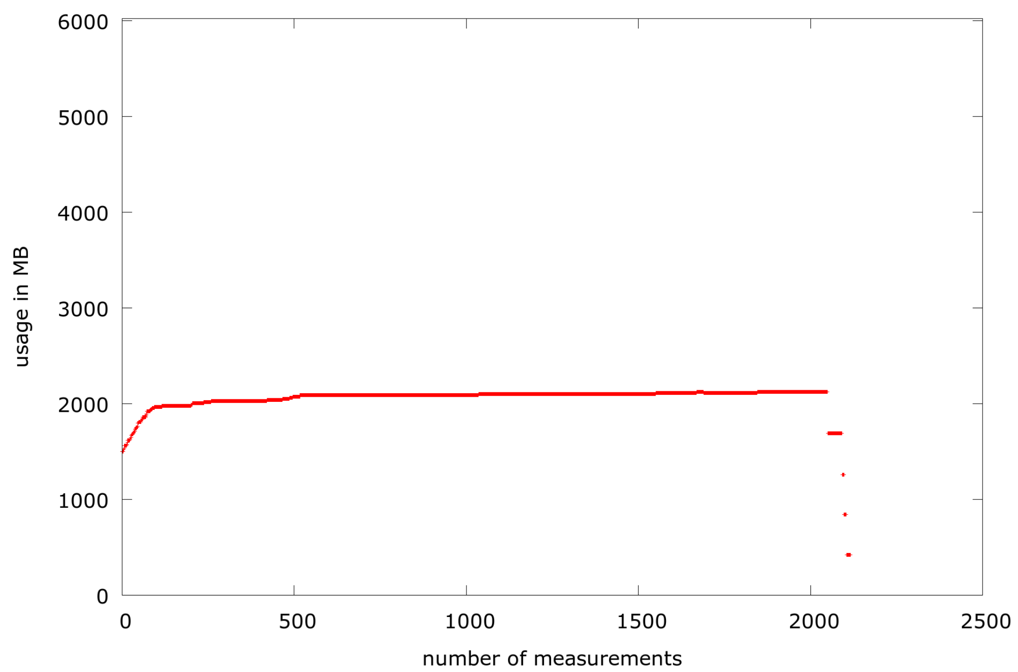


Figure 6.2: RAM usage of the device with the devicescanner running

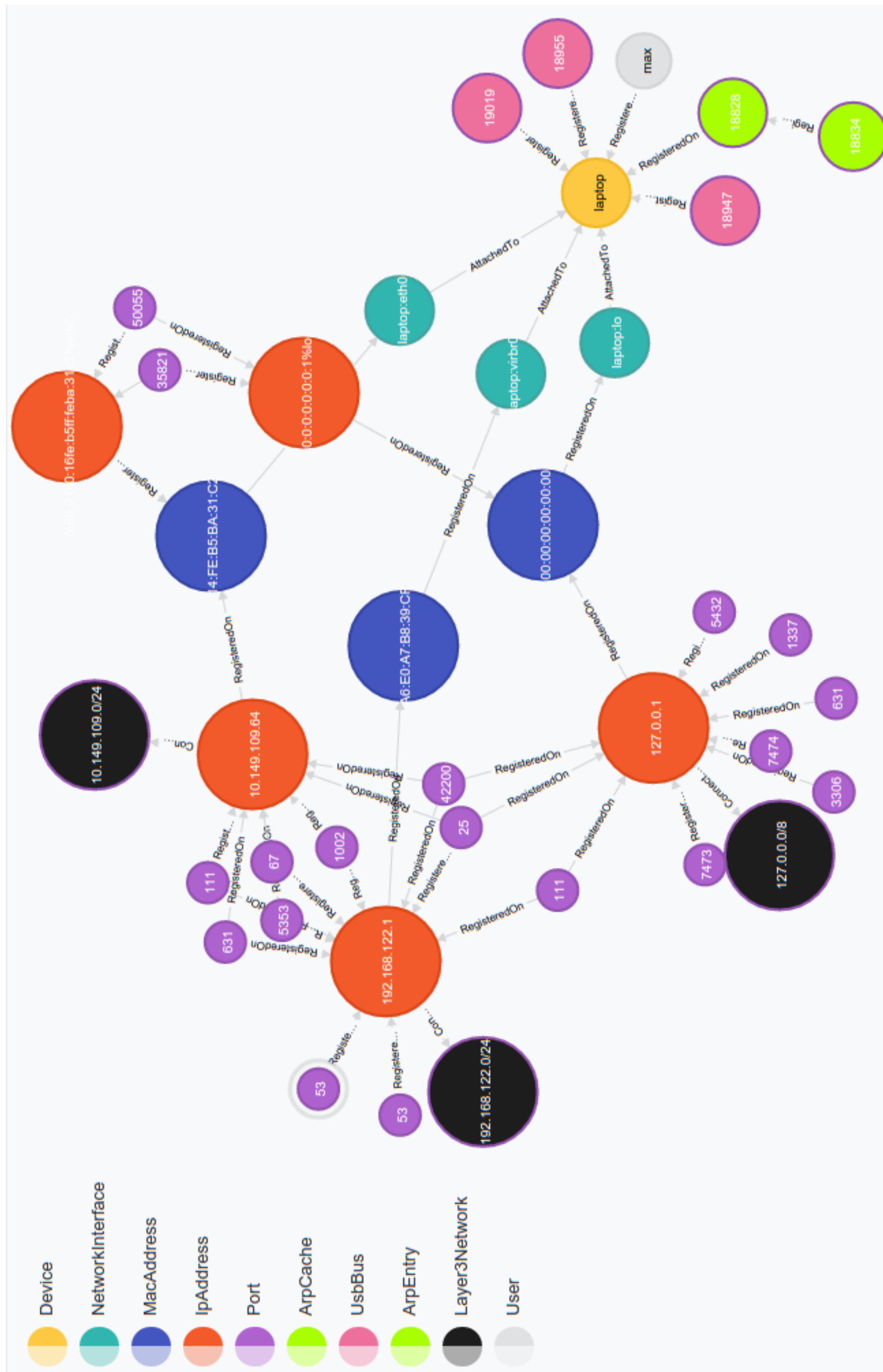


Figure 6.3: Example of the data a local devicescanner writes into the database

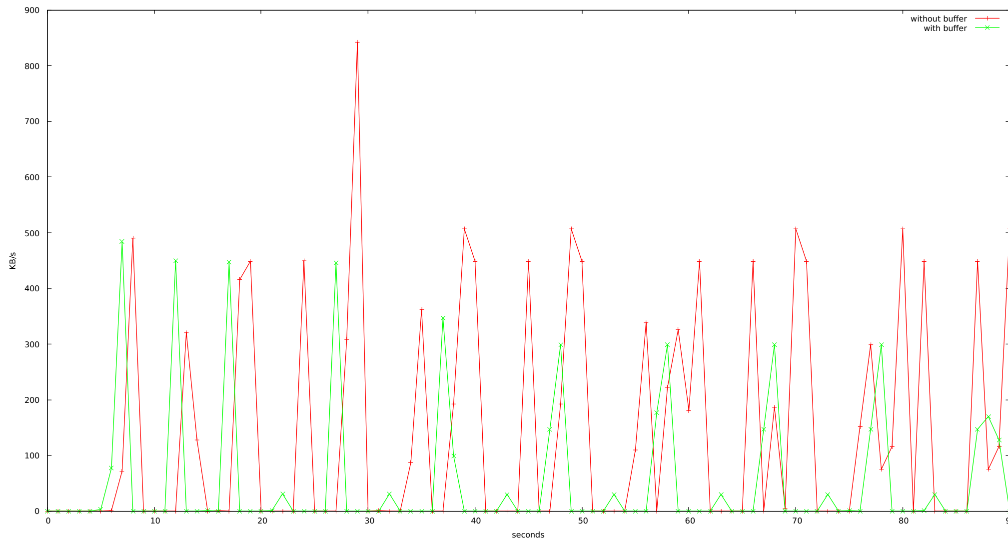


Figure 6.4: Bandwidth usage without using a local buffer versus badwidth usage when using a local buffer

6.2.4 Network traffic

The bandwidth consumption of the devicescanner is important for the same reasons as the CPU and memory usage: the devicescanner should interfere as little as possible with user activities on the device. Table 6.4 shows a comparison of the bandwidth consumption with the usage of a local buffer and without it. The collection of this data is done by using `Collectl`. It collects the amount of kilobytes per second which gets exchanged over the interface which connects the devicescanner with the database. The data is collected in a one second interval. The results are depicted in figure 6.4.

It shows that the usage of the local buffer decreases the frequency as well as the amount of data exchanged with the database.

6.2.5 Logging delay

This section looks at how long it takes for a change of a network property to be detected by the Analyzer. This is evaluated base on the interval in which the data is scanned for. The results are depicted in figure 6.5.

For this evaluation we look at how fast newly opened ports get detected by the Analyzer. For this purpose, we open up a new TCP port every five seconds while the local devicescanner and analyzer are running. The devicescanner was running with seven active modules with five modules running at scanning intervals of 600, 35, 30, 20, 25 respectively. The remaining two modules, responsible for scanning TCP and UDP

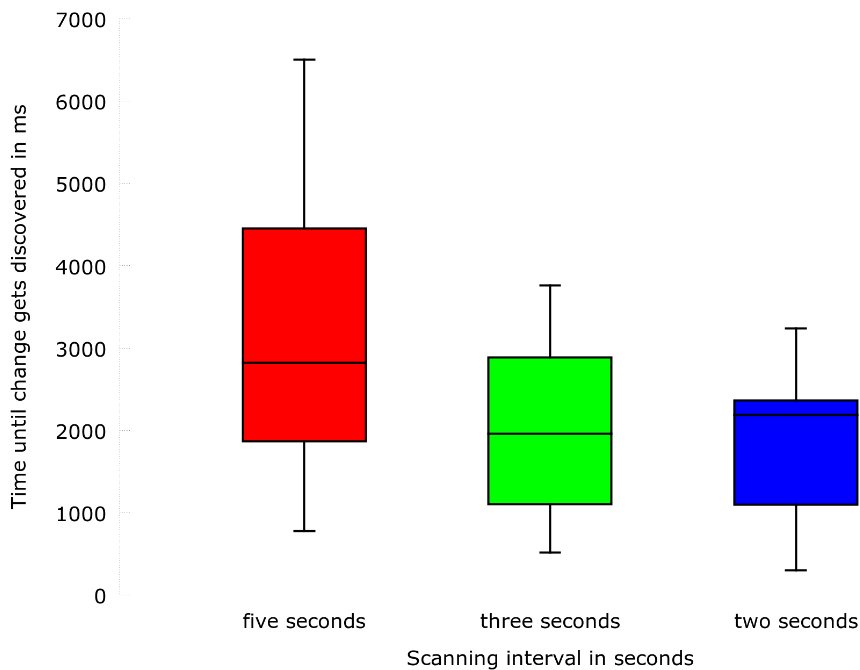


Figure 6.5: Time it took the Analyzer to detect a new port for different scanning intervals

ports, were running at different intervals. For this test, three iterations were done with scanning intervals of five, three and two seconds for the port scanning modules.

The box-and-whiskers plots show the distribution of the measurement results for each scanning interval respectively. 50 percent of the values lie within the box, where the horizontal line inside the box corresponds to the mean of all values. This means that one quarter of the values lie between the upper border of the box (upper quartile) and the end of the whisker (vertical line). And one quarter lies between the lower quartile and the end of the lower whisker. The end of the whiskers correspond to the biggest and smallest values respectively. Additionally, each port got detected in 1.5 times the scanning interval at most.

You can see that for scanning intervals of five and three seconds, 75 percent of the ports get detected in a smaller time than the actual interval. For the two second interval it is approximately 50 percent, which is probably caused by the modules not being able to execute fully in two seconds.

Chapter 7

Conclusion and Outlook

In this thesis the basic design of a system to detect security violations and misconfigurations in a network has been outlined and parts of it have been implemented.

At the beginning of the thesis a list of security violations and misconfigurations which should be detectable by the resulting system has been explained. The security violations and misconfigurations can occur in a network because of an attack or a wrong configuration or usage of important network resources. Furthermore, it has been outlined which requirements the system should meet.

Next, a list of related work consisting of different intrusion detection systems and network monitoring tools has been presented, including an evaluation of the weaknesses and strengths of each of the systems. A result of this evaluation was, that they often don't collect all the available information from the network, because they are deployed with multiple independent instances at different locations, each with a limited view (SNORT). Or they have a global view, but they only correlate states and thresholds of network properties, not the actual data (Nagios).

Next, a design for a system that has a global view of the network and is able to collect detailed data of network resources was introduced and partly implemented by using the `Spring Data` framework, a `Neo4J` database an the `Graphaware Changefeed` module.

Lastly, the developed system has been evaluated in respect to the requirements defined earlier and in respect to its performance. The evaluation showed that the requirements have been mostly met and that the usage of the system does not obstruct modern hardware in a big way.

Next comes a short outlook onto further work that could be done in order to improve the system.

First, data which gets collected by the Local Devicescanner and sent to the database does not get verified in any way. If a malicious program is installed on a client, which knows

how the sending of the data to the database is done, it could alter those packets before they get sent into the network. Alternatively, the source code of the locally installed Devicescanner could be altered in a way that it alters the collected data to its liking. There are two ways of having a chance to detect this: analyzing random traffic samples or verifying the data against other data already in the database.

By monitoring random ARP-request and -reply traffic samples you could, for example detect if a client sends a wrong ARP table to the database. By checking data against other data in the database, before it gets inserted into the database, you could for example detect if a client sends wrong information about its opened ports. If a `Client A` states that he has port 23 closed, but the database suggests that another `Client B` has port 23 open and there is a `Flow` node connecting port 23 of `Client B` to port 23 of `Client A`, `Client A` should not be able to delete his port 23 and the `Flow` node without `Client B` confirming this action.

Secondly, causal analysis could be done by utilizing the fact that each change in the database gets stored by the Changefeed module. Additionally alerts could be based on how long a certain node persists in the database. For example, inconsistencies in the ARP caches across the network could be only considered if they persist longer than the validity time of an ARP entry to mitigate effects described in section 2.1.4.

Thirdly, the collection methods of the device scanners for most data points rely on Linux specific commands or files. This results in the fact that the system can only be reliably used in a network consisting of Linux machines exclusively. Since `Neo4J` and `Java` are platform independent, it could be ported to other operating systems by finding equivalent data collection methods.

Bibliography

[1] E. D. Cardenas, “Mac spoofing : An introduction,” 2003.

[2] “Iana: Service name and transport protocol port number registry,” http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml, accessed: 2015-03-07.

[3] “Telnet security,” http://www.cs.cmu.edu/~droh/755/encr_telnet.html#Security, accessed: 2015-03-01.

[4] “Snmp object navigator: ifoperstatus,” http://tools.cisco.com/Support/SNMP/do/BrowseOID.do?translate=true&objectInput=1.3.6.1.2.1.2.2.1.8, accessed: 2015-03-07.

[5] “Linux manpage of arp(7),” http://linux.die.net/man/7/arp, accessed: 2015-03-07.

[6] “Snmp object navigator: ipnettophysicaltable,” http://tools.cisco.com/Support/SNMP/do/BrowseOID.do?local=en&translate=Translate&objectInput=1.3.6.1.2.1.4.35, accessed: 2015-03-07.

[7] “Rfc 2131: Dynamic host configuration protocol,” http://tools.ietf.org/html/rfc2131, accessed: 2015-03-01.

[8] “Linux manpage of dhcping(8),” http://linux.die.net/man/8/dhcping, accessed: 2015-03-07.

[9] D. G. D. Davis, “Kerberos with clocks adrift: History, protocols, and implementation,” 1996.

[10] “Rfc 1794: Dns support for load balancing,” https://tools.ietf.org/html/rfc1794, accessed: 2015-03-10.

[11] “Deploying load balancing,” https://msdn.microsoft.com/en-us/library/windows/desktop/ee871580(v=vs.85).aspx, accessed: 2015-03-10.

[12] “How to calculate bandwidth utilization using snmp,” http://www.cisco.com/c/en/us/support/docs/ip/simple-network-management-protocol-snmp/8141-calculate-bandwidth-snmp.pdf, accessed: 2015-03-10.

[13] "Was ist split-tunneling?" http://www.lrz.de/fragen/faq/vpn/vpn12/, accessed: 2015-03-01.

[14] T. Shinder, "Remote access vpn and a twist on the dangers of split tunneling," http://www.isaserver.org/articles-tutorials/configuration-security/2004fixipsectunnel.html, accessed: 2015-03-01.

[15] "Rfc 3697: Ipv6 flow lable specification," http://tools.ietf.org/html/rfc3697, accessed: 2015-03-10.

[16] "Linux man page: flow-tools(1)," http://linux.die.net/man/1/flow-tools, accessed: 2015-03-10.

[17] A. M. J.D. Meier, *Improving Web Application Security: Threats and Countermeasures Roadmap.* Microsoft, 2003.

[18] G. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning.* Nmap Project, 2009.

[19] C. Eckert, *IT-Sicherheit: Konzepte - Verfahren - Protoklle.* Oldenbourg, 2001.

[20] "Sslyze (github)," https://github.com/nabla-c0d3/sslyze, accessed: 2015-03-07.

[21] "Turning usb peripherals into badusb," http://srlabs.de/badusb/, accessed: 2015-03-04.

[22] J. L. K. Nohl, S. Krißler, "Badusb - on accessories that turn evil," https://srlabs.de/blog/wp-content/uploads/2014/07/SRLabs-BadUSB-BlackHat-v1.pdf, 2014, accessed: 2015-03-04.

[23] "Linux man page: virsh(1)," http://linux.die.net/man/1/virsh, accessed: 2015-03-08.

[24] "Smart tools," http://www.smartmontools.org/, accessed: 2015-03-08.

[25] "Linux man page: smartctl(8)," http://linux.die.net/man/8/smartctl, accessed: 2015-03-08.

[26] C. Modi, D. Patel, H. Patel, B. Borisaniya, A. Patel, and M. Rajarajan, "A survey of intrusion detection techniques in cloud," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 42–57, 2013. [Online]. Available: http://openaccess.city.ac.uk/1737/

[27] G. Schäfer, *Netzsicherheit - Algorithmische Grundlagen und Protokolle.* dpunkt.verlag, 2003.

[28] "Sectools.org: Top 125 network security tools," http://sectools.org/, accessed: 2015-02-13.

[29] S. Schupp, "Limitations of network intrusion detection," 2000.

[30] B. H. J. V. k. D. I. Bente, J. v. Helden, "Esukom: Smartphone security for enter- prise networks."

[31] "If-map protocol," http://if-map.org/, accessed: 2015-02-02.

[32] A. L. S. T. Z. A. Sadighian, J. M. Fernandez, "Ontids: A flexible context-aware and ontology-based alert correlation framework."

[33] "Nagios host checks," http://nagios.sourceforge.net/docs/3_0/hostchecks.html, accessed: 2015-03-09.

[34] S. Mongkolluksamee, "Strengths and limitations of nagios," http://meetings.apnic.net/__data/assets/pdf_file/0004/18913/Operations_03_Srenght-and-Limitations-of-NAGIOS_Sophon-Mongkolluksameet.pdf, accessed: 2015-03-09.

[35] "Munin project page," hhttp://munin-monitoring.org/, accessed: 2015-03-08.

[36] "Snmp oid tree," http://tools.cisco.com/Support/SNMP/do/BrowseOID.do?local=en&substep=2&translate=Translate&tree=NO, accessed: 2015-01-21.

[37] I. Robinson, J. Webber, and E. Eifrem, *Graph Databases.* O'Reilly Media, Inc., 2013.

[38] "Accessing data with neo4j," https://spring.io/guides/gs/accessing-data-neo4j/, accessed: 2015-02-15.

[39] "Graphaware changefeed module," https://github.com/graphaware/neo4j-changefeed, accessed: 2015-02-19.