university of groningen / faculty of science and engineering / TNO

Automated ICT Infrastructure Modeling as a first step of Automated Cyber Security Analysis

MSc. Thesis

Student: Jorrit Oosterhof
Student number: S2528312
Date: September 6, 2019
First supervisor: Prof. Dr. A. Lazovik (Distributed Systems/JBI)
Second supervisor: Dr. G. Azzopardi (Information Systems/JBI)
Third supervisor: Frank Fransen (TNO)
Fourth supervisor: Jan Pieter Wijbenga (TNO)

Contents

Abstract

Today's society and businesses depend more and more on IT systems and networks, in particular the internet. Companies increasingly run their communication and critical business processes on systems connected to the internet or in the cloud. Using the internet for communication and business processes has a lot of advantages, however, the internet also poses threats to a business. Examples of threats would be: Nation-states trying to disrupt critical infrastructures or stealing information, data leaks due to configuration shortcomings and a lack of security measures, attacks using email messages (phishing), DDoS attacks, etcetera.

In contrast to the first and second recommendations of the Center of Internet Security, many organisations and companies do not have a clear view of what is in their network. The reason is that they tend to do that manually, which is a hard task. The manual task is complicated, because networks tend to change, have many devices, system software gets updated, etcetera. Therefore, a manual model quickly becomes outdated.

To keep a model of an infrastructure up to date, an automatic method of determining which devices are on the network is needed. Detecting changes in the network becomes much easier as a manual analysis is no longer required. This up to date inventory can be used to create a model of the network, which can also help in preventing attacks.

In this research, we propose a method of automatically generating a model of an IT infrastructure, to be used for cyber security analysis, from multiple sources. It covers the process of selecting sources, such as network scanners, collecting and storing data from them and merging the data into a final model. We created a proof of concept which implements this process and it can successfully generate a model of a test infrastructure. The proof of concept shows that it is possible to generate a model of an IT infrastructure, as well as adding new sources to improve the resulting model.

1 Introduction

Today's society and businesses depend more and more on IT systems and networks, in particular the internet. Companies increasingly run their communication and critical business processes on systems connected to the internet or in the cloud. Using the internet for communication and business processes has a lot of advantages, however, the internet also poses threats to a business. For example, the Dutch NCSC[1] identifies several threats: Nation-states trying to disrupt critical infrastructures or stealing information, data leaks due to configuration shortcomings and a lack of security measures, attacks using email messages (phishing), etcetera [36].

Well known examples of nation-states disrupting critical infrastructures are the NotPetya attack in 2017 and Stuxnet in 2010. The NotPetya malware primarily dealt lots of damage to companies and governments in Ukraine, but also other countries as collateral damage. It is believed to be targeted at Ukraine and carried out by Russia [48][54]. Stuxnet was a worm that targeted the nuclear program of Iran. It is believed to be created by the United States and Israel [4].

Other risks include DDoS attacks and data leaks. A DDoS attack is a Distributed Denial of Service attack, which uses lots of internet traffic, for example using a botnet, targeted at a server to make it unavailable due to the enormous amount of traffic. For example, in early 2018, multiple banks suffered from DDoS attacks [38][39][42]. In this case, the Dutch newspaper 'De Volkskrant' found out that the DDoS attacks were performed, because 'it was funny' and that the attacker, among others, also targeted the Dutch Tax Authority [16][40]. DDOS attacks are a large problem, because it disrupts services of the targeted company, such as the banks mentioned. However, DDOS attacks are also used if the attackers do not agree with (some actions of) the company or website they target. For those who do not have the means to perform a DDOS attack themselves, there are services that offer to perform DDOS attacks [55]. Such a service was also used by the attacker of the Dutch banks [16].

Data leaks can happen for various reasons. For example, Mariott International discovered a data leak in 2018 and found out that the attackers had access to their systems since 2014 [32]. Sometimes, a data leak happens due to the information being publicly available by mistake, as is the case with the data leak of a WiFi hotspot finder app [53].

Given the presence of lots of threats against an organisation's infrastructure, they need to protect their digital assets (information, systems, network, etc.). The Center for Internet Security defines 20 controls to protect an organisation and data from known cyber attack vectors [11]. The first control is *"Inventory and Control of Hardware Assets"*. The reasoning behind this control is that new devices, which might not yet be properly configured, or personal devices from employees, which are not always exclusively on the company network, are good targets for an attacker to try and compromise a company's network [12]. The reasoning behind the second control, *"Inventory and Control of Software Assets"*, is as follows. If software that can be installed and executed on devices in the network is not managed by the organisation, attackers can use software that is not up to date or not even needed for business purposes to compromise the host device and from there penetrate the network [13]. Especially for large organisations, having a good overview of hardware and software assets is challenging due to a large number of devices.

In contrast to the first and second recommendations of the Center of Internet Security, many organisations and companies do not have a clear view of what is in their network [23][50][9]. However, cloud services tend to have an excellent overview of (virtual) devices

[1]Nationaal Cyber Security Centrum (National Coordinator for Security and Counterterrorism)

and software in their network, as it is their core business to sell server space. The reason that regular organisations do not have a clear view is that they tend to do that manually, which is a hard task. The manual task is complicated, because networks tend to change, have many devices, system software gets updated, etcetera. Therefore, a manual model quickly becomes outdated. When there is no inventory when an attack is happening, it is not the right time to do an inventory analysis to find the reason. Which device is compromised? What software does it run? Who uses it? If the network contains a large number of devices, these questions are hard to answer without having a proper inventory.

As security of systems and applications improves, attacks will become more and more advanced. To protect themselves from threats, organisations have implemented security measures and monitoring solutions. However, attackers always improve their attacks in order to defeat the security of their targets. Nowadays, incident response is still based on humans, who have to deal with lots of information about what is happening in an organisation. This human factor needs time to comprehend what is going on and take measures to prevent any damage. However, attacks like NotPetya and Stuxnet are automated. Humans can not keep up with automated attacks. To be able to counteract such attacks, it has been suggested that the security of an infrastructure should also be automated [21].

An automatic method of determining which devices are on the network is not only helpful in establishing an inventory of what is on the network, but it can also be used to automatically keep the inventory of the network up to date. Detecting changes in the network becomes much easier as a manual analysis is no longer required. This up to date inventory can be used to create a model of the network, which can help in preventing attacks. For example, tools like SecuriCAD can be used to automatically assess the security of the network based on a model [18]. It can be used to automatically check if the network violates any network policies or it could be used to predict attack paths throughout the network. This enables the security analyst to identify and fix weak points in the network.

This research was done at TNO and the research questions of this thesis were part of the assignment and further specified during the research. At TNO, automated security analysis is an important topic that is actively being worked on. Currently, TNO has developed a prototype for calculating attack steps through a network, but the mock model of the demo infrastructure currently available has to be manually made. The desire is to automate the creation of the model in the prototype. A screenshot of this prototype is shown in figure 1.

Research questions

In this thesis we present a proof of concept of how a network can be automatically modelled for cyber security analysis. The main point of interest of this thesis is how to create a model based on multiple sources, to be used for cyber security analysis.

- How can a model of an IT infrastructure be automatically created?
- Are there sources available to collect enough data about an IT infrastructure?
- Is this data accurate and reliable?
- How can the information be retrieved and combined?

Another real question is whether the method to build such a model is usable or sufficient for automated security analysis of an IT infrastructure? We would like to know whether it is feasible to automatically gather information about an IT infrastructure and if it is possible to collect enough information to perform automated security analysis.

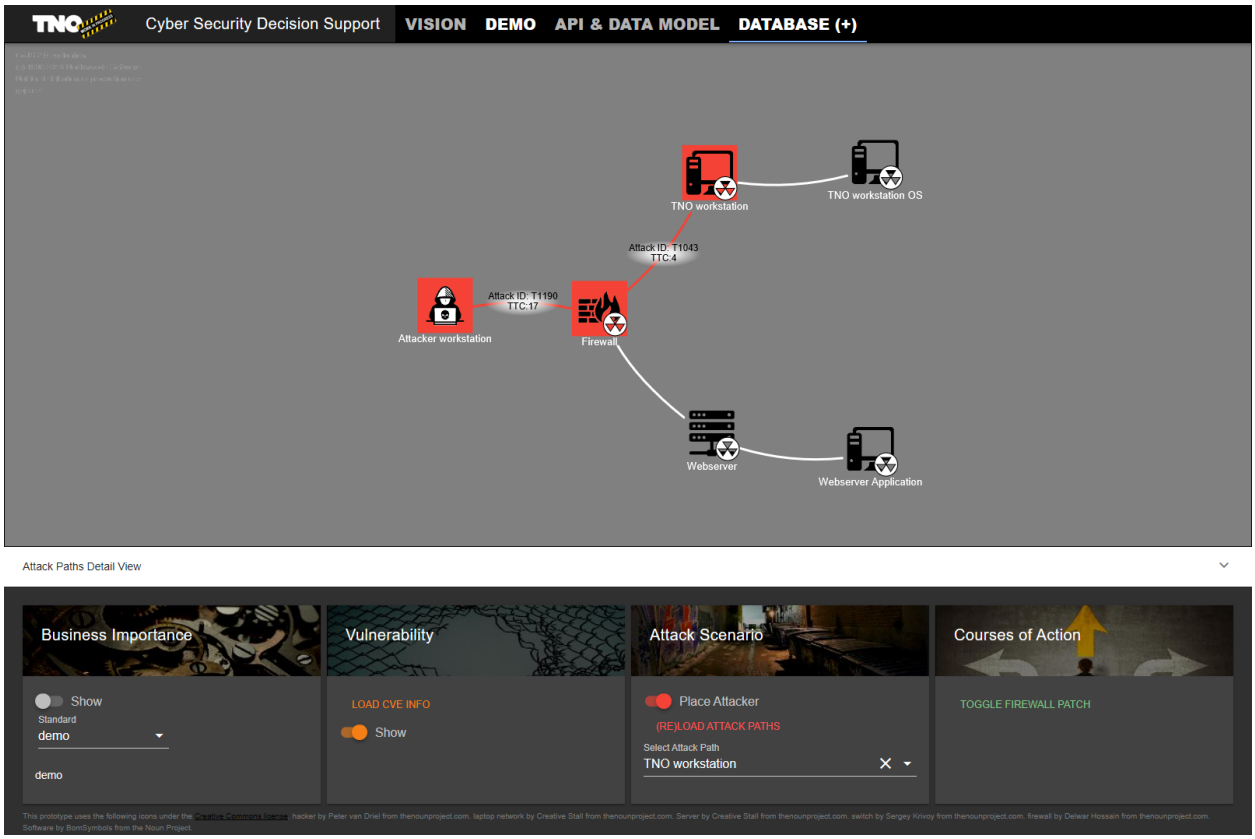


Figure 1: The TNO Cyber Security Decision Support tool

The main point of interest mentions that the model should be based on multiple sources. This means that the method should be flexible in adding a new source. If we need information from a new source, it should be possible to add a new source without much hassle. Is it feasible to collect or merge the information in such a way that it does not matter where the information comes from?

Because the intended goal of the method is cyber security analysis, it is relevant to be able to go back in time and see what changed. Can we recreate a situation in the past based on data from back then?

Finally, the method should be transparent. The mapping from multiple data sources to a final model should not be 'magic'. Instead of a black box that does the mapping, the process should be transparent. Is there a way for the method to be transparent, such that a non-technical person could maintain the rules according to which the mapping takes place?

Intended uses of the model

The intended usage of the model in this thesis is automated cyber security analysis. Within cyber security analysis, the model can be used to detect policy violations, predict weak points in the network by simulating attacks, detect new/unknown devices, etcetera. However, the model may also be useful outside the scope of cyber security, for activities such as inventory management or improving the network topology.

A goal for TNO is to automate security of enterprise IT infrastructures. One way to do this is to use a model of the infrastructure to calculate attack defence graphs. An attack defence graph is a graph that shows the paths that end in a state where an attacker has compromised the system or otherwise accomplished their goal [31]. In addition to only show-

ing the paths, the attack defence graph also contains possible defences and uses information about these defences to show probabilities of (sub)attacks succeeding. Attack defence graphs can be used to find weaknesses in an IT infrastructure. Security analysts can use these attack defence graphs to address these weaknesses.

In 2005, IBM introduced a control loop model as part of their autonomic computing concept [30]. This model, called the MAPE-K reference model, consists of 4 functions:

- **Monitor** Collect details from managed resources and correlate them into symptoms that can be analysed.
- **Analyse** Perform data analysis and reasoning on the acquired symptoms to determine if changes are necessary.
- **Plan** Create or select a procedure to apply the desired changes.
- **Execute** Schedule and perform the necessary changes.

This model was created with fully autonomous systems in mind. The authors of [21] recognised that organisations will be hesitant in having a fully autonomous system to reconfigure their IT infrastructure in response to threats or other security incidents. Because the control loop can still be a reference for understanding what needs to be automated in security operations, the authors adapted the control loop to include the roles of security analysts and decision makers (figure 2).

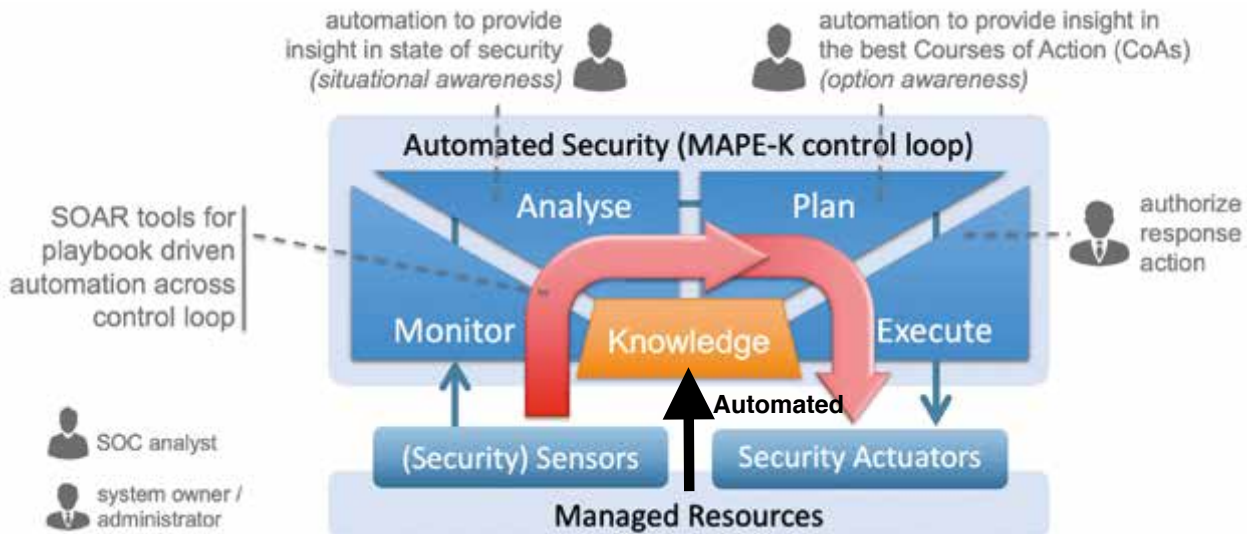


Figure 2: Conceptual figure of MAPE-K model with a 'human in the loop' [21]

To be able to create such an automated security system, the system needs access to information about the network in the monitoring stage and build a model for the analysis stage. Therefore, the model needs to have information about all devices on the network, as well as the software running on those devices, user information, etcetera. Moreover, the automated model generation could also identify new systems or configuration changes in the infrastructure. This may also trigger automated security analysis in order to determine whether the changes introduces new vulnerabilities.

Thesis outline

Section 2 describes related work regarding automatically modelling of IT networks and infrastructures, section 3 describes our definition of a model of an IT infrastructure and relevant

concepts, section 4 describes the methods that were used to arrive at a way to automatically generate a model of an IT infrastructure and how it was validated, section 5 describes how the data collection and mapping process was implemented using a prototype, section 6 describes the results of the presented method, section 7 concludes the thesis and section 8 gives an overview of future work that could improve the presented method.

2 Related work

In the literature, we could not find much research about the subject of automatically generating a model of an ICT infrastructure. However, there have been efforts to establish such a method or to identify the requirements for automatically generating a model.

The authors of [10] recognise that automatically collecting the required data and building a model would decrease the effort of modelling and improve the data quality. The authors propose a tool that can automatically instantiate a part of the CySeMoL [51] modelling language elements based on results from the NeXpose[2] vulnerability scanner.

The authors of [29] did a similar study and mapped the meta-model of Archimate[3] to the output of network scanners, in this case NeXpose. A limitation of this method, as well as the previous one is that only one data source is used to generate the model.

Similarly, the authors of [57] recognise the need to automatically generate a model of an IT infrastructure for security purposes. Instead of collecting data from one source, they focus on collecting data from multiple sources. However, their study only uses data from two sources and does not seem easily extendible.

In [20], the authors investigate requirements for automated enterprise architecture model maintenance. They review literature on the subject and conduct a survey to find out whether enterprise architecture automation is demanded in practice. Based on their literature review and survey, they propose a list of requirements for a tool that facilitates automated enterprise architecture maintenance.

The authors of [47] recognise the need to automatically create a model of an IT infrastructure as well as the need/usefulness of using multiple sources/tools to arrive at a model. They present an extensive list of tools that can be used for this purpose. However, these tools need manual modelling of the infrastructure at hand.

There are researchers who have developed a method to automatically create enterprise topology graphs. In [6], the authors present a method to automatically create an enterprise topology graph of an enterprise IT infrastructure. The authors define an enterprise topology graph as 'a graph capturing a fine-grained technical snapshot of the complete enterprise IT, i. e., including all processes, services, software, infrastructure, and Cloud offerings as nodes, as well as their relations and dependencies as edges'.

In [3], the authors present an algorithm for automated enterprise network topology discovery. This algorithm is also capable of dealing with incomplete data and uncooperative devices. In the real-world scenario where this algorithm was tested, it performed well.

The company Edgewise Networks offers a Data Flow Mapping in their Edgewise zero trust platform, which shows data flows through the network [17]. They show the path (of data) through the network, which hosts are talking to each other, which applications do the talking, etcetera.

There is also a method of enhancing security testing via automated replication of IT-asset topologies, as presented in [7]. The authors recognise that creating and maintaining a test environment for security testing can be a cost-intensive task. They propose a method that can automatically replicate a production environment in a virtual environment, where security tests can be performed without disrupting business processes.

In contrast to some of the research mentioned here, we setup a method to automatically generate a model of an IT infrastructure from multiple sources. Our method stores data from these different sources in a database where it is snapshotted, such that one can 'go

[2]Available at `https://www.rapid7.com/products/nexpose/`

[3]Available at `https://www.archimatetool.com`

back' in time. The data is then merged and a model is generated from the merged data. We defined the model itself by creating a data model using Object Role Modelling, which is semantically stable.

The solutions presented in the works above are good solutions on their own, however, they do not provide what we are looking for in this thesis. We like to have an automatically generated model of an IT infrastructure that combines information from different sources, such that it can be used for automated security analysis.

Because we want more information than these solutions provide, these solutions could be viewed as just another source. The proof of concept presented in this thesis combines information from multiple sources and allows for more sources to be added. Therefore, these solutions can be used to improve the automatically generated model.

This section describes related work that shows research that has been conducted by others to solve the problem. The next section describes relevant techniques and solutions that are available and can be used to solve the problem.

3 Relevant concepts

This section provides an overview of the techniques that can be used for collecting data, storing the data and merging the data. Section 3.1 describes the definition of a model of an IT infrastructure and shows the structure of the remainder of this section. Section 3.3 describes the different possibilities of collecting data about an IT infrastructure, section 3.4 describes database types and their characteristics, which can be used for storing the data, section 3.5 describes strategies for merging data from different sources and section 3.6 describes what kind of network architectures can be expected in corporate networks.

This section provides background information about the techniques and solutions that are already available to solve this problem. If this information is already known to the reader, this section may be skipped.

3.1 IT infrastructure model

The authors of [6] define an enterprise topology graph as 'a graph capturing a fine-grained technical snapshot of the complete enterprise IT, i. e., including all processes, services, software, infrastructure, and Cloud offerings as nodes, as well as their relations and dependencies as edges'.

In this thesis, the goal of the model is to capture a complete view of the enterprise IT infrastructure at hand. Challenges are to determine which information is needed to be able to create a model as well as which information is needed for the intended usage of the model.

The process of generating a model, as described below, is derived from the following considerations and inspired by the Extract, Transform and Load process [34]. In this thesis, we would like to generate a model from multiple sources. Because it is impossible to use every possible source that may exist, a logical consequence is that a selection has to be made of which sources to use.

Naturally, data has to be collected from the sources before it can be used. The requirement of being able to go back to a previous situation leads to the desire to save the collected data in a database, such that there are snapshots of the data, which can be used to recreate a past situation, possibly with new insights.

We would like to have one model of an IT infrastructure, because that allows us to reason about the state of the infrastructure. We need to know what the target model looks like, such that the process can be implemented to generate this target model. This means that we need to know what should be in the model. The level of detail is dependent on the purpose of the model. For example, a model for security purposes should contain information about installed software patches, security related settings, etcetera. A model that is only used to view a graph of communicating hosts does not need such a level of detail.

The proposed process of generating a model of an IT infrastructure consists of a few steps. Once we know what data we need, we need to select sources to retrieve this data from, collect the data and then use it to generate the model. Figure 3 shows a schematic overview of what the process of automatically generating a model in this thesis looks like.

1. **Selecting sources**
   To generate a model of an IT infrastructure, you need information about the infrastructure. To know which information you need, it is helpful to have a data model that tells you wat you need. Section 3.2 discusses a data modelling technique to create a semantically stable data model. Then, you can find sources that offer the required data and choose one or more of them.

2. **Collection**
   The next step is to actually collect the data about the infrastructure. This involves setting up the selected sources, collecting the information, adapting it to your needs, etcetera. There are many different ways of collecting data about an infrastructure. It is possible to use active or passive scanners, vulnerability scanners, agents, etcetera. Section 3.3 provides background information about the different ways of collecting data.

3. **Storage**
   The collected data must be stored somewhere such that it can be used. There are multiple types of database systems that can be used for storing the data. Section 3.4 describes different types of database systems and section 4.3 describes the process of collecting and storing the data about the infrastructure and the databases used for storing the collected data.

4. **Generating model**
   To arrive at a model, the collected data must be used and merged to generate the model. There are many strategies when it comes to merging data. Section 3.5 provides an overview of a set of different strategies and section 4.4 describes which strategies are used and how the the model is generated.

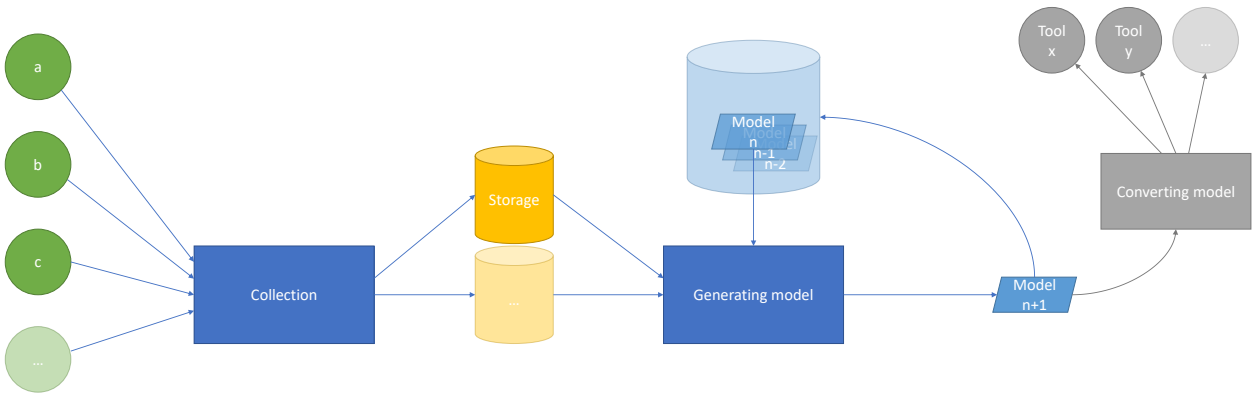


Figure 3: Process of automatically generating a model

3.2 Creating a data model

There are a lot of ways to collect information about a network. For each possible way of collecting information, tools are available to collect that information as explained in section 3.3.

First, you need to know what your final model should look like. One way of clearly deciding on what you need or want to know is to create a data model of how the final IT infrastructure model should look like. This way, you have a formal definition of how the model should look like and which information is needed to generate the final model. Using the data model, you can find tools that can scan the network and provide the required information to generate the final IT infrastructure model. This section describes one way of creating a data model for the final IT infrastructure model.

To create a data model, we followed the Conceptual Schema Design Procedure (CSDP) [25]. CSDP is a procedure for conceptually designing an information system. The resulting

data model is built using Object Role Modelling (ORM) [24] which is closely related to the CSDP. ORM is a way of modelling based on factual statements about the domain that is to be modelled. An advantage of using ORM is that it is semantically stable, which means that extending the data model does not change anything except adding the new facts that come with the new information.

First, examples from the Universe of Discourse are transformed into elementary facts. In this case, the universe of discourse is an IT infrastructure. After having the facts and fact types, constraints are added such that the data model only allows the information we are interested in.

The remainder of this section provides an introduction to the Object Role Modelling language to allow the reader to understand the data model. This section may be skipped if ORM is already known to the reader.

Object Role Modelling

Figure 4 shows the main symbols that are used within the ORM data models in this thesis. In ORM, the designer states facts, such as '`Person` has `Name`'. In this case, Person is an entity type. Entities are described by facts and are depicted as in figure 4a. Name on the other hand is a value type, as seen in figure 4b, which can be a string, a number, a date, etcetera. An entity type can also have a reference scheme, that unique identifies the entity. For a person, this could be a '`personID`' or '`socialSecurityNumber`'. An entity type with reference scheme looks like figure 4c.

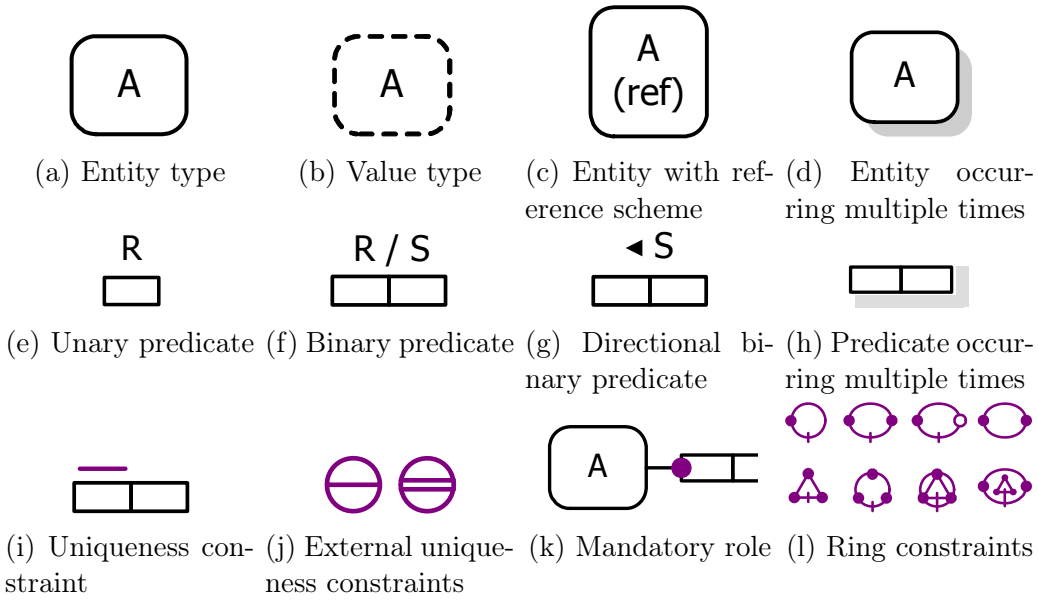
Entity types are connected to each other or to value types using predicate. The simplest predicate is the unary predicate, shown in figure 4e, which is a property that can be true or false. An example fact that illustrates a boolean predicate is '`Invoice` is paid'. In this case, the text 'is paid' comes in place of 'R'. A predicate involving another entity or value is depicted in figure 4f. Given the fact '`Person` drives/is driven by `Car`', in figure 4f 'R' is replaced by 'drives' and 'S' is replaced by 'is driven by'. This is written like this, because it is possible to present a fact in two ways: '`Person` drives `Car`' and '`Car` is driven by `Person`'. If a predicate should only be read from right to left, an arrow is added as seen in figure 4g. Additionally, if a role is mandatory, a purple circle is placed at one of the ends of the connecting line, as seen in figure 4k.

Each predicate must also have a uniqueness constraint. Uniqueness constraints are depicted using a purple bar above a role, as seen in figure 4i. Uniqueness constraints can also span over multiple rows. A uniqueness constraint states that instances of a role or a combination of roles must be unique in the fact type population.

It is also possible to define an external uniqueness constraint, which allows the designer to enforce a combination of roles from multiple predicates to be unique. figure 4j shows how external uniqueness constraints look like. An external uniqueness constraint is connected to the corresponding roles by a dotted line. When the external uniqueness constraint shows two lines, it is a preferred external uniqueness constraint.

Within one ORM diagram, it can become too busy with entity types, value types and roles. To avoid a crowded diagram, it is possible to create multiple separate diagrams. Because entity types can occur in more than one diagram, ORM adds a shadow to those entity and value types, to indicate that they occur in multiple places as seen in figure 4d and in figure 4h for roles.

ORM also offers ring constraints, which indicate that a binary predicate is, from left to right, top to bottom, irreflexive, asymmetric, antisymmetric, reflexive, intransitive, acyclic,



(a) Entity type (b) Value type (c) Entity with reference scheme (d) Entity occurring multiple times

(e) Unary predicate (f) Binary predicate (g) Directional binary predicate (h) Predicate occurring multiple times

(i) Uniqueness constraint (j) External uniqueness constraints (k) Mandatory role (l) Ring constraints

Figure 4: Subset of main ORM graphic symbols [24]

intransitive and acyclic, or intransitive and asymmetric. Some of these constraints can also be combined. They are connected to the corresponding predicate by a dotted line.

3.3 Collecting data

There are several options for collecting data about devices in the network. Two network scanning strategies are active and passive scanners. In case of managed devices, a feasible option is having an agent on each device that reports information about the device. Furthermore, a management system, which is used to keep an inventory of the infrastructure and monitor services in the infrastructure can also be a good source. Sections 3.3.1 to 3.3.4 describe the following topics. Additionally, section 3.3.5 describes obstacles that can be encountered when collecting data about an infrastructure.

- Active scanners
- Passive scanners
- Agents
- Management systems

3.3.1 Active scanners

Active scanners initiate communication, e.g. by sending a request or a special packet, with a device and use the response to deduce particular information about the device. They use the response to determine whether a device exists and, depending on the capabilities of the scanner, what operating system and software it runs. Determining the OS and software can also be done later, using a different kind of request or packet. Usually, the input is a list of all IP addresses or subnets to scan. An example could be merely pinging the IP address. Figure 5 shows a schematic of how an active scanner works.

As active scanners send network traffic to devices in the network, each device should be able to handle this traffic. However, this is not necessarily the case. All systems can crash due to malformed of unexpected packets, but Industrial Control Systems typically expect

network traffic in a specific way and are more likely to crash due to active network scanners [60].

There are several different things that active scanners can do. For example, active scanning can be used to do port scans, vulnerability scans, device fingerprinting, etc.

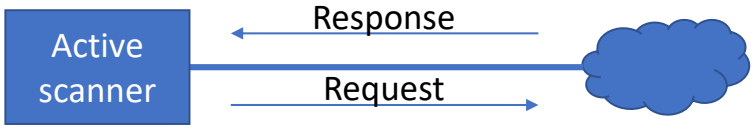


Figure 5: Schematic overview of an active scanner

Port Scanning Port scanning is the process of checking which ports are open on a host. There are multiple ways to check whether a port is open. One way of checking whether a port is open is to connect to the port, i.e. open a connection and after it is established, close it immediately [44].

Another way, also called a half-open scan, is to start opening a connection and as soon as the target responds with an acknowledgement packet, the scanner sends an RST packet (reset), effectively closing the connection before it was fully set up.

A third way is a stealth scan. With this type scan, the packets sent to ports are configured in such a way that they go unnoticed, the receiving host will not log them. The scanner can send a FIN packet (finish, used for terminating a connection) and uses the response to determine whether a port is open or not [41]. However, intrusion detection systems can be configured to detect these types of scans nowadays.

Device Fingerprinting Device fingerprinting is the process of determining the software that runs on a discovered host. If this is done actively, the scanner sends packets to the target host and uses the response to try and find out the operating system or software that is running on the target. Most operating systems respond differently to certain requests. These specific characteristics can ben used to guess which operating system is running on a host. This is called TCP fingerprinting, which is also used by Nmap[4].

Vulnerability scanning A vulnerability scanner, such as Nessus[5], scans the network for vulnerabilities. It scans for devices on the network and determines the operating system and software that is running on a host. Using signatures of vulnerabilities, it can verify whether a host has vulnerabilities [28].

There are several approaches to finding vulnerabilities, both active and passive. Active scanning involves polling devices in the network and determining what software they run, etcetera. There are many approaches, for example, vulnerabilities can be found by trying to see if they exist without disrupting the device, but also by exploiting the vulnerability.

Furthermore, there are authenticated and unauthenticated scans. During an authenticated scan, the vulnerability scanner is given the credentials of the hosts in the network, giving the scanner the opportunity to login to the hosts and then determine the running software and vulnerabilities.

[4]Available at `https://nmap.org` (Open Source)

[5]Available at `https://www.tenable.com/products/nessus/nessus-professional` (Commercial)

3.3.2 Passive scanners

Passive scanners, such as Senrio[6] and Ettercap[7], are limited to sniffing network traffic. As a result, a passive scanner only detects a host when it sends or receives network traffic. If a host does not generate traffic while the passive scanner is sniffing, it will not be detected. Figure 6 shows how a passive scanner can intercept traffic.

Another limitation is that a passive scanner can only see the data that travels along the connection that the scanner is sniffing. To detect devices throughout the entire network, one must place passive scanners at key points in the network, such that all network traffic is likely to be picked up by on of the passive scanners.

A problem for passive scanners is a switched network[8]. As a switch directs traffic only to the ports that the traffic is intended for, host-to-host traffic will not travel past a passive scanner unless the scanner is in between the two hosts. This problem could be solved by using managed switches, which can send all traffic to a monitoring port [62].

As passive scanners do not generate network traffic themselves (for detection purposes), they can be safely used in a network that contains industrial control systems.

Passive scanners can be used for different purposes, such as intrusion detection, vulnerability scanning, device fingerprinting, etc.

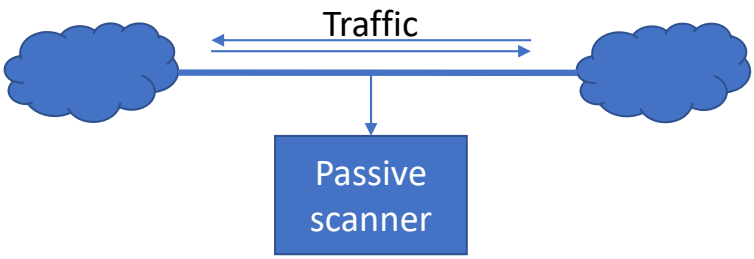


Figure 6: Schematic overview of a passive scanner

Device fingerprinting If device fingerprinting is done passively, it can only use the traffic to try and deduce what operating system a device runs. A downside to passively fingerprinting devices is that it is always limited by the available traffic on the network, which makes the process slower and potentially less accurate than active OS fingerprinting.

Part of device fingerprinting is determining the operating system that runs on the host. This can be done in different ways. According to [33], simulating network congestion can give information about the running operating system due to differences in packet retransmission. Analysing DHCP traffic from a host can also indicate which operating system is running on the host.

Intrusion Detection Systems Intrusion Detection Systems (IDSs), such as Snort[9], try to detect intrusions by monitoring network traffic and identifying certain patterns that could indicate an attack, unauthorised access to (devices in) the network, etcetera. Such an attack can, depending on the circumstances, be detected as it is happening or after the attack has finished.

[6] Available at `https://senr.io/products.html#discovery` (Commercial)
[7] Available at `http://www.ettercap-project.org` (Open Source)
[8] For a detailed description of the inner workings of switches, refer to [52].
[9] Available at `https://www.snort.org` (Open Source)

There are two types of intrusion detection: signature based detection and anomaly detection [5]. Signature based detection uses previous knowledge about attacks to detect them. However, as it uses signatures of existing, known attacks, it cannot detect new attacks.

Anomaly detection is able to detect new attacks, as it observes the network traffic, establishing a baseline of what is normal traffic. Afterwards, it detects anomalies in network traffic, e.g. a device communicating with another device, which it has never done before, which could be (part of) an attack.

The relevance of an intrusion detection system lies in the fact that it is already scanning all network traffic as it is a passive scanner. If an intrusion detection system is already available, using a separate sniffer is not needed, as the IDS can provide the same information as a sniffer, apart from the extra information about intrusions that the IDS provides.

Vulnerability scanning There are several approaches to finding vulnerabilities, both active and passive. A passive vulnerability scanner must be able to extract vulnerability information from the network traffic it sniffs. As it can only get the information from the traffic it intercepts, this is a harder task than actively scanning for vulnerabilities. However, if active scanning is not possible for some reason, passively scanning for vulnerabilities can be useful.

3.3.3 Agents

For devices that a company has under control, such as desktop workstations in an office, servers, etcetera, the company can install an agent on those systems that report the relevant information about the system. This can be a custom agent or an agent that is part of a system that collects information about the infrastructure. The agent can also be part of the SNMP protocol, which is described at the end of this section.

Running an agent on devices in the network is an excellent way to collect information about the device, as the agent has access to all information about the system and as such is the most reliable source. However, the need for other sources, such as network scanners, is still present as it may be impossible to install the agent on all devices. Think of phones and laptops of employees (BYOD), systems that do not allow extra programs installed or Internet of Things devices.

Although an agent is reliable because it has access to all information about the device it is running on, it may miss something. Think of an application that was not installed using the package manager, but compiled and executed in the background. If the agent only pulls a list of software from the package manager, the locally compiled application is not detected.

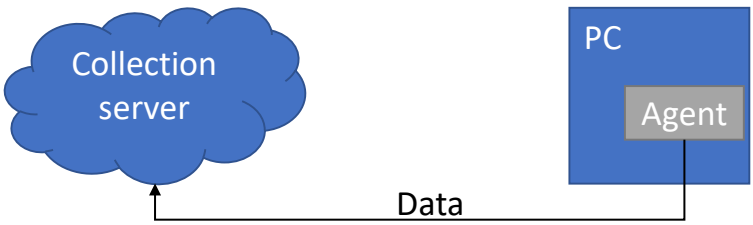


Figure 7: Schematic overview of how an agent sends data to a central collection server

There also exists another option, which is executing a script on a host that needs to be identified and have the result uploaded to the system managing the collected information. This process is schematically seen in figure 7. The advantage of this approach is that it is not

needed for the collection server to be able to see all devices it needs to collect information from. If the devices report to the collection server themselves, only those devices need to know about the location of the collection server.

Simple Network Management Protocol SNMP is a protocol that can be used for monitoring managed devices. In a network where SNMP is used, there are one or more managers, which are responsible for querying devices and responding to events from those devices. Each managed device runs an agent that can be asked to return information about the device. In case of errors, the agent can (asynchronously) send a message to the manager [19].

This protocol is widely supported by many devices and manufacturers, ranging from workstations to routers and switches and IP phones. Using this protocol could be beneficial compared to writing a custom agent. As an example, it is used by Observium[10], which is a tool for monitoring devices.

A similar principle (excluding changing settings on the agents) is used in an SNMP alternative called System Center Operations Manager[11], developed by Microsoft.

3.3.4 Management systems

Management systems can be a good source for information, as seen schematically in figure 8. For example, a type of management system that is likely to be present within an organisation is a monitoring tool. Monitoring tools, such as Nagios[12], are used to, as the name suggests, monitor systems. They can be used to send a message to the system administrator when a server goes down.

Management systems usually contain a list of at least the most important devices in the infrastructure. This list can be used for an active scanner to detect the running operating system and running services.

When the management system also monitors running services, it can also provide information about what services are running, which makes it a good source for building a model of the infrastructure.

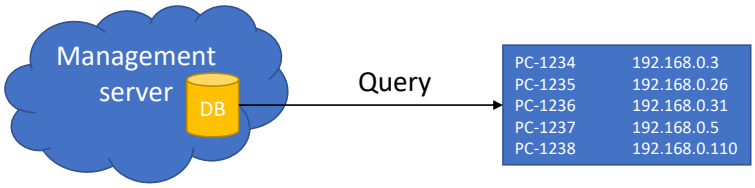


Figure 8: Schematic overview of how a management system could be used

In addition to management systems, basic network protocols such as DHCP and DNS can also be used to locate devices in the infrastructure. A DHCP server hands out IP addresses and maintains a list of used addresses, which can be used by an active scanner. A DNS server has a list of names that resolve to IP addresses, which can also be used by an active scanner, given that the IP addresses are part of the infrastructure.

[10] Available at `https://www.observium.org` (Open Source)
[11] `https://www.microsoft.com/en-us/cloud-platform/system-center`
[12] Available at `https://www.nagios.org`

3.3.5 Obstacles

When scanning the network, there are situations and techniques that may make it more difficult to find all devices in the infrastructure or collect information from them. This section lists obstacles that can be encountered when collecting information about an infrastructure.

VLAN Problems arise when Virtual LANs[13] are used, as traffic destined for a device in a certain VLAN will not be sent to a device that is not part of that VLAN. Therefore, when using passive scanners, the scanner will only receive traffic within its own VLAN. Similarly, an active scanner can only probe devices within its own VLAN. This is an issue that should not be overlooked, as it can result in useful data going undetected.

To overcome this obstacle, the active and passive scanners must be part of all VLANs or there must be a separate scanner per VLAN.

NAT Network Address Translation[14] could also cause problems in scanning the network. Even though it should not happen, although running a virtual machine will often also use NAT, if someone were to setup a network behind a NAT, the hosts on that subnetwork will not be discovered by scanning software. In fact, all traffic from those hosts will appear to originate from one IP address.

To discover these hosts, the scanner should be aware of the NAT and treat the corresponding IP address as a subnetwork. To collect information about the hosts behind the NAT, a scanner could be setup behind the NAT, such that the hosts can also be detected or the NAT logs could be used to identify the hidden hosts. In case of virtual machines, there should be a scanner within the internal virtual network to identify all virtual machines.

VPN Virtual Private Networks allow users to connect to the network from anywhere in the world by creating a secure tunnel between the device and the network. However, detecting devices connected via a VPN might not work as well as local devices. For example, without proper configuration, broadcast packets may not be sent over the VPN.

Actively checking the IP range used by the VPN server could be used to detect all devices connected through a VPN as well as using the VPN logs to reliably detect connected hosts.

Devices without agents Some tools make use of agents or remote access to detected hosts. Having extensive access to a device allows for collecting a lot of data which is accurate, due to being collected directly at the source. However, there can be various reasons why running an agent is not possible. For example, people can bring their own devices, which do not run an agent to the network or an Industrial Control System can be part of the network, but does not support running an agent, etcetera.

When such access, using an agent, to a device at hand is not possible, the collected information becomes inaccurate, due to having to guess what operating system and other software is running on the host or because some information can not be collected without extensive access.

Unfortunately, there is not a trivial way to overcome this obstacle, other than installing an agent on devices that do not yet have one. If this is not possible, it could be an option to record the relevant information about these devices in a management system or to use less accurate information from other sources, such as an active scanner.

[13]For a detailed description about VLANs, refer to [52].
[14]For a detailed description about NAT, refer to [52].

Unavailable information Some information may be unavailable. For example, if needed information about a device is not of a technical nature, this information could simply be non-existent. In this case, the information must be provided manually. Examples of this type of information include the function a device offers, its physical location, etcetera.

Uncertainties and errors It may happen that one scanner has a different observation about a device than another scanner. E.g. scanner A reports that a host runs Windows and scanner B reports that that host runs Linux. The obstacle is that one of them is wrong and it is a challenge to find out which one is wrong.

This problem could be solved by giving the sources a different amount of trust. In a simple setup, one could always trust scanner B to provide to correct answer and resort to scanner A if B does not have the requested information. However, this is just one example strategy. See section 3.5 for more information about merging data from conflicting sources.

3.4 Storing data

There are different types of database systems. Amongst others, the main distinction is between relational database management systems (RDBMS) and NoSQL databases. This section describes relational databases and four different types of NoSQL databases.

Relational databases A relational database consists of tables or relations. The definition of a relation defines what the data looks like, e.g. what each column contains, but not the data itself. Each row or tuple is an instance of the relation [26].

Usually, the data in a relational database is normalised, such that there is no data duplication or as little as possible. This can result in many small relations, which depend on each other to retrieve all information about an entity, such as a customer, in the database. Relational databases satisfy the ACID principles [49].

Column-oriented databases Column-oriented databases, such as Cassandra[15], consist of tables where the data is not stored according to row, but according to columns. Traditional row-based databases are fast in writing as they can write an entire row away quickly. However, in reading data, column-based databases have an advantage, as columns that are not needed for a query do not need to be accessed and therefore, the data will not be read.

Another advantage that column-oriented databases have is the efficiency of aggregation. For example, calculating the sum of a column in a row-store is expensive, as each row has to be accessed individually and the corresponding value has to be extracted. In a column-store however, only the column is needed and the 'sum' can walk over the values and sum them.

Column-oriented databases also have an advantage when compressing data. Compression of row-based storage is relatively difficult as the data in a row is not necessarily similar, whilst in column-based storage, the data within a column usually is similar and can be compressed more easily [27].

Document-based databases This type of database stores data in collections of documents. For example, for an application with users, there would be a collection called users and each user would be stored as a document. Such a document could be a JSON object or an XML object.

[15] Available at `https://cassandra.apache.org` (Open Source)

In document based databases, such as MongoDB[16], a collection usually contains documents that describe a similar entity, such as the users collection mention above. However, it is not required to have the same sort of documents within a collection.

Typically, although not required, the documents are similar, but not every document must have the same properties. It is up to the programmer to decide how and where to store objects [1][35].

A document can, next to properties, also contain arrays or nested documents. An example would be movie documents containing an array of actors. This may result in actors being defined in multiple places (movies), but in contrast to relational databases, document-based do not support join operations. Therefore, it is necessary if one wants to avoid implementing join-like operations within the application.

However, there is nothing that prevents the programmer to use ID's to identify documents, have a separate actor collection and give each movie an array of actor ID's.

Key-Value databases These databases store data in a hashmap type structure where data is identified by a key, which points to the value. The value can be anything and holds the data. This type of database is essentially just a two-column table with a 'key' column and a 'value' column. Usually, the key column is integer or string/binary and the value column is string/binary [37].

Key-Value databases, such as Redis[17] and memcached[18], can be in-memory or persistent on disk. If the database is in-memory, it is usually used for caching. If the database is stored on disk, the data is persistent.

These databases are convenient when an application has entities that are uniquely identifiable by some ID, such as sessions or shopping carts [2].

Graph databases This type of database can be used to store relations between entities. An example would be who starred in movies and who directed those movies.

Relational databases can also be used to store such relations and using 'joins', one can query this information. However, regular RDBMS are not suited for graph traversal [56]. SQL does not provide a way of traversing a graph other than performing multiple joins. The deeper the graph traversal, the more joins are needed. However, it is not possible to traverse the entire graph, as it is unknown how deep the traversal goes and therefore, how many joins are needed.

Graph databases solve this problem and provide an intuitive way of storing graph data and querying it. They provide the possibility to add vertices and relations and traverse the graph according to criteria or to just traverse x depths from a vertex to see neighbouring vertices and relations (for example: all movies a person starred in and all the costars and directors, which is 2 levels deep).

This type of database stores data in the form of a graph. The vertices are objects holding data and the edges represent the relations between these objects (vertices).

Multi-storage type databases In addition to the types of databases mentioned above, or rather in extension to them, there are multi-storage type databases or multi-model databases. When an application requires multiple types of databases, e.g. a graph database and a document-based database, the developer needs to maintain two different databases. As the

[16] Available at `https://www.mongodb.com` (Open Source)
[17] Available at `https://redis.io` (Open Source)
[18] Available at `https://www.memcached.org` (Open Source)

different databases have their own requirements, this may be challenging. Furthermore, keeping data consistent across the different database can also be challenging [45].

Databases which support different models are created to solve these issues. Internally, these database systems may use different storage techniques. For example, ArangoDB uses a document store where its data is stored [46].

3.5 Merging data

To merge data from different sources, data fusion can be used [58] as well as a process similar to the steps in [57].

Data fusion is the process of fusing data from multiple sources into one dataset. There are a few possibilities when having data from multiple sources. The data can be complementing, e.g. for an entity x from source A, source B can contain information that wasn't known yet about x. Complementing fusion aims to make the data more complete.

The data can also be competitive. This means that the data is conflicting, e.g. source A and B both say something about x, but only one of them is correct. The challenge is how to choose which source to believe. Competitive fusion aims to improve accuracy of the data.

Finally, the data can also be cooperative, meaning that the data from both sources allows us to infer new data.

3.5.1 Information fusion

The data that is collected about an ICT infrastructure is not raw data, like data from sensors. It is higher level data that is mostly not numerical. Therefore, the data merging process is best described with the concept of information fusion. In information fusion, there are two types of inconsistencies [8]: schematic inconsistencies and data inconsistencies.

Schematic inconsistencies The first type of inconsistencies is about the data structure. It contains issues like a data source knowing attribute a about an entity where another data source does not collect information about a (think of data source 1 knowing the operating system of a device and data source 2 does not, but it does know the location of the device). Another issue is when the same attributes have different names (think of data source 1 having a column 'IP Address' and data source 2 having a column 'ipaddress').

Schematic inconsistencies should already be solved in the data storing process, as the data should be stored in a generic data model. It is not useful to work with the data in its original form. Having a generic data model dictating how to store the data enables easier access to the collected data that is to be merged.

Data inconsistencies The other type of inconsistencies is data inconsistencies. These inconsistencies are about two sources having data about an entity, but they don't have the **same** data. Where schematic inconsistencies involve the data *structure*, data inconsistencies involve the data itself. In solving data inconsistencies, one can encounter three scenarios: both sources say the same (i.e. no conflict at all), there are uncertainties and there are contradictions.

Uncertainties Uncertainties are found when given an attribute, there is at least one source that knows the value and others do not. This is a conflict between one or more '`NULL`' values and at least one 'non-`NULL`' value.

A 'NULL' value is usually used to indicate a value is unknown. It might also be used to indicate the attribute is not applicable for the given entity.

Solving a conflict with 'NULL' values is easier than solving contradictions.

Contradictions A contradiction happens when two or more sources claim different values for an attribute of an object. The reasons for contradictions can be typo's, semantics (e.g. multiple ways to describe the same thing ('Microsoft Windows 7' vs 'Windows 7')), one source being outdated, etcetera.

3.5.2 Information fusion strategies

The authors of [8] identify several strategies for merging data with uncertainties and conflicts. These strategies themselves fall into three categories as outlined below.

Conflict ignorance Strategies that fall under this category do not decide about the conflicts at all. They do not need information about conflicts as it is not used. Two strategies in this category are *Pass it on* and *Consider all possibilities*.

Pass it on With this strategy, the conflicting value are simply passed to another party, for example the user or some other application, which has to make the decision.

Consider all possibilities Using this strategy, all possible values (as given by the sources) are given. Effectively, instead of choosing a value, this strategy just chooses all of them.

Conflict avoidance Strategies that fall under this category, as the name suggests, do not solve conflicts directly. These strategies make the decision of whether to handle conflicts and how before inspecting the values. Because the decision is not made while inspecting the values, these strategies are computationally more efficient.

Take the information This strategy basically takes the information that is available and ignores information that is not available. This essentially means that 'NULL' values are ignored and when there is an uncertainty (conflict between 'NULL' and 'NON-NULL' values), the 'NON-NULL' value automatically 'wins'. This strategy does not however take conflicts into account and therefore is only applicable to situations that have only uncertainties and no contradictions.

No gossiping This strategy is quite simple. The idea is that if there are inconsistencies and no 'protocol' to solve them, why not ignore them. This strategy only results into sure facts, i.e. information that all sources agree on and all inconsistencies, both uncertainties and contradictions are simply ignored.

Even though this looks like a conflict ignorance strategy, it is not as it is aware of the conflicts and actively makes the decision to ignore them.

Trust your friends This strategy bases its decision of what to do with conflicts on someone else. The idea is that someone else decides what the correct value or strategy is, the trustworthy sources are decided up front and the strategy applies that decision to all data values it encounters.

This decision can be made by the user, but could also be made automatically, for example based on which source is the most reliable source or the largest, cheapest, etcetera. The decision can also be made by an algorithm as illustrated in [59].

Conflict resolution Strategies that fall under this category resolve conflict by regarding all metadata and data values before making a decision. As expected, this is computationally more expensive than strategies in the aforementioned strategies.

There are two forms of strategies in this category: deciding strategies and mediating strategies. Deciding strategies choose a value from the values that are delivered by the sources. This decision can be based on only the data values or also on the metadata, just as conflict avoiding strategies. Mediating strategies try to mediate. Therefore, they might arrive at a value that is not among the available values. They might create a new value.

Cry with the wolves This strategy is based on majority voting. This means that the strategy assumes that correct values are more persistent that false values. Therefore, it will choose the value that is reported by the majority of the sources. To break ties, additional criteria are needed.

Roll the dice This strategy just picks a random value from all the available ones. If there is no information that could help in choosing the right value, picking a random one is a good choice, anyone of them could be correct. As this strategy is basically no more than generating a random number and picking a corresponding value, this strategy is computationally cheap.

Meet in the middle This strategy is a mediating one. It tries to find a value in the middle that is as close as possible to all the others. This value does not necessarily exist yet among the possible values. Besides the median, as described here, another example includes taking the average.

Keep up to date This strategy assumes that the most recent values are more trustworthy than older values. This requires that sources also report when they established their values.

3.6 Network Architectures

A common campus network architecture that is used for corporate networks is a 3-tier or 2-tier hierarchical architecture. [14] On a campus there are often multiple buildings that need to be connected to each other and the outside world. For these types of businesses, a 3-tier network is suitable, as well as for large office buildings with multiple floors/areas/wings and a lot of employees. For small businesses or buildings, a 2-tier network is more suitable. Figure 9 shows a comparison between how a 3-tier and 2-tier network look like.

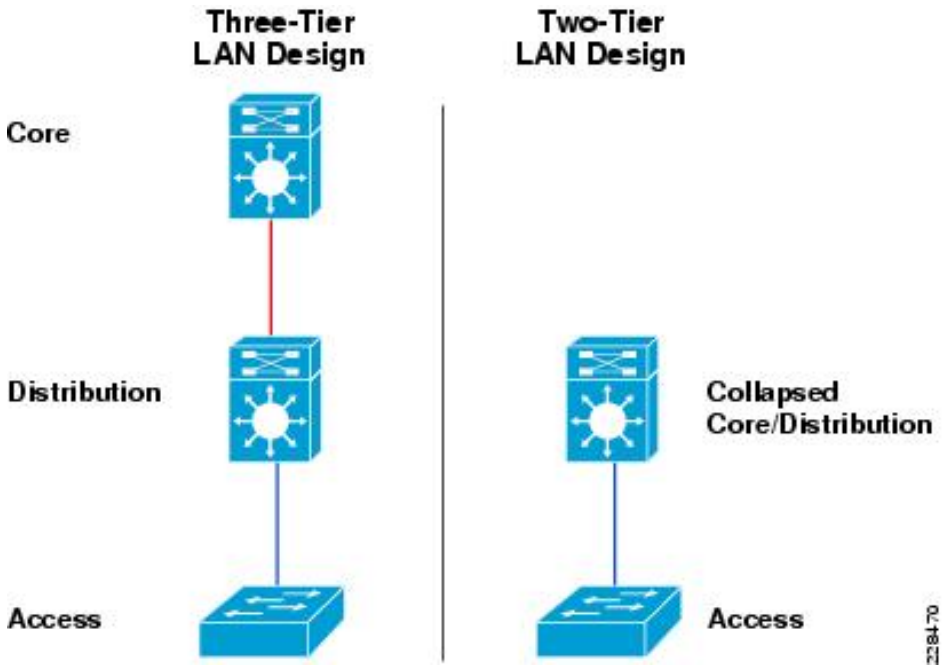


Figure 9: Schematic view of 3- and 2-tier networks [15]

3.6.1 3-tier network architecture

A 3-tier network architecture has 3 layers: the access layer, the distribution layer and the core layer. Each of these layers has its own function and purpose as explained below.

Access layer The access layer is the first layer of the 3-tier architecture and is the layer that all devices, such as PCs, printers, etcetera, connect with. In an office building, the access layer contains amongst other the ethernet plugs in the walls, as well as the wireless networking access points.

Distribution layer The distribution layer sits between the access layer and the core layer. Its function is to aggregate the access switches and manage traffic within the combination of access and distribution layer (i.e. it manages traffic flows from end devices in the access layer). The distribution layer also has to connect the access layer to the core layer. Effectively, it keeps traffic from the access layer that does not need to go outside away from the core layer.

Core layer The core layer is the backbone of the network. It provides connectivity between the different parts of the network. In the case of a campus, each building would have its own distribution and access layer and the core layer provides the connection between the buildings as well as access to the internet. If the core is down, the network is down and therefore, the core must be setup redundantly such that if one component goes down, the others can take over until the component is repaired or replaced and working again.

3.6.2 2-tier network

In a 2-tier network, which is suitable for small buildings or businesses, the core and distribution layers are combined, such that there are only an access layer and a core/distribution layer.

3.6.3 Network topologies

There are several different network topologies that one can use to setup a network. Choosing the right topology is important to setup a good network. The list below shows some basic network topologies [61].

- **Bus topology**
  A network based on the bus topology uses a common backbone (cable) for communication. Each host is connected to the backbone and can see all traffic on the bus. Hosts ignore traffic that is not meant for them. This topology was used in the early days of networking, but nowadays it is obsolete.

- **Ring topology**
  Using ring topology, all computers in the network are forming a ring. Each host as two connections, one to each of its two neighbours. The traffic goes in one direction from source to destination. Each host in between will just retransmit incoming traffic to the other connection unless the host itself is the destination. Adding and removing hosts to/from the network interrupts the network as it cannot function when the ring is not closed. This topology is therefore hard to maintain and therefore also obsolete.

- **Star topology**
  In a star network, there is one central hub, which connects to every other host in the network. All communication from the hosts in the network must pass through the central hub. If this central hub is not a switch, this topology basically acts as a bus topology. If the central hub is a switch, the network functions as a star network.

- **Tree topology**
  This topology is merely an extension to the star topology. The original star topology has not only hosts connected to the central hub/switch, but each of the hosts can again be the central hub/switch of a 'sub'-star network.

- **Mesh topology**
  In a mesh network, there exist multiple routes to connect to another device. For example, if host A has a connection with host B, they can communicate with each other. However, if this connection breaks, they can still communicate with each other via host C, provided there exists a path via host C.

Which topologies are used within corporate networks?

The 2- and 3-tier hierarchies described above work with an access layer where end devices connect to the network. This is mostly done using switches [22], therefore in the access layer, a star network topology is common. In the distribution and core layers, availability is important and the nodes in those layer are setup redundantly. As the network must recover to using a backup node when a primary node fails, the distribution and core layer nodes are interconnected. These layers make use of a mesh topology, which can be fully connected.

3.7 Summary

As shown in this section, there are lots of technologies available to use for automatically generating a model of an IT infrastructure. It is possible to use active scanners to probe devices for information, passive scanners to capture network traffic, agents to have full access

to devices and collect information directly on the device, management systems to gather (possible non-technical) information about devices.

Collected information can be stored in different types of databases, such as relational databases. There are also NoSQL databases, such as column-oriented databases, key-value databases, document-oriented databases, graph databases, etcetera. There are also databases which combine different types of NoSQL databases.

There are many different types of merging techniques, ranging from simple techniques to choose data from a source over another, such as asking some other party to decide, to more involved techniques, which look at the data, such as choosing the most recent value or take a mean value.

The next section discusses which techniques and methods are used in the proposed method of automatically generating a model of an IT infrastructure.

4 Methods

This section describes the techniques and methods that are used to generate a model of an infrastructure and also why these specific techniques and methods are chosen. Section 4.2 describes the data model that shows how the final IT infrastructure model should look like, section 4.3 how the sources can be used and describes the database systems that can be used for the data and which database systems are actually chosen and section 4.4 describes the merging process that is used and how the model is generated.

4.1 Proposed process of generating the model

This sections describes the proposed process of generating a model of an IT infrastructure. It reuses the structure from section 3.1. Figure 10 shows a schematic overview of the proposed method.

1. **Creating data model**
   To generate a model of an IT infrastructure, you need information about the infrastructure. To know which information you need, it is helpful to have a data model that tells you wat you need. Section 4.2 describes the data model that we created for the final IT infrastructure model. Based on the information that is needed from the infrastructure, sources can be selected to provide the requested information.

2. **Collection and storage**
   The next step is to actually collect the data about the infrastructure. This involves setting up the selected sources, collecting the information, cleaning and filtering the data, etcetera. The collected data must be stored somewhere such that it can be used. Section 4.3 describes the process of collecting and storing the data about the infrastructure and the databases that are suitable for storing the collected data.

3. **Generating model**
   To arrive at a model, the collected data must be used and merged to generate the model. Section 4.4 describes the proposed merging process.

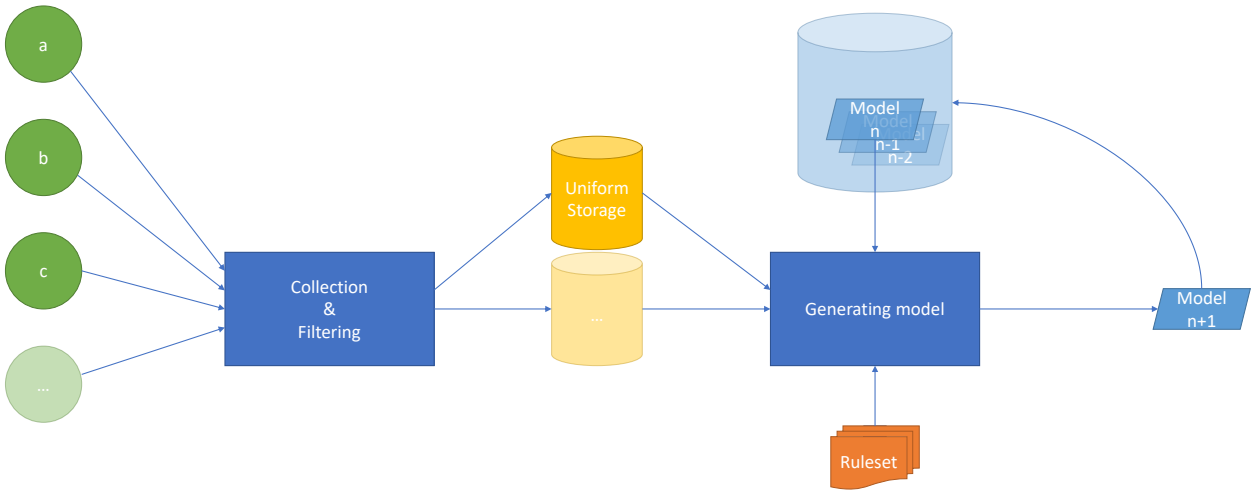


Figure 10: Proposed process of automatically generating a model

4.2 Data model

This section describes what the data model for the final IT infrastructure model looks like. The goal of the model is indispensable in determining what information to include in the model. As the question in this thesis is not about a specific use case of the model but more about whether it is feasible to build a model of an IT infrastructure, the model is a proof of concept and therefore, the model is concise.

The model contains information about the network that a device is in, what operating system type and software types it runs and the communications between applications. To store the model and to reason about it, we need to know what it looks like. For this purpose, we created a data model describing the entities in the model and how they are related to each other.

The data model is explained in section 4.2.1, which describes connections of devices, section 4.2.2, which describes facts about devices, such as software types run by devices and section 4.2.3, which describes interconnections between applications on the network.

4.2.1 Connections

In an infrastructure, devices are typically connected to a network. In a larger infrastructure, devices are often connected to VLANs to separate groups of devices from each other. In the data model, these VLANs are denoted as `VirtualNetwork`. The notion of a virtual network is used due to the fact that most companies use VLANs in their network to separate hosts into distinct groups or departments. If VLANs are not used, there is just one 'virtual' network.

Apart from being connected to a virtual network, devices (denoted as `Device`) usually offer some function (`Function`). (such as a web server or an email server, etcetera). The router is a special kind of device and is denoted as `GatewayDevice`. These connections are described by the following facts and can be seen in figure 11.

- `Device` is connected to `VirtualNetwork`
- `Device` offers `Function`
- `VirtualNetwork` has `Label`
- Each `GatewayDevice` is an instance of `Device`

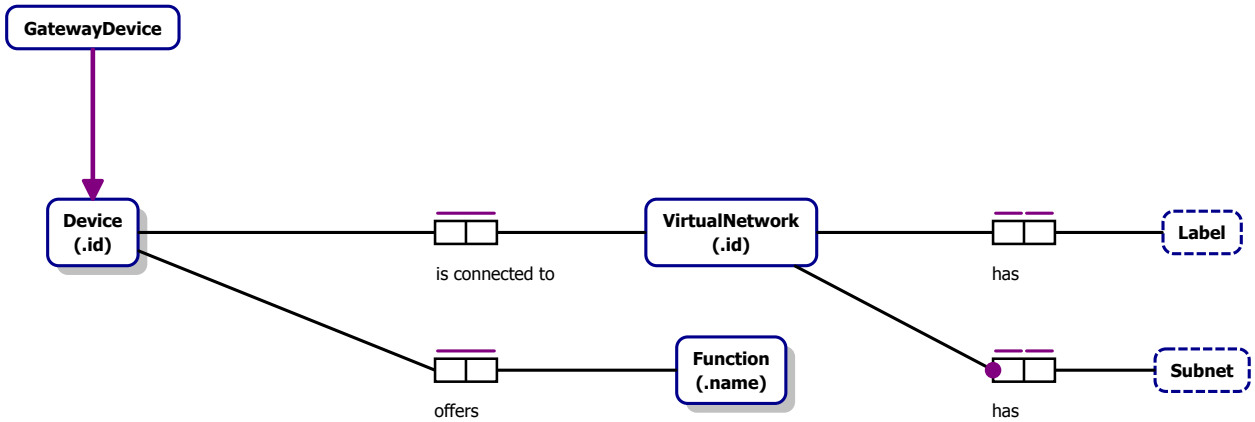


Figure 11: Data model of connections in an infrastructure

4.2.2 Device facts

The data model also contains information about devices themselves. Devices run an operating system, which has a certain version and name. Additionally, devices run application, which also have a version and a name.

These relations are depicted in figure 12. Each device runs an operating system, which is denoted as `OperatingSystemType`. The data model only has a notion of a type of operating system as information about an instance of an operating system does not add value to the model. The same holds for applications, which are denoted as `ApplicationType`. Both `OperatingSystemType` and `ApplicationType` have a name and a version. The combination of name and version are used to identify the application or operating system type, as indicated by the external uniqueness constraint.

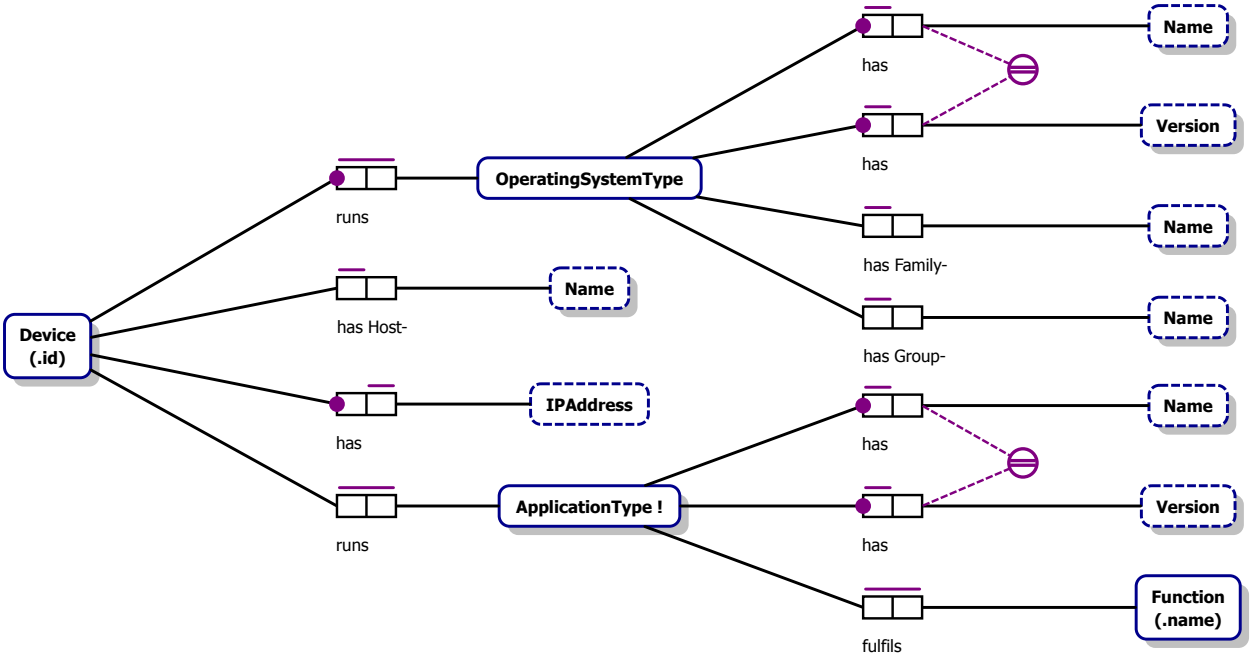


Figure 12: Data model of device facts

In addition to having a name and a version, an `OperatingSystemType` is part of an `OperatingSystemFamily`. An operating system family can be Windows, Unix, et cetera. This information can be useful to make assumptions about security issues, as a specific vulnerability could for example only impact Windows based operating systems.

As mentioned in section 4.2.1, a device offers a function. The functions offered by a device are usually made possible by an application. Therefore, the data model has the relation that an `ApplicationType` fulfils a `Function`.

- `Device` runs `ApplicationType`
- `Device` runs `OperatingSystemType`
- `ApplicationType` has `Name`
- `ApplicationType` has `Version`
- `ApplicationType` fulfils `Function`
- `OperatingSystemType` has `Name`
- `OperatingSystemType` has `Version`
- `OperatingSystemType` is part of `OperatingSystemFamily`

4.2.3 Interconnections

The data model should also model the connections between devices, as these connections are interesting for security purposes in the final model of the infrastructure. Because a connection is usually made by some application, which communicates with some other application, the data model also includes instances of applications, denoted by `Application`, which communicate with each other, as seen in figure 13.

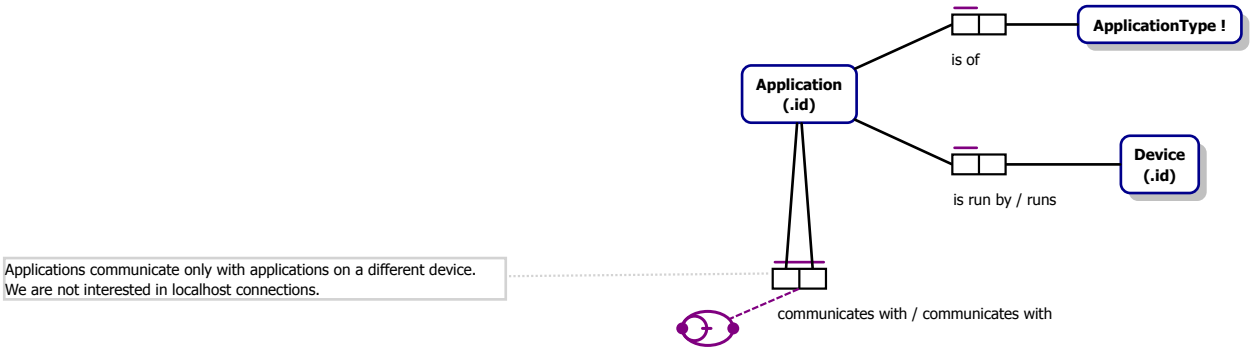


Figure 13: Data model of interconnections

These `Application` instances are of an `ApplicationType`, therefore inheriting a name and version. Furthermore, an `Application` is run by a device. This relation is also present for `ApplicationType` (figure 12), however it is possible that for one `ApplicationType`, more than one `Device` runs that `ApplicationType`. In contrast, each `Application` is run by at most one `Device`. Therefore, it is possible to identify communicating devices through the `Application` instances.

The ring constraint on the `Application` communicates with `Application` fact means that this fact is both symmetric and irreflexive. Being symmetric means that if A communicates with B, then B also communicates with A. Being irreflexive means that no application communicates with itself.

Please note that it is possible to have two applications on the same device communicate with each other and that the data model does allow that. However, two applications communicating on the same device is not interesting for the final model. Unfortunately, ORM does not support the restriction that the two applications must by run by a different device.

- `Application` is off `ApplicationType`
- `Application` is run by `Device`
- `Application` communicates with `Application`

4.3 Collecting and storing information

The information that is collected by sources comes in various formats. The output from a tool can be stored in XML files, JSON files, plain text files, etcetera. Some tools do not have output files at all, but provide a web interface that displays the collected information. Such tools usually also provide a REST API to provide the data.

To collect the data from the tools, they must be executed and the result must be saved. For example, a network scanner must be installed and then be executed to scan the network. The result, in whatever format, can then be used for generating the model.

For using the data, it is beneficial to have one location and interface to query the data. There are a few reasons why this is useful. First, it allows preprocessing the data, for example,

converting dates to a uniform format. Second, it creates an abstraction layer between the output of the sources and the program that processes the data. When building the model, it is no longer necessary to know the details of each source's output. The data collection is isolated. Therefore, after collecting the data, the data is stored in a database.

Another reason to store the data in the database is the requirement to have snapshots. Each scan must be stored separately in the database, instead of overwriting the previous data. This allows for generating a model based on previous scans, which allows to view the situation as it was at an earlier point in time.

Data that is collected by network scanners tends to differ from device to device. Not everything is known about all devices and some type of device has different properties than others. Because of these characteristics, a document based database is suitable for storing this type of data as document based databases do not define a predetermined schema that will result in NULL values when an attribute is not known.

The raw data that is collected by passive scanners is time based data, because they intercept network traffic that passes through the connection. This raw data is packet-based, which means that each packet is a new entry. Furthermore, each packet has usually the same kind of information, such as destination address, destination port, source address, source port, etcetera. Therefore, a column based is suitable as column based databases are suited for time-series data. However, if the passive scanner also does processing of the data, it might present the data in a form that is more suitable for another type of database.

In a network, devices are connected to switches and routers, which in turn are also connected to other switches or routers and to other devices, a network is a graph. Therefore, a model of a network is graph based, because of all these connections to other devices and therefore, a graph database is suitable for storing this information.

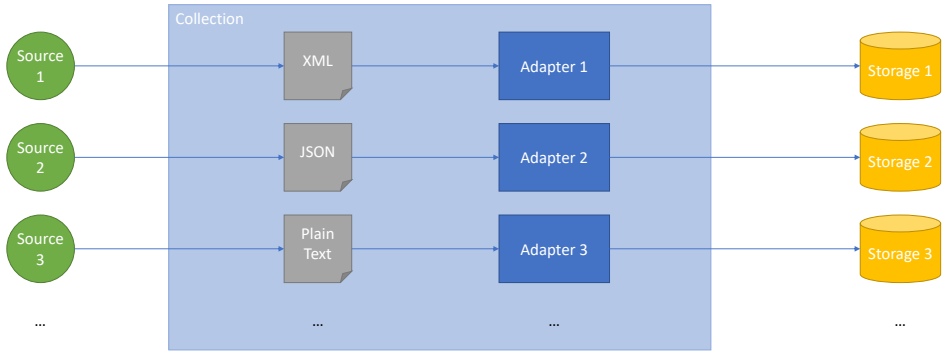


Figure 14: Process of collecting the data and storing it

To store collected information in a database, it must be read and transformed in a structure that is understandable by the database software. Therefore, if the tool at hand does not offer a way to save data directly to a database, an adapter must be written that can parse and extract the data from the output files of the selected sources and store it in their respective databases. To generate these database, an ORM data model[19] can be created for each tool based on the information that is provided by each source, because the tools for creating an ORM data model offer the ability to generate a database based on the data model. Figure 14 shows a schematic overview of the process of collecting data and storing it in the databases.

[19]See the Data Models appendix for the data models (appendix C)

4.4 Merging collected information

Having the collected data in a database, the data can be used to generate a model. First, the source data is retrieved from the database and then it is merged.

In the proposed method, we chose to do the merging according to a set of rules. Such a ruleset can be used to transparently control the merging process and the rules could be maintained by someone who does not have the expertise to edit the merging algorithm.

There is one ruleset for each fact of the data model that describes what the final IT infrastructure model must look like, which can be found in appendix A. A number of sources is available for the generation of the model, however, a source does not necessarily provide information for each and every fact in the final model. For example, think of a source that does not have information about the operating system of devices. This source is then not used for a fact that states that a device runs an operating system.

Fact X: <Description>

<image showing fact>

Data Sources

- Source 1
- ~~Source 2~~
- Source 3
- ...
- Source N

Rule

If Source 1 has the info, use ...
Else if Source 3 ..., use ...
...
Else if Source N ..., use ...

Clarification

Source 1 <Description>
Source 2 <Source 2 does not provide this information, because ...>
Source 3 <Description>
...
Source N <Description>

Figure 15: Ruleset structure

Every ruleset follows a predefined structure, as seen in figure 15. First, the fact is shown using an image of the fact as it is represented in the data model. Then the list of sources used to generate the model is given. If a source does not provide the necessary information about this fact, it source is crossed out. The list of sources is followed by the rules, which dictates that if a source has the information, the information is used. Otherwise, if the next source has the information, use the information, etcetera. The ruleset is concluded with a clarification section that explains why certain choices are made and under which conditions it might not be able to provide the requested information. If a source cannot provide the information, the clarification explains why it cannot provide the information.

In the rulesets, there is a specific order in which sources are used. This order is predefined based on the reliability and completeness of the sources and must be determined by the user. As explained in section 3.5, there are many strategies to merge the data. The ruleset implements two strategies that were explained in section 3.5: 'Take the information' and 'Trust your friends'. The 'Trust your friends' strategy is found in the fact that the sources have an ordering, set by the user, according to which their data is used. The 'Take the information' strategy is found in the fact that if a source does not have the requested information, the next source is queried instead.

After the merging is complete, you have information about the network that is as complete as possible given the input sources. This information is transformed to the structure of the final IT infrastructure model and saved in the database. Each new model should be stored separately, such that it is possible to go back in time and, for example, see changes between the current model and an earlier model.

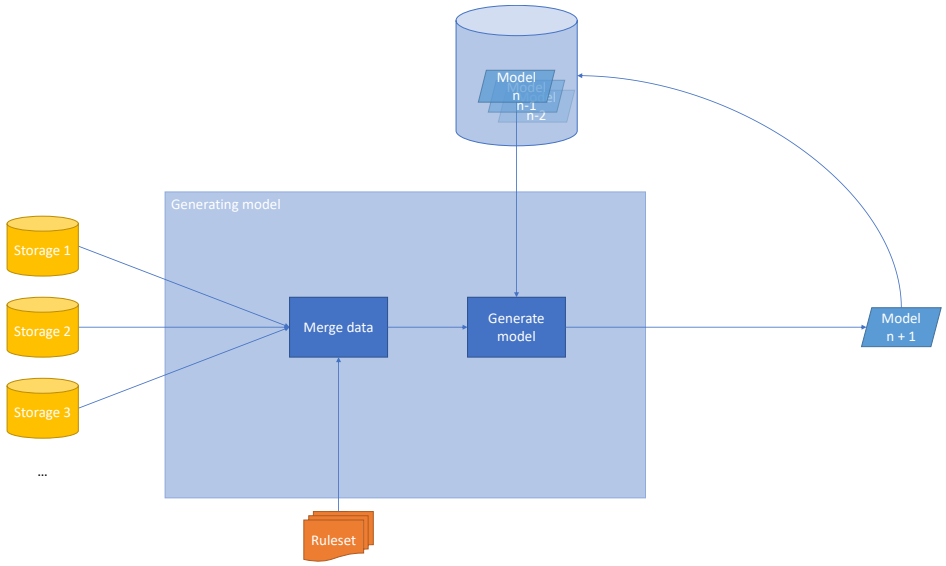


Figure 16: Process of merging the data and generating a model

4.5 Adding a new source

While creating the proof of concept[20], we found that we needed information that could not be detected by network scanners. As seen in section 4.2, we also need information about the function of devices. What is the reason they are used? Examples would be 'workstation', 'webserver', etcetera. This question cannot be answered by network scanners but is usually recorded in some management system.

The fact that we need more information also led to the realisation that using multiple sources is nice, but it should be possible to add more sources if it turns out that more information is needed that cannot be provided by existing sources.

To overcome this issue, the method allows for extending the sources by considering the source adapters to be separate modules. By separating the logic of parsing the source data and storing it, it is possible to write your own module and 'plug it in'. This makes the method flexible as it can be adapted to your own needs by using your own sources instead of being forced to use the sources which are already supported.

[20]See the next section (section 5)

5 Proof of concept

This section describes the proof of concept that implements the method. The goal of the proof of concept is to show that it is feasible to use multiple sources to generate a model of an IT infrastructure.

The proof of concept has two parts. On the one hand there is the collection and storage of data and on the other hand is the mapping and merging of data. Section 5.1 describes the sources that were chosen for the proof of concept and what information is needed to generate a model for cyber security analysis. In section 5.2, the process of collecting and storing the data is described. In section 5.3, the process of mapping and merging the data is described. Finally, section 5.4 describes how a new source was used in the prototype.

5.1 Selecting sources

To select sources for building a model, some free and open source tools were installed on a test network and used. Based on the information they provided, the tools are selected. The list below shows an example of what kind of information is useful for cyber security analysis. To build a model, you need to select sources such that the combination of selected sources provides the following information. The italic points are collected by the chosen sources for the proof of concept.

- **Network related**
  - *Which devices are on the network*
  - *IP and Mac addresses of the devices*
  - *Open ports*
  - *Whether two devices communicate with each other*
  - Firewall configuration
- **OS related**
  - *Installed operating system type*
  - *OS version*
  - *Hardware information*
- **Application related**
  - *Installed software types*
  - *Software versions*
  - *Running services*
- **Security related**
  - Security status
  - Vulnerabilities
  - Last updates virus/malware DB
  - Configuration errors

Appendix B provides an overview of the tested sources and their advantages and disadvantages. Based on these advantages and disadvantages, as well as the selection criteria above, Nmap, Open-AudIT and Snort were selected to provide information for the model.

Nmap is chosen to retrieve the basics about the devices in the network and to find new devices within subnets with basic information. Nmap is a useful tool for detecting devices in the network and to estimate what kind of software they are running.

Open-AudIT is chosen for the more detailed information it provides. As it has access to all information about a device using remote access or a running agent, it can collect lots of information about the device. It provides information about the operating system, installed applications, but also about the hardware, such as CPU, GPU, motherboard, network card. In addition, it is possible to collect information about user accounts.

Finally, Snort is chosen to detect relations between devices and applications. As Snort is a passive scanner, which means it can intercept traffic, it is a good source for detecting which devices communicate with other devices and as the originating and target port number are known, it is possible to deduce which application is responsible for the communication.

In addition to these tools, we created a CSV file as a new data source with properties of some hosts to provide information that can not be detected by a network scanner, such as the function of a device. The purpose of this source of information is to represent output from some management system that records this information.

5.2 Data source adapters

For this purpose of storing the data in a database, an adapter is needed for each source of information. Before the adapter can be written, we need to know what the database, which will hold the collected data, looks like. ORM, which is used in section 4.2, is not only useful for creating a data model, it also allows for translating the data model to a working (relational) database. Therefore, we created an ORM data model for each source of information, such that we have a convenient way to create the intermediate databases as well as a comprehensive overview of the information that we can use from each source.

The data models that were created for each source do not necessarily contain all the information that is offered by the source. A choice has been made in which information is part of the data model and which information is not. The information that is offered by the data models is chosen based on usefulness for the proof of concept of modelling a network based on collected information as well as usefulness for cyber security information. For the latter case, think of for example information about device hardware, which is not used for the proof of concept, but useful for cyber security analysis.

Using the data model, it is possible to write adapters to store the raw source data in a database that is generated using the data model. The method uses an importer program to import the data from the sources into the database. As each source uses its own format to present the collected data, there is no uniform way to write an adapter that can process them all.

For each source, an adapter is written, such that the data can be stored in the database. Because it is the intention to make the method flexible, the logic for importing the data is confined in separate modules. For each source, a module is written that can import the data and also load the the data from the database again. These modules are built using a structure which is known to the importer program. Therefore, it is possible to write a new module for a new source and plug it in to the program. Figure 17 shows a schematic overview of the process of using the Network Scan Importer/Mapper.

Alternatively, it is possible to write a source module that does not use a database to store the data, but directly retrieves the data from the source and presents it to the mapping stage. This alternative route is visualised by the 'Custom Source Module' in figure 17.

In figure 17, the ruleset step is faded, because in the proof of concept implementation, there is no functionality to use a ruleset yet. Even though the order in which sources are trusted can be customised in the config where the supported source modules are defined,

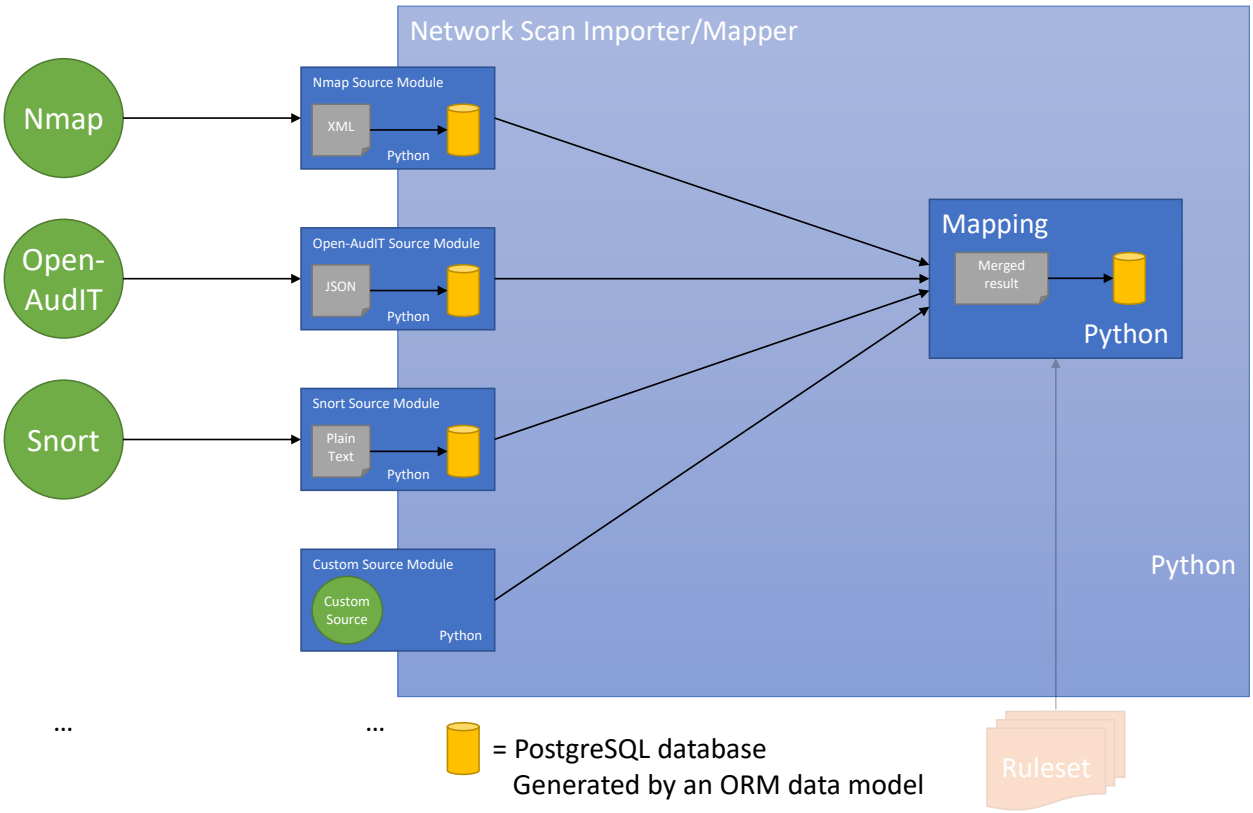


Figure 17: Schematic overview of the importer/mapper program

there is no support for reading rules from a separate party such as a ruleset file or a database.

Each adapter must be individually used by the user to import the data into the database. Then the user can map the data to the target model using the data that is stored in the source databases.

Within each source module, the following strategy is followed. A structure of classes is built that closely follows the data model. For example, for the Nmap data source, there is an `NmapScan`[21] class, which has a version integer, a `ScanConfig` object and a `ScanResult` object. The `ScanResult` object in turn has a `datetime` and a list of `Hosts`, etcetera.

These classes are all derived from an abstract base class that has `parse`, `save` and `load` methods. The advantage of this strategy is that the code is easy to follow, because to parse the output of a scan, one only needs to call the `NmapScan.parse` method and then the whole scan is transformed into these objects. The whole scan can then be stored in the database using a simple call to the `save` method of the resulting scan, which will call the `save` method of the `ScanConfig` and `ScanResult` objects, etcetera.

In the same way, the scan can also be loaded again from the database. However, the difference is that the scan is lazily loaded, which means that the `ScanResult` object is retrieved from the database if and only if one tries to use the `scanResult` property on the `NmapScan` object.

For the proof of concept, a relational PostgresSQL database is used to store both the individual source data and the final model data. PostgreSQL is chosen, because of the data models that are made for the final model and each individual data source[22]. The tool that

[21] See appendix C.1 Nmap Scan Result for the data model from which these class names are derived.

[22] See section 4.2 and appendix C.

we use for creating the data models[23] natively supports generating a PostgreSQL database from the data model, which is advantageous when a data model still has regular changes. Additionally, due to the small scale of test network, the chosen database type does not have a significant impact yet.

The snapshotting requirement is (partially) fulfilled here, because each scan is saved under a different ID, which means that each time a new scan result is imported into the database it is added to the database and does not overwrite existing data.

Appendix C describes the data models of the sources that were selected for building the model. Appendix C.1 on page 78 discusses the data model for the Nmap scan results, appendix C.2 on page 80 shows the data model for Open-AudIT information and appendix C.3 on page 84 describes the data model for information collected using Snort.

5.3 Mapping to model

The mapper from the separate data sources to one model uses the source modules to retrieve the source data, which can either be loaded from the database or by any other means, depending on the implementation of the source module. The mapper tries to make it relatively easy to add another data source to the system. The source module should include an object, representing its data source, with a `getDeviceMappers` method. This method should return a list of `DeviceMapper` objects, which should contain as much information as the data source can offer.

At the moment, the `DeviceMapper` class only holds enough information to fill the data model described in section 4.2. If more information is needed, the `DeviceMapper` class needs to be extended and all existing data source adapters must be updated to properly include the newly requested information in their `DeviceMapper` list, if applicable.

The `DeviceMapper` class is also responsible for the merging procedure, as explained in section 4.4. After merging the `DeviceMapper` objects from each data source, there is only one list of (merged) `DeviceMapper` objects left, which is used to build the model.

For deciding which information should be included in a `DeviceMapper` object from each source, fact derivations are used. These fact derivations can be found in appendix A. These fact derivations describe which of the used source have information about the fact and in which order this information should be used. Below is an example of a fact derivation.

In the fact derivations, there is an assumed order of trustworthiness for the sources. The rules are written based on this order. The following order of trustworthiness is the result of reviewing the sources. This order is the same in each fact derivation.

1. Open-AudIT
2. Nmap
3. Device Management Report
4. Snort

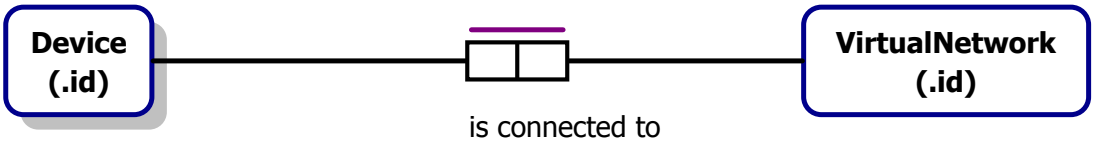
Up front, each source is given a weight and in the merging process, all the time the source with the highest weight is trusted and its information is used. In cases where only one source has information and the other has not, the information is taken, irregardless of the source. At the moment, the fact derivations themselves are not used to control the merging process in the proof of concept implementation, but the idea behind them is used.

Currently, the merging strategy used by the proof of concept is a combination 'Trust your friends' and 'Take the information'. This combination of strategies is chosen for two

[23] Object Role Modeling, see section 3.2.

reasons. First, it computationally efficient and in the test environment, we have access to a data source which has full access to most devices and therefore has reliable information.

Fact C1: Device is connected to VirtualNetwork



Data Sources

- Nmap
- Open-Audit
- Snort
- Device Management Report

Rule

If Open-Audit detects a host, use fact '`Device` has `Address`' in combination with '`Network` has `Subnet`' or use fact '`NetworkCard` has `IPvXAddress`' in combination with '`NetworkCard` has `IPvXSubnet`'.
Else if Nmap detects a host, use the facts '`Host` has `Address`' and '`ScanConfig` scans `IPOrSubnet`'.
Else if the device has a record in the device management report, use the fact '`Device` has `IPAddress`'.
Else if Snort detects a host, use fact '`HostApplication` has `IPAddress`'.

Clarification

Nmap If an Nmap `Host` has an `Address`, it is connected to a network. Even though the network is not necessarily a virtual network, the fact can still be used. The notion of a virtual network is present due to the fact that most companies use VLANs in their network to separate hosts into distinct groups or departments.

Even though the (IP) address itself is not a network, it can be used to determine to which network a host is connected or at least determine which hosts are part of the same network. Using the information about which IP addresses and subnets were scanned, it is possible to find the corresponding subnet. In the case that only IP addresses were given as scan input, it is possible to guess a subnet by comparing the address to default private network subnets and other hosts that may have the same prefix, which is likely to be the subnet they belong to.

Open-AudIT Given an IP address and subnets of networks, it is possible to determine whether a device with a certain IP belongs to one of the networks (subnets).

In case Open-AudIT had access to a device and has information about its network card, it can use the IP address and subnet information from the network card acquire the relation between the device and a network.

Snort If Snort detects traffic, it records a source IP address and a target IP address. These IP addresses can be used to determine the existence of devices and to find out in which network they are. However, extra information about the subnets that are used within the network is needed to match an IP address to a network. Snort can also detect traffic to and from servers outside the network, which should be ignored as being part of the network within the model.

Device management report If there is a record for the device, there is an IP address. These IP addresses can be used to determine the existence of devices and to find out in which network they are. As mentioned above however, the IP address alone is not enough.

5.4 Adding a new source

As explained in section 4.5, the sources we initially chose could not retrieve all information we wanted. We also needed the function of devices, which could not be found by network scanners. Open-AudIT does have a function field, but because we used it more as a scanner than a management system, it was empty.

Because we want the method to be flexible, we decided to simulate a management system, as a user of the method may not use Open-AudIT as a management system, but something else. This other management system should then be added as a new source. We simulated the management system by means of a CSV file, an example of which can be found in table 1.

CompanyIdentifier	Hostname	IPAddress	Functions
nmapScanner	nmapScanner	10.0.0.3	Scanning network
openAudITScanner	openAudITScanner	10.0.0.5	Scanning network
snortScanner	snortScanner	10.0.0.201	Passive scanning|Being a gateway
snortScanner	snortScanner	192.168.11.201	Passive scanning|Being a gateway
snortScanner	snortScanner	192.168.10.201	Passive scanning|Being a gateway
ordinaryHost2	ordinaryHost2	192.168.11.10	Webserver
ordinaryHost4	ordinaryHost4	192.168.11.12	Mailserver

Table 1: Example contents of CSV file simulating a management system

The idea behind the CSV is that it contains the functions of devices. The first column is a company identifier, which companies use to identify devices. This does not necessarily have to be the same value as the hostname. The final column contains the functions of the corresponding device. If a device has multiple functions, they may be separated by a vertical bar (|).

Adding a new source may also mean that the target model needs updating, because the reason to add the source was to add more information to the model. This is also possible, however, is not as easy as adding a new module. To add more information, one has to update the data model of the target model and update the mapping stage to include the new information, including extending the 'DeviceMapper' class and writing merging logic for the new information. However, as the data model is created using ORM, the data model is

semantically stable, which means that extending the data model does not change anything except adding the new facts that come with the new information.

In all cases, to maintain the transparency, the fact derivations should be updated to include how a new source is to be used in the merging stage as well as how extended parts of the data model should be filled in by the different sources, both the existing and new sources.

6 Results

This section describes the result of the proof of concept of the method. First section 6.1 describes the test environment that is used to test the method and section 6.2 describes the findings as a result of testing the method on the test environment.

6.1 Test environment

In order to do experiments to find out what kind of information can be collected and to verify the method of automatically generating a model, an actual test network is needed. The test network is entirely created within a virtual environment. This virtual environment runs on 1 real life device within VirtualBox[24].

The network has a small number of hosts, as listed in table 2. Some of these hosts run network scanners and tools and others merely exist and do nothing. The network contains a few scanners, such as Nmap, Open-AudIT and Snort, which are described in appendix B. There are also some ordinary hosts, running different versions of Ubuntu, a host running CentOS, a host running Windows and one host that is not running an agent. All these hosts are divided over three subnets that mimic the concept of VLANs.

Host	**Subnet**
root	192.168.10.0/24
snortScanner	192.168.10.0/24, 192.168.11.0/24, 10.0.0.0/24
nmapScanner	192.168.10.0/24, 192.168.11.0/24, 10.0.0.0/24
openAudITScanner	10.0.0.0/24
observiumScanner	10.0.0.0/24
nagiosScanner	10.0.0.0/24
ordinaryHost1	192.168.11.0/24
ordinaryHost2	192.168.11.0/24
ordinaryHost3	192.168.11.0/24
ordinaryHost4	192.168.11.0/24
ordinaryHost5	192.168.11.0/24
ordinaryHost6	192.168.11.0/24
minimalWindows10	10.0.0.0/24

Table 2: The hosts and their subnets in the test environment

Some hosts exist in multiple subnets. In case of the Nmap scanner, the reason is simple, because Nmap can only collect a MAC address if Nmap itself is part of the same subnet as the target device.

The Snort scanner is a different story. The Snort scanner is the gateway for every other host. The reason is that Snort is a passive scanner and all traffic must pass by it if it is to be intercepted by Snort. Because now all traffic, even to the internet, must pass by the Snort host, Snort can intercept it, such that it can be used for detecting communications between devices.

[24] Available at `https://www.virtualbox.org`

Network architecture

Conceptually, the test network is a simple star topology network. There are three subnets, connected to each other using the Snort scanner, which acts a gateway. However, the real life host is the root router for the virtual machines, which makes it a tree topology network.

For the proof of concept, the network is kept simple. Additionally, because the network lives entirely within a virtual (VirtualBox) environment, it is non-trivial to include routers and switches to emulate a 2-tier or 3-tier network architecture. However, network scanners should not have a problem with such an architecture, provided they have access to the entire network or have multiple instances that can reach different parts of the network.

6.2 Findings

Using the test network described in section 6.1, the method is tested. After running the scanners, feeding the results into the adapters and using the mapper to create the model, there is a model in the database. This is not yet interesting. Therefore, the model was used as an input the TNO's Cyber Security Decision Support tool.

It was possible to convert the model in the PostgreSQL database to the ArangoDB database and the structure of TNO's Cyber Security Decision Support tool, because TNO's tool has devices that are connected to a network, which is also represented as a device. Our model only has devices, but networks are not represented as a device. In the conversion, it was enough to generate a device representation of the networks in our model and convert our devices to TNO's device representation. The result is that the model can be easily loaded into TNO's CSDS tool, as seen in figure 18.

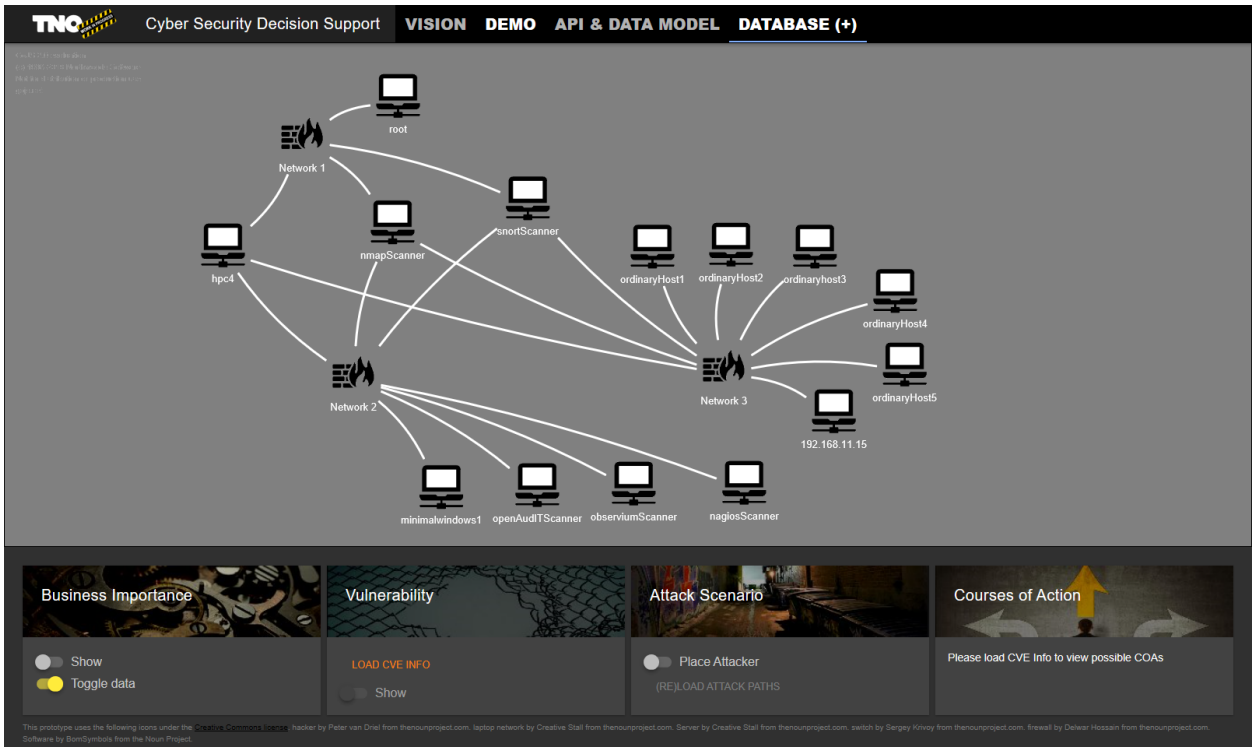


Figure 18: The test environment represented by the model in the TNO Cyber Security Decision Support tool

Please note that the layout in figure 18 is made manually. The conversion to the CSDS

tool does not automatically determine the best place to layout the devices. Automatically determining where each node should be placed is outside the scope of this thesis.

The information presented in figure 18, but also the information in the final model database is mostly a correct representation of the test environment as described in section 6.1. The main reason for this result is that Open-AudIT has access to all devices and can retrieve the true information about them. One exception is 192.168.11.15, to which Open-AudIT has no access. As a result nothing is surely known about this device. The other exception is hpc4, which is the host computer and is part of every (virtual) subnet.

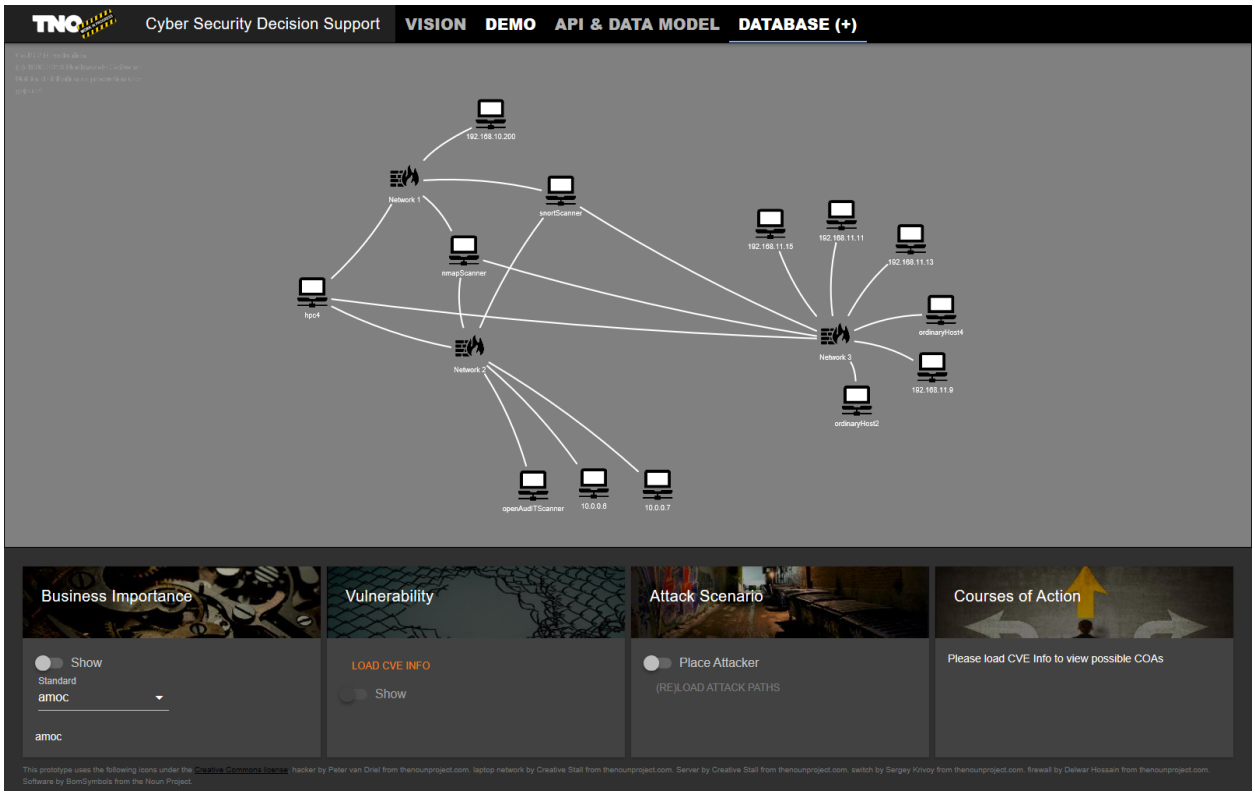


Figure 19: The test environment represented by the model in the TNO Cyber Security Decision Support tool, but without results from Open-AudIT

If we omit the Open-AudIT results, we get a similar figure as seen in figure 19, but this time there are a lot of devices that are only known by their IP address. As the scans were performed within an VirtualBox environment, it was not possible for Nmap to retrieve hostnames from other devices by reverse DNS or other techniques. The reason for the hostnames still present in the model are either because the hostname was given in the device management CSV file or Nmap managed to find a hostname anyway. The latter case is true for the nmapScanner itself and the hpc4 device, the name of which was found due to the fact that its name was available outside the virtual network.

We have similar findings if we list some of the collected information in a table per device. For example, table 3[25] shows the results for a selection of devices when only the data from Nmap is used to generate the final model. It can be seen that the hostname is detected for only some devices as explained above. Additionally, the operating system is guessed by Nmap, which is correct in most cases, though the guess itself is still broad. In case of the

[25]The first column ('Device') shows the device to which the row belongs. The remaining columns show the data that is recorded in the final model, based on the given input sources.

Windows device, the guess is incorrect. Additionally, the merging process did not manage to recognise that the Snort scanner device has multiple IP addresses. This is due to the fact that combining devices to be one device depends on IP addresses or hostnames to match.

Device	**Hostname**	**IP address**	**Operating System**	**Number of applications**
Minimal Windows 10		10.0.0.14	FreeBSD 6.2-RELEASE	5
Nmap Scanner	nmapScanner	10.0.0.3 192.168.10.3 192.168.11.3	Linux 3.8 - 4.9	3
Open-AudIT		10.0.0.5	Linux 3.2 - 4.8	5
Ordinary Host 1		192.168.11.9	Linux 3.2 - 4.8	3
Ordinary Host 2		192.168.11.10	Linux 3.2 - 4.8	3
Ordinary Host 3		192.168.11.11	Linux 3.2 - 4.8	4
Ordinary Host 4		192.168.11.12	Linux 3.2 - 4.8	4
Ordinary Host 6		192.168.11.15	Linux 3.2 - 4.8	3
Snort Scanner	_gateway	10.0.0.201	Linux 3.2 - 4.8	1
Snort Scanner		192.168.10.201	Linux 3.2 - 4.8	3
Snort Scanner		192.168.11.201	Linux 3.2 - 4.8	1

Table 3: Information retrieved about some devices using only Nmap

In table 4 we have results from the merged data from an Nmap scan, a Snort scan and the example csv file as presented in table 1. Because the csv file also defines some hostnames, the result shows more known hostnames. This also means that the merging process was able to recognise that the Snort scanner device has multiple IP addresses and combined them into one device entity. However, because the Nmap source has a higher score than the CSV file source, the merging process kept the '_gateway' hostname, which means that the Snort scanner device still has two entries in the table. If we were to trust the CSV file source over the Nmap source, the Snort scanner would be merged to one device with three IP addresses.

However, the extra sources do not offer information about running operating systems, which means that the guesses from Nmap are still used for the operating system column. The number of applications did increase in some cases, because Snort detects traffic between devices as well as the ports that the traffic originates from or travels to. Therefore it is possible to deduces that an application uses these ports.

If we also add the data from Open-AudIT to the merging process, we get the results in table 5. This time, the operating system is detected accurately. For example, the Minimal Windows 10 machine is correctly detected to have Microsoft Windows 10 Enterprise Evaluation 10.0.10240[26] running. Also the linux virtual machines are recognised correctly now. Whereas the guess from Nmap was a range of linux versions, the data from Open-AudIT shows us that the ordinary host machines are running different versions of Ubuntu and there is even one running CentOS. Additionally, the applications installed on the devices are detected by Open-AudIT, as seen in the increased numbers[27] in the 'Number of applications' column. As mentioned before, there is one host to which Open-AudIT does not have access,

[26]Please note that the operating system names in the table are shortened for readability.

[27]Please note that these numbers do not only include installed applications, but also installed libraries and such.

Device	Hostname	IP address	Operating System	Number of applications
Minimal Windows 10		10.0.0.14	FreeBSD 6.2-RELEASE	5
Nmap Scanner	nmapScanner	10.0.0.3 192.168.10.3 192.168.11.3	Linux 3.8 - 4.9	5
Open-AudIT		10.0.0.5	Linux 3.2 - 4.8	6
Ordinary Host 1		192.168.11.9	Linux 3.2 - 4.8	5
Ordinary Host 2	ordinaryHost2	192.168.11.10	Linux 3.2 - 4.8	5
Ordinary Host 3		192.168.11.11	Linux 3.2 - 4.8	7
Ordinary Host 4	ordinaryHost4	192.168.11.12	Linux 3.2 - 4.8	7
Ordinary Host 6		192.168.11.15	Linux 3.2 - 4.8	6
Snort Scanner	_gateway	10.0.0.201	Linux 3.2 - 4.8	1
Snort Scanner	snortScanner	192.168.10.201 192.168.11.201	Linux 3.2 - 4.8	5

Table 4: Information retrieved about some devices using only Nmap, Snort and the example CSV file

which is the 'Ordinary Host 6' machine. Because Open-AudIT does not have information about this machine, the guesses from Nmap are still used for this device.

Device	Hostname	IP address	Operating System	Number of apps
Minimal Windows 10	minimalWindows1	10.0.0.14	Windows 10	233
Nmap Scanner	nmapScanner	10.0.0.3 192.168.10.3 192.168.11.3	Ubuntu 18.04	647
Open-AudIT	openAudITScanner	10.0.0.5	Ubuntu 18.04	759
Ordinary Host 1	ordinaryHost1	192.168.11.9	Ubuntu 18.04	636
Ordinary Host 2	ordinaryHost2	192.168.11.10	Ubuntu 16.04	622
Ordinary Host 3	ordinaryHost3	192.168.11.11	Ubuntu 14.04	803
Ordinary Host 4	ordinaryHost4	192.168.11.12	CentOS 7.6	458
Ordinary Host 6		192.168.11.15	Linux 3.2 - 4.8	6
Snort Scanner	snortScanner	10.0.0.201 192.168.10.201 192.168.11.201	Ubuntu 18.04	683

Table 5: Information retrieved about some devices using Nmap, Open-AudIT, Snort and the example CSV file

The proof of concept had the goal to find out whether it is feasible to generate a model of an IT infrastructure from multiple sources. As seen above, it is indeed feasible to generate a model of an IT infrastructure from multiple sources. Additionally, it is possible to extend the proof of concept with more sources as an input.

7 Conclusions

The proof of concept presented in this thesis shows that it is possible to automatically generate a model of an IT infrastructure. Section 5.1 shows that a lot of information is needed, from hardware information to operating system and applications to configurations and firewalls, to automatically generate a model of an IT infrastructure.

The first research question is: How can a model of an IT infrastructure be automatically created? This question is answered by the process that we proposed in this thesis. You need to have sources that collect information about the infrastructure. Then you need to collect that information, store it in a database and merge the data and generate the model from that merged data.

The second research question is: Are there sources available to collect enough data about an IT infrastructure. In this thesis, we found that there are lots of tools that can provide the requested data, such as the tools used in this thesis (Nmap, Open-AudIT and Snort). However, not all data can be retrieved using tools, such as the function of a device, why is it used? This information must be provided by humans.

The third research question is: Is the collected data accurate and reliable? As seen in section 3.3.5, when there is (almost) no access to a device, the data that is collected becomes less complete. Some tools have the ability to guess what software is running on a device, but this is less accurate than getting the information from the device itself. It all depends on the level of access that a tool has.

We found that it is best to use tools which have access to the devices in the network. The more access a tool has to its devices, the more information it can collect. Additionally, more access to a device also means that the collected information is more accurate. For example, as seen in section 6.2, Open-AudIT can have full access to the devices it scans, if provided with access credentials. This allows Open-AudIT to collect detailed information about the hardware of the device, as well as installed applications, the operating system, users, etcetera. In contrast, Nmap can only communicate with a device and infer information from its responses. Therefore, it has to guess the operating system and running applications, which is far less reliable. In fact, Nmap can only see applications that listen to a network port. Nmap does not detect applications that do not have a server of some sort.

The fourth research question is: How can the information be retrieved and combined? Retrieving information from a source is just a matter of consulting the source, e.g. querying a database with information, executing a tool and querying its result, etcetera. The challenge lies in storing the information in a format that is usable for merging and actually merging the data. For this purpose, the data has to be transformed to some generic form, which in turn can be merged. This merged data can then be used to generate a final model. There are lots of ways to do this merging process, as explained in section 3.5. In case of a source that has reliable information, a simple merging strategy, such as the 'Trust your friends' strategy can be sufficient, however, if there are only sources that are not completely reliable, this might work well enough.

In general, if the sources are reliable, the model will be an almost correct representation of the real world, within the limits of the level of detail. However, if the sources are not reliable, which happens when full access to the device is not available or possible, the model becomes a less correct representation and can only be an approximation as good as the most reliable source. For cyber security analysis, this means that full access to the devices is the best way to model the infrastructure at hand, in order to reason about the safety and weak points of the infrastructure.

If more or more reliable information is needed, the method needs to use data from one or more new sources. The method presented in this thesis does allow for integrating new sources, as the only requirement is to map the results of the new source to the generic form that is merged. If new information needs to be added to the model, the generic form needs to be extended as well as the adapters of all sources (if applicable). This requires more work as it also includes adding logic for merging this new information.

Another research question is whether the method is usable or sufficient for cyber security analysis. The proof of concept in this thesis merely shows that it is possible to generate a model of an IT infrastructure based on multiple sources. However, there is no usable security information, except OS and application information, in the final model at the moment. However, the fact the the model could be used to feed TNO's Cyber Security Decision Support tool shows that if more security related information would be available in the model, it could be usable. This must be verified by actually adding this information in future work.

Because the method uses multiple sources, another research question is whether it is feasible to collect or merge the information in such a way that it does not matter where the information comes from. In the prototype, this was solved by using an intermediate class that holds the requested data in a uniform format and also has merging logic. Therefore, it is possible to merge the information without the merging stage needing to know where the information came from or how it got there. The proof of concept also offers a way to add new source modules, which have to supply their data to the merging stages using the aforementioned intermediate class, allowing for extensibility, which does not affect the merging process as long as new kinds of information do not need to be included in the merging. The proof of concept is able to perform its duties, including importing data from sources, in a matter of minutes with a small scale network, as described in section 6.1.

However, the more devices there are, the longer the process takes and the more information a source can collect, the longer it takes to import and use the data. With a large network, the execution time of this process may explode. On the other hand, in the proof of concept, every step is (manually) executed sequentially. There is no reason why this could not be executed in parallel. It is possible to execute importing the data from different sources in parallel and then do the merging step, which should reduce the time.

The research questions also mention the ability of snapshotting. It is possible to snapshot the model such that it is possible to go back in time? In the proof of concept, this is not yet possible. However, the source data is snapshotted per scan, which makes it possible to re-generate a model from past data.

Using Object Role Modelling for formalising how the model should look like proves to be beneficial for actually creating the model. It makes it easy to transform what you know about the model into a database that enforces the facts that you inputted. It is also a useful tool for creating databases for the tools that have different forms of output per tool, such that there is one place to look for the raw data.

Explicit research question answers

This section explicitly describes why the research questions of this thesis are answered or why they are not answered as well as how they are answered.

The first research question is: How can a model of an IT infrastructure be automatically created? This question is answered, because this thesis proposes a method of how to accomplish automatically creating a model of an IT infrastructure. The method describes the process of selecting sources, collecting information about the infrastructure using these

sources, merging the data and creating a model from the data. The answer of this question is the proposed method of collecting, storing and merging the data about an IT infrastructure.

The second research question is: Are there sources available to collect enough data about an IT infrastructure. This question is answered, because it is shown that there are sources available to collect data about an IT infrastructure, but not all data that someone wants to have is available to collect by software, such as the reason a device is used. The answer to this questions is that there are sources available to collect data about an IT infrastructure, but there are not necessarily sources available to collect *enough* data.

The third research question is: Is the collected data accurate and reliable? This question is answered, because this thesis describes in section 3.3.5 that when there is (almost) no access to a device, the data that is collected becomes less complete. Some tools have the ability to guess what software is running on a device, but this is less accurate than getting the information from the device itself. It all depends on the level of access that a tool has. This can also be seen in the Findings (section 6.2) section where we see that when we add data from Open-AudIT, a tool with on-device access, to the merging process, we get more detailed and accurate information about the devices in the infrastructure.

The fourth research question is: How can the information be retrieved and combined? This question is answered by the description of merging strategies (section 3.5) and collection options (section 3.3). Furthermore, the thesis describes how two of the merging strategies were implemented in a proof of concept. There are many ways to retrieve data about an infrastructure with methods like active and passive scanning, running agents on devices, maintaining a management system with information about the devices, etcetera. The data from these different sources can be merged using many strategies, of which we chose 'Take the information' and 'Trust your friends'. These strategies combined take the information from the most trusted source and complete missing information with information from the less trusted sources if available.

The fifth research question is: Is this method usable or sufficient for automated security analysis of an IT infrastructure? This question is not directly answered by the thesis, except for being shortly mentioned in the conclusions section. However, if more information is added, this method could be usable for automated cyber security analysis. Therefore, at this moment, the method is not yet usable for cyber security analysis, but with more security related information, such as CVE/CPE information, information from vulnerability scanners, etcetera, the method could be usable for automated cyber security analysis. In fact, the final model can already be imported in TNO's Cyber Security Decision Support tool, which in it manual model has CVE/CPE information which is used for calculating an attack path. Therefore, adding CVE/CPE information could already be enough to demonstrate attack paths on an actual IT infrastructure.

The sixth research question is: Is it feasible to collect or merge the information in such a way that it does not matter where the information comes from? This question is not explicitly answered by this thesis. The thesis shows it is possible, but does not consider the execution time, which is relevant for this question. As the conclusions above mention, more information and devices slow the process down, but executing the importing step for each source in parallel reduces execution time. A lot harder, would be to also try to parallelise the merging process, e.g. find the entries that should be merged and then merge them in parallel.

The seventh research question is: Can we recreate a situation in the past based on data from back then? This question is answered by the fact that the method uses a separate storage step between collecting and merging the data, which can be used as well as storing

each newly generated model separately. In the proof of concept, only the former option is implemented. It is possible to recreate a situation in the past based on 'old' data. The advantage of using old data to recreate a model is that if more information is used in the merging process since that time, it is possible to use the extra information to generate a more complete model of the past than the model that could be generated back then.

The final research question is: Is there a way for the method to be transparent, such that a non-technical person could maintain the rules according to which the mapping takes place? This question is not answered by this thesis. The idea is that the rulesets, as seen in appendix A, are read by the merging stage and used to perform the merging step. The proof of concept does not implement this functionality and therefore, this research question is part of the future work section.

8 Future work

This method can be improved in different ways. For example, the merging process can be improved by actually using conflicting data to determine what the value should be. Additionally, machine learning or artificial intelligence could also be used to improve the merging process.

One of the research questions is whether the method is usable or sufficient for cyber security analysis. At the moment, this is not the case, because the method is currently limited to basic information about the infrastructure. However, it is also possible to infer properties based on the collected information, for example by using the list of installed applications to determine whether a virus scanner is installed and up to date.

Another way to add more information to the model is to use more sources to generate the model from. As the method is flexible in using more sources, adding new sources is a good possibility to extend the model. For security analysis, it can be helpful to add vulnerability scanners, active directory information, CPE and CVE data to enrich application data, etcetera, as new sources.

Using larger numbers of devices and more information per device, the method will slow down. To improve merging speeds, the merging process should take advantage of parallel computing where possible. For example, when the merging process knows which pieces of information belong to a single device, it can do the merging of these pieces in parallel to merging the information of another device.

This method could also function as a network virus scanner as a side effect. As having an agent on every device is the best way to collect extensive information about them, such an agent could also calculate hashes of files on the device and run them through a service such as VirusTotal[28].

The proof of concept model in this thesis models communications between application. However, because not all applications may use known ports or are detected as using a port, this approach may not be sufficient. An improvement could be to model a device communicating with a device. Alternatively, an agent could be used on device to determine which ports are used by which application during a passive scan.

Additionally, as mentioned before, the ruleset, in the form of the fact derivations, is not directly used by the proof of concept. Instead, the general idea is implemented in the code, but changing the ruleset has no effect without changing the code. To have real transparency, the ruleset should be used by the code while merging the data.

[28] `https://www.virustotal.com/`

A Fact derivations

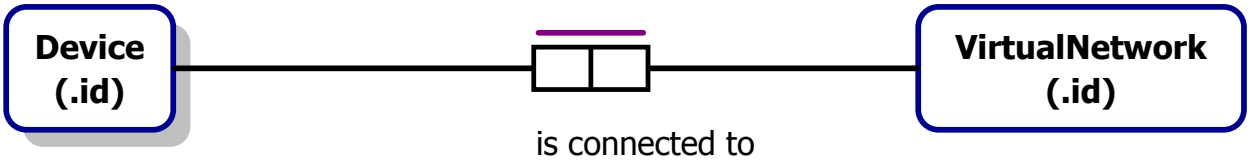
This appendix shows how all model facts are derived from the various sources. Please note that in cases where the Open-AudIT facts '`NetworkCard` has `IPvXAddress`' or '`NetworkCard` has `IPvXSubnet`' are mentioned, the '`X`' represents the IP version. This notation is used to avoid the 'same' fact being mentioned twice. Therefore, when these shortened versions are used, the following facts are meant:

- '`NetworkCard` has `IPv4Address`'
- '`NetworkCard` has `IPv6Address`'
- '`NetworkCard` has `IPv4Subnet`'
- '`NetworkCard` has `IPv6Subnet`'

In the fact derivations, there is an assumed order of trustworthiness for the sources. The rules are written based on this order. The following order of trustworthiness is the result of reviewing the sources. This order is the same in each fact derivation.

1. Open-AudIT
2. Nmap
3. Device Management Report
4. Snort

Fact C1: Device is connected to VirtualNetwork



Data Sources

- Nmap
- Open-Audit
- Snort
- Device Management Report

Rule

If Open-Audit detects a host, use fact '`Device` has `Address`' in combination with '`Network` has `Subnet`' or use fact '`NetworkCard` has `IPvXAddress`' in combination with '`NetworkCard` has `IPvXSubnet`'.
Else if Nmap detects a host, use the facts '`Host` has `Address`' and '`ScanConfig` scans `IPOrSubnet`'.
Else if the device has a record in the device management report, use the fact '`Device` has `IPAddress`'.
Else if Snort detects a host, use fact '`HostApplication` has `IPAddress`'.

Clarification

Nmap If an Nmap `Host` has an `Address`, it is connected to a network. Even though the network is not necessarily a virtual network, the fact can still be used. The notion of a virtual network is present due to the fact that most companies use VLANs in their network to separate hosts into distinct groups or departments.

Even though the (IP) address itself is not a network, it can be used to determine to which network a host is connected or at least determine which hosts are part of the same network. Using the information about which IP addresses and subnets were scanned, it is possible to find the corresponding subnet. In the case that only IP addresses were given as scan input, it is possible to guess a subnet by comparing the address to default private network subnets and other hosts that may have the same prefix, which is likely to be the subnet they belong to.

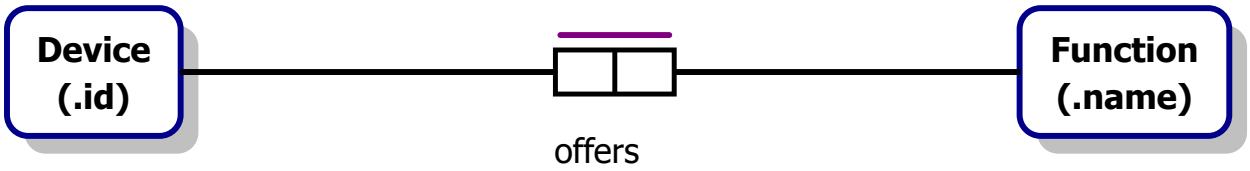
Open-AudIT Given an IP address and subnets of networks, it is possible to determine whether a device with a certain IP belongs to one of the networks (subnets).

In case Open-AudIT had access to a device and has information about its network card, it can use the IP address and subnet information from the network card acquire the relation between the device and a network.

Snort If Snort detects traffic, it records a source IP address and a target IP address. These IP addresses can be used to determine the existence of devices and to find out in which network they are. However, extra information about the subnets that are used within the network is needed to match an IP address to a network. Snort can also detect traffic to and from servers outside the network, which should be ignored as being part of the network within the model.

Device management report If there is a record for the device, there is an IP address. These IP addresses can be used to determine the existence of devices and to find out in which network they are. As mentioned above however, the IP address alone is not enough.

Fact C2: Device offers Function



Data Sources

- ~~Nmap~~
- Open-AudIT
- ~~Snort~~
- Device Management Report

Rule

If the function field of Open-AudIT is populated, use its fact '`Device` has `Function`'.
Else if there is a record for this device in the Device Management Report, use fact '`Device` has `Function`'.
Else if Open-AudIT was merely used for scanning purposes and the Device Manage Report does not have a record for this device, the data sources mentioned above do not offer information about the function of a device. As an alternative, the following sources can be used to acquire this information:

- Management System
- Manual input
- Interview

Manual process

The information about the function a device offers can be acquired in different ways:

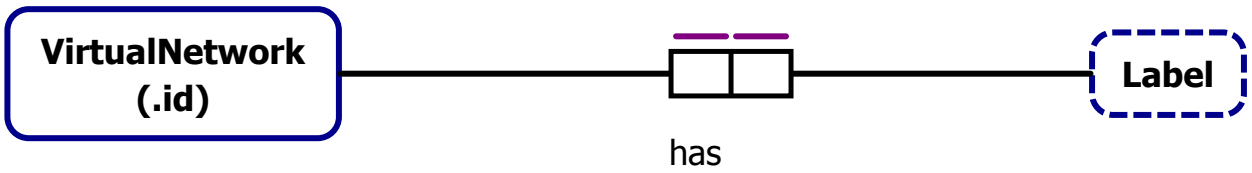
- If there is a management system that contains the requested information, an importer has to be written to extract the information from the management system. This option is similar to the Device Management Report.
- If the organisation maintains a list of functions that devices offer, it could be used to detect the function based on requirements that a device must comply with to have that function, given that those requirements are known and enough to determine the function.
- The list of devices could be given to a domain expert, for example the network manager, who is able to manually determine which function is offered by those devices.

Clarification

Nmap Nmap does not offer information about the function of a device, as that is not a technical property of a device or its software.

Snort Snort does not offer information about the function of a device, as that is not reliably detectable from only the network traffic.

Fact C3: VirtualNetwork has Label



Data Sources

- ~~Nmap~~
- Open-AudIT
- ~~Snort~~
- ~~Device Management Report~~

Rule

If Open-AudIT provides a name for the (virtual) network, use fact one.
Else The other data sources mentioned above do not offer information about the name of a virtual network. As an alternative, the following sources can be used to acquire this information:

- Management System
- Manual input
- Interview

Manual process

The information about the name of a virtual network can be acquired in different ways:

- If there is a management system that contains the requested information, an importer has to be written to extract the information from the management system.
- If there is a list of names for virtual networks, it could be used to detect the name based on the subnet.
- The list of virtual networks could be given to a domain expert, for example the network manager, who is able to manually determine the names of those networks.

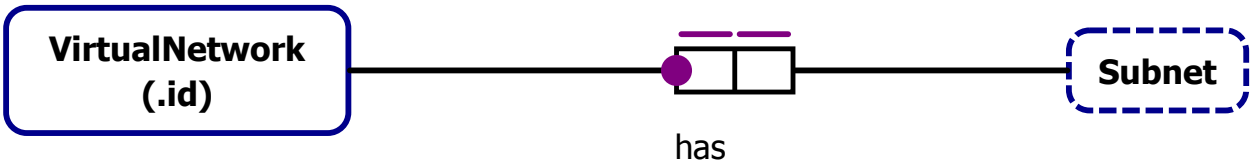
Clarification

Nmap Nmap does not offer information about the label or name of a network, as that is not a technical property of the network that can be detected.

Open-AudIT Open-AudIT does not offer information about the label or name of a network, as that is not a technical property of the network that can be detected. However, it is possible for the operator of the Open-AudIT instance to give different networks a name.

Snort Snort does not offer information about the label or name of a network, as that is not a technical property of the network that can be detected.

Fact C4: VirtualNetwork has Subnet



Data Sources

- Nmap
- Open-AudIT
- Snort
- ~~Device Management Report~~

Rule

All data source mentioned above, except the device management report, offer some form of subnet availability. However, there are a few limitations:

- Open-Audit does not have subnet information about a device if it does not have agent access to it. Therefore, if a device does not belong to any already known subnet, which can also be networks that Open-AudIT detected, the IP address can not be used to determine the subnet.
- For Nmap results, the requirement is that subnets were used as an input parameter. This means that if Nmap was instructed to scan one or more single IP addresses, it is not possible to know to which subnet they belong.

Procedure

Each known subnet is considered to be a separate (virtual) network. Subnets are acquired as follows. If Open-AudIT has information about the network card of a device, it also includes a subnet, which can be used to determine which networks exist. Additionally, Open-AudIT provides a list of networks that it knows of, which can be a good starting point in determining the available networks.

Nmap supports one or more subnets as input to scan those networks. These input subnets are also used to determine which networks exist.

Finally, Snort scan results include an interface address, which can be converted to a subnet.

All the collected subnets from above are used to determine which networks there are. Subnets that appear multiple times result in only one VirtualNetwork instance.

Clarification

A subnet is a description of an address space. Usually one network only hands out addresses from one address space, which is defined by the subnet. Therefore, the subnet is useful for determining which networks exist.

Fact C5: Each GatewayDevice is an instance of Device

Data Sources

- ~~Nmap~~
- ~~Open-AudIT~~
- ~~Snort~~
- ~~Device Management Report~~

Rule

None of the data sources mentioned above offers information about whether a device is a gateway device. As an alternative, the following sources can be used to acquire this information:

- Management System
- Manual input
- Interview

Manual process

The information about whether a device is a gateway device can be acquired in different ways:

- If there is a management system that contains the requested information, an importer has to be written to extract the information from the management system.
- If there is a list of gateway devices, it could be used to detect the name based on the IP address or hostname.
- The list of devices could be given to a domain expert, for example the network manager, who is able to manually determine which devices are gateway devices.

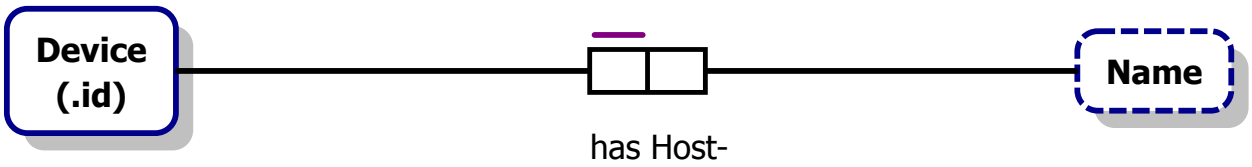
Clarification

Nmap Nmap can offer information about whether a device is a gateway device, but that will then be based on the detected operating system, for example, is it an OS that is used by routers? As detecting operating systems is not a certain process, Nmap is not a good source for deriving this fact.

Open-AudIT Open-AudIT does not offer information about whether a device is a gateway device, because it is not part of the info that is collected about the network card of devices nor the network list that Open-AudIT offers.

Snort Snort does not offer information about whether a device is a gateway device, because traffic that goes to or comes from a gateway device does not necessarily have the gateway device as source or target. IP packets have the IP address of their original source or target in them and not the IP addresses of intermediate devices.

Fact D1: Device has Host- Name



Data Sources

- Nmap
- Open-AudIT
- ~~Snort~~
- Device Management Report

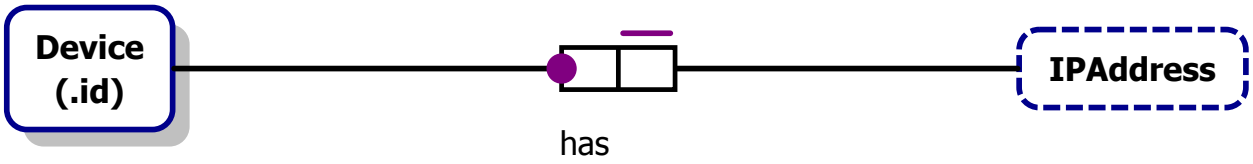
Rule

If Open-AudIT detects a host, use fact '`Device` has Host- `Name`'.
Else if A host is mentioned in the Device Management Report, use fact '`Device` has `Hostname`'.
Else if Nmap detects a host, use fact '`Host` has `Hostname`'.

Clarification

All the sources, except Snort, are able to detect a hostname of a device, so to derive this fact, the hostname can be taken from the first source that has detected a hostname.

Fact D2: Device has IPAddress



Data Sources

- Nmap
- Open-AudIT
- Snort
- Device Management Report

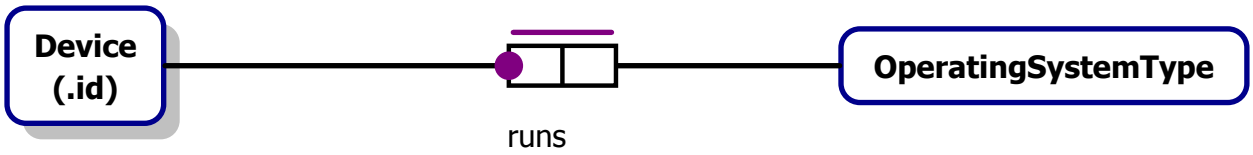
Rule

If Open-AudIT detects a host, use fact '`Device` has `IPAddress`' or '`NetworkCard` has `IPvXAddress`'.
Else if A host is mentioned in the Device Management Report, use fact '`Device` has `IPAddress`'.
Else if Nmap detects a host, use fact '`Host` has `IPAddress`'.
Else if Snort detects a host, use fact '`HostApplication` has `IPAddress`'.

Clarification

All the sources detect devices with at least an IP address. Therefore, each source identifies detected devices with an IPAddress, which can be used to derive this fact.

Fact D3: Device runs OperatingSystemType



Data Sources

- Nmap
- Open-AudIT
- ~~Snort~~
- ~~Device Management Report~~

Rule

If Open-AudIT detects a host, use fact '`Device` has `OperatingSystem`' and its subfacts.
Else if Nmap detects a host, use fact '`Host` has `OperatingSystemGuess`' and its subfacts.

Clarification

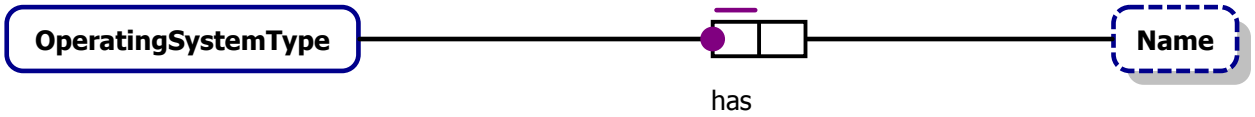
Open-AudIT If a `Device` has an `OperatingSystem`, it can be used to determine which Operating System it uses. As the fields of the operating system are only filled if Open-AudIT had access to the device, the information is correct.

Nmap If a `Host` has an `OperatingSystemGuess`, it can be used to determine which Operating System it possibly uses. However, as it is merely a guess, it is not reliable information as it might be incorrect.

Snort Snort does not offer information about the operating system of the hosts that generate the sniffed traffic. It could be possible to analyse the IP packets and try to find OS specific characteristics, but Snort does not offer that functionality out of the box.

Device Management Report The Device Management Report example from appendix C.4 does not offer information about operating systems.

Fact D4: OperatingSystemType has Name



Data Sources

- Nmap
- Open-Audit
- ~~Snort~~
- ~~Device Management Report~~

Rule

If Open-AudIT has detected an `OperatingSystem`, use the fact `OperatingSystem` has `Name`.
Else if Nmap has detected an `OperatingSystemGuess`, use the fact `OperatingSystemGuess` has `Description`.

Clarification

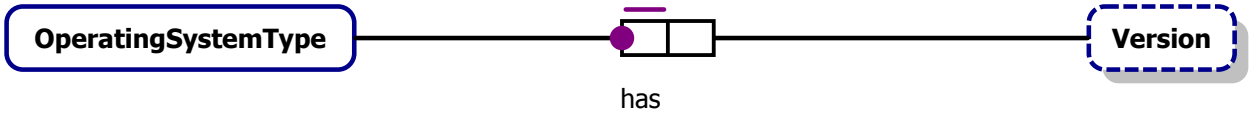
Open-AudIT Open-AudIT has information about each operating system it detects on a device or none if it did not have access. Therefore, once a connection is found between a device and an `OperatingSystem` from Open-AudIT, the information about the `OperatingSystem` can be used to fill in the name and version information about the `OperatingSystem Type`.

Nmap Nmap has information about each operating system it (thinks it) detects on a host. Therefore, once a connection is found between a host and an `OperatingSystemGuess` from Nmap, the information about the `OperatingSystemGuess` can be used to fill in the name and version information about the `OperatingSystemType`.

Snort Snort does not offer information about the operating system of the hosts that generate the sniffed traffic. It could be possible to analyse the IP packets and try to find OS specific characteristics, but Snort does not offer that functionality out of the box.

Device Management Report The Device Management Report example from appendix C.4 does not offer information about operating systems.

Fact D5: OperatingSystemType has Version



Data Sources

- Nmap
- Open-Audit
- ~~Snort~~
- ~~Device Management Report~~

Rule

If Open-AudIT has detected an `OperatingSystem`, use the fact `OperatingSystem` has `Version`.
Else if Nmap has detected an `OperatingSystemGuess`, use the fact `OperatingSystem-Class` has `Version`. The corresponding `OperatingSystemClass` can be found using the fact `OperatingSystemGuess` has `OperatingSystemClass`.

Clarification

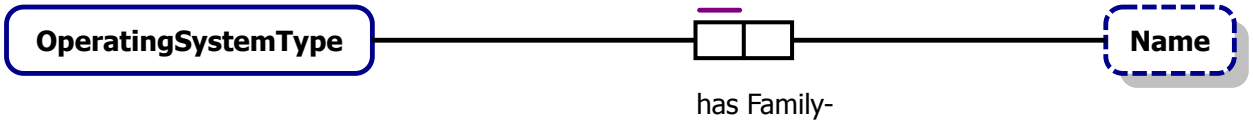
Open-AudIT Open-AudIT has information about each operating system it detects on a device or none if it did not have access. Therefore, once a connection is found between a device and an `OperatingSystem` from Open-AudIT, the information about the `Operating-System` can be used to fill in the name and version information about the `OperatingSystem Type`.

Nmap Nmap has information about each operating system it (thinks it) detects on a host. Therefore, once a connection is found between a host and an `OperatingSystemGuess` from Nmap, the information about the `OperatingSystemGuess` can be used to fill in the name and version information about the `OperatingSystemType`.

Snort Snort does not offer information about the operating system of the hosts that generate the sniffed traffic. It could be possible to analyse the IP packets and try to find OS specific characteristics, but Snort does not offer that functionality out of the box.

Device Management Report The Device Management Report example from appendix C.4 does not offer information about operating systems.

Fact D6: OperatingSystemType has Family- Name



Data Sources

- Nmap
- Open-Audit
- ~~Snort~~
- ~~Device Management Report~~

Rule

If Open-AudIT has detected an `OperatingSystem`, use the fact `OperatingSystem` has `Family`.
Else if Nmap has detected an `OperatingSystemType`, use the fact `OperatingSystem-Class` has `Family`. The corresponding `OperatingSystemClass` can be found using the fact `OperatingSystemGuess` has `OperatingSystemClass`.

Clarification

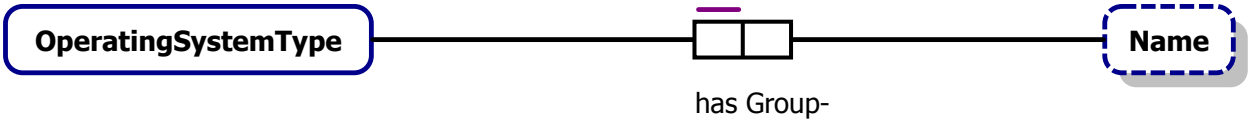
Open-AudIT Open-AudIT has information about each operating system it detects on a device or none if it did not have access. Therefore, once a connection is found between a device and an `OperatingSystem` from Open-AudIT, the information about the `Operating-System` can be used to fill in the family information about the `OperatingSystemType`.

Nmap Nmap has information about each operating system it (thinks it) detects on a host. Therefore, once a connection is found between a host and an `OperatingSystemGuess` from Nmap, the information about the `OperatingSystemGuess` can be used to fill in the family information about the `OperatingSystemType`.

Snort Snort does not offer information about the operating system of the hosts that generate the sniffed traffic. It could be possible to analyse the IP packets and try to find OS specific characteristics, but Snort does not offer that functionality out of the box.

Device Management Report The Device Management Report example from appendix C.4 does not offer information about operating systems.

Fact D7: OperatingSystemType has Group- Name



Data Sources

- Nmap
- Open-Audit
- ~~Snort~~
- ~~Device Management Report~~

Rule

If Open-AudIT has detected an `OperatingSystem`, use fact `OperatingSystem` has `Group`.
Else if Nmap has detected an `OperatingSystemType`, use the fact `OperatingSystemClass` has `Description`. The corresponding `OperatingSystemClass` can be found using the fact `OperatingSystemGuess` has `OperatingSystemClass`.

Clarification

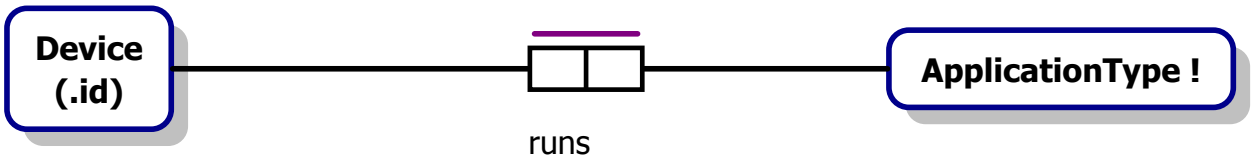
Open-AudIT Open-AudIT has information about each operating system it detects on a device or none if it did not have access. Therefore, once a connection is found between a device and an `OperatingSystem` from Open-AudIT, the information about the `OperatingSystem` can be used to fill in the family information about the `OperatingSystemType`.

Nmap Nmap has information about each operating system it (thinks it) detects on a host. Therefore, once a connection is found between a host and an `OperatingSystemGuess` from Nmap, the information about the `OperatingSystemGuess` can be used to fill in the family information about the `OperatingSystemType`.

Snort Snort does not offer information about the operating system of the hosts that generate the sniffed traffic. It could be possible to analyse the IP packets and try to find OS specific characteristics, but Snort does not offer that functionality out of the box.

Device Management Report The Device Management Report example from appendix C.4 does not offer information about operating systems.

Fact D8: Device runs ApplicationType



Data Sources

- Nmap
- Open-Audit
- ~~Snort~~
- ~~Device Management Report~~

Rule

If Open-AudIT detects a device, use facts '`Device` has `Application`', '`Device` has `Service`', '`Device` has `ODBCDriver`', '`Device` has `SoftwareUpdate`' and their subfacts.
Else if Nmap detects a host, use fact '`Host` has `OpenPort`' and its subfacts.

Clarification

Open-AudIT If a `Device` has an `Application`, `ODBCDriver`, a `Service` or `SoftwareUpdate`, it can be used to determine which applications are running on the device. As this information is only available when Open-AudIT has access to the device, this information is correct.

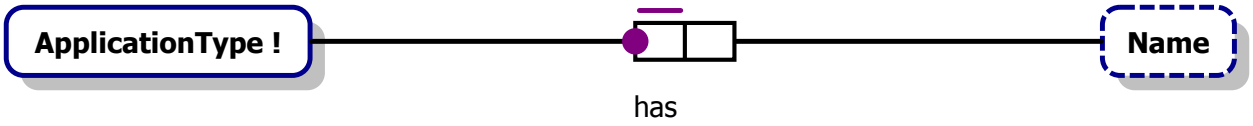
Nmap If a `Host` has an `OpenPort`, it can be used to determine which applications are running on the host and expose themselves over the network. However, this approach may not be reliable as it depends on what a listening application responds to Nmap.

Snort Snort does not offer information about applications running on the hosts that generate the sniffed traffic other than the port number the traffic originates from. It is possible to guess the application using a list of known ports that are used by certain applications. However, this is not a reliable way of finding which software is running on the host, because everyone can use almost every port for their own purposes.

Additionally, a lot of traffic goes to or comes from random ports, such as a web browser visiting a web page. This means that reliably detecting running software on hosts is not possible with Snort.

Device Management Report The Device Management Report example from appendix C.4 does not offer information about applications.

Fact D9: ApplicationType has Name



Data Sources

- Nmap
- Open-Audit
- ~~Snort~~
- ~~Device Management Report~~

Rule

If Open-AudIT has detected an `Application`, `ODBCDriver`, `Service` or `SoftwareUpdate`, use the fact '`Application/ODBCDriver/Service/SoftwareUpdate` has `Name`'.
Else if Nmap has detected an `OpenPort`, use the fact `OpenPort` has Product- `Name`

Clarification

Open-AudIT Open-AudIT has information about each application, ODBC driver, service and software update it detects on a device or none if it did not have access. Therefore, once a connection is found between a device and one of these software types from Open-AudIT, the information about the software type can be used to fill in the name and version information about the `ApplicationType`.

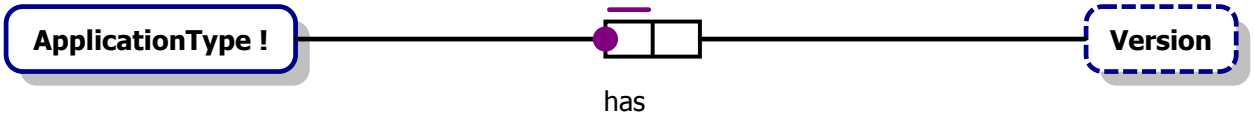
Nmap Nmap has information about each application or service it (thinks it) finds on an open port. Therefore, once a connection is found between a host and an `OpenPort` from Nmap, the information about the `OpenPort` can be used to fill in the name and version information about the `ApplicationType`.

Snort Snort does not offer information about applications running on the hosts that generate the sniffed traffic other than the port number the traffic originates from. It is possible to guess the application using a list of known ports that are used by certain applications. However, this is not a reliable way of finding which software is running on the host, because everyone can use almost every port for their own purposes.

Additionally, a lot of traffic goes to or comes from random ports, such as a web browser visiting a web page. This means that reliably detecting running software on hosts is not possible with Snort.

Device Management Report The Device Management Report example from appendix C.4 does not offer information about applications.

Fact D10: ApplicationType has Version



Data Sources

- Nmap
- Open-Audit
- ~~Snort~~
- ~~Device Management Report~~

Rule

If Open-AudIT has detected an `Application`, `ODBCDriver`, `Service` or `SoftwareUpdate`, use the fact '`Application/ODBCDriver/Service/SoftwareUpdate` has `Version`'.
Else if Nmap has detected an `OpenPort`, use the fact '`OpenPort` has `Version`'.

Clarification

Open-AudIT Open-AudIT has information about each application, ODBC driver, service and software update it detects on a device or none if it did not have access. Therefore, once a connection is found between a device and one of these software types from Open-AudIT, the information about the software type can be used to fill in the name and version information about the `ApplicationType`.

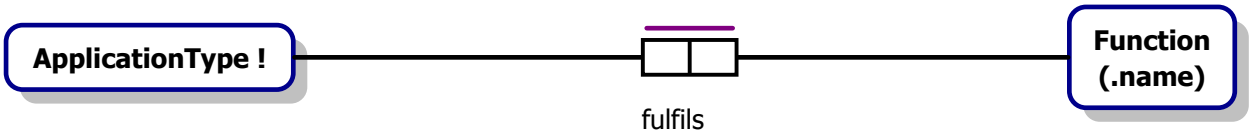
Nmap Nmap has information about each application or service it (thinks it) finds on an open port. Therefore, once a connection is found between a host and an `OpenPort` from Nmap, the information about the `OpenPort` can be used to fill in the name and version information about the `ApplicationType`.

Snort Snort does not offer information about applications running on the hosts that generate the sniffed traffic other than the port number the traffic originates from. It is possible to guess the application using a list of known ports that are used by certain applications. However, this is not a reliable way of finding which software is running on the host, because everyone can use almost every port for their own purposes.

Additionally, a lot of traffic goes to or comes from random ports, such as a web browser visiting a web page. This means that reliably detecting running software on hosts is not possible with Snort.

Device Management Report The Device Management Report example from appendix C.4 does not offer information about applications.

Fact D11: ApplicationType fulfils Function



Data Sources

- ~~Nmap~~
- ~~Open-Audit~~
- ~~Snort~~
- ~~Device Management Report~~

Rule

The data sources mentioned above do not offer information about the function of an application type. As an alternative, the following sources can be used to acquire this information:

- Management System
- Manual input
- Interview

Manual process

The information about the function an application type offers can be acquired in different ways:

- If there is a management system that contains the requested information, an importer has to be written to extract the information from the management system.
- The list of application types could be given to a domain expert, for example the employee who manages which applications are allowed/installed, who is able to manually determine which function is offered by those application types.

Clarification

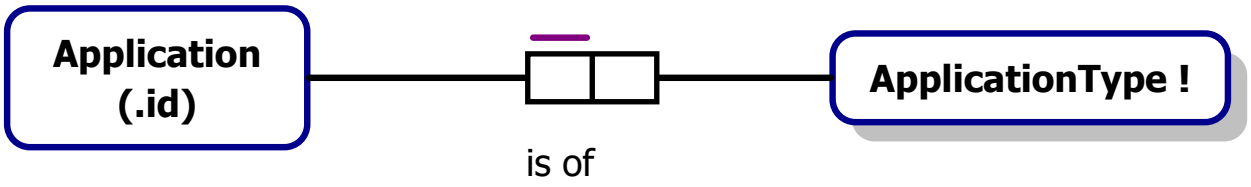
Open-AudIT Open-AudIT does not offer information about the function of an application, as that is not a technical property of software.

Nmap Nmap does not offer information about the function of an application, as that is not a technical property of software.

Snort Snort does not offer information about the function of an application, as that is not reliably detectable from only the network traffic.

Device Management Report The Device Management Report example from appendix C.4 does not offer information about applications.

Fact I1: Application is of ApplicationType



Data Sources

- Nmap
- Open-Audit
- Snort
- ~~Device Management Report~~

Rule

If Snort has detected a communicating application, use facts `HostApplication` has `IP-Address` and `HostApplication` has `Port` to match the application to an application type using information from Nmap `OpenPort` instances. A port could also be empty, which makes it impossible to match an application.

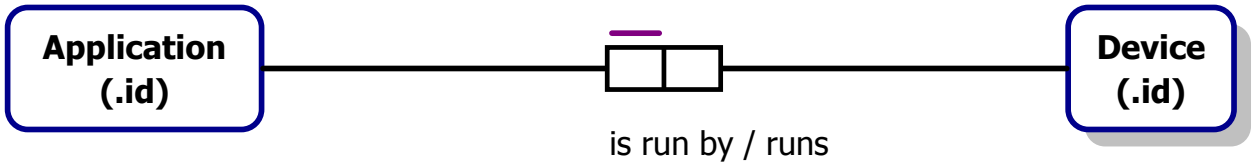
Clarification

Snort Snort does not offer information about applications running on the hosts that generate the sniffed traffic other than the port number the traffic originates from. It is possible to guess the application using a list of known ports that are used by certain applications. However, this is not a reliable way of finding which software is running on the host, because everyone can use almost every port for their own purposes.

Open-AudIT and Nmap In terms of whether the traffic originates due to an application type that was detected using Nmap or Open-AudIT, however, is possible, because the port number from the traffic can be matched to the port number of an Nmap `OpenPort` or to an application detected using Open-AudIT, which may need a list of known ports used by this application.

Limitations Because not all applications may use known ports or are detected as using a port, this approach may not be sufficient. Alternatively, the approach could be to model a device communicating with a device.

Fact I2: Application is run by Device



Data Sources

- Nmap
- Open-Audit
- Snort
- Device Management Report

Rule

If Snort has detected a communicating application, use the fact `HostApplication` has `IP-Address` to find the device this application runs on. All other sources provide a list of hosts or devices which can be matched based on the found IP address. This can be done using the Open-Audit facts '`Device` has `IPAddress`' or '`NetworkCard` has `IPvXAddress`', Nmap fact '`Host` has `IPAddress`' and Device Management Report fact '`Device` has `IPAddress`'.

Clarification

Snort Given the IP address that sends or receives the traffic, it is possible to find the device that is responsible for this traffic by finding the device that has the same IP address as the source or target `HostApplication`.

Fact I3: Application communicates with Application

Data Sources

- ~~Nmap~~
- ~~Open-Audit~~
- Snort
- ~~Device Management Report~~

Rule

If Snort has detected a communicating application, use the the fact '`HostApplication` communicates with `HostApplication`' to find the applications communicating with each other.

Clarification

Snort Given the an entry in the communications that Snort intercepted, it is possible to derive this fact with the help of IP address and port information, which is already used by Facts I1 and I2.

B Tested Sources

This appendix describes the considerations that led to selecting the sources that were used for the proof of concept presented by this thesis.

Using the test network described in section 6.1, the following sources were tested: Nmap, Snort, Observium, Open-AudIT and Nagios XI. The sources that were selected are Nmap, Open-AudIT and Snort.

- **Nmap**
  Nmap is an active scanner that, when given an IP address or range, pings the devices to find them. It can also find out what services and operating system are running on discovered devices. However, this information is only limited to the open ports, the information that the running services behind the open ports provide, publicly available information about services that tend to use certain ports and information about how different operating systems respond to specific network packets.

  Advantages Nmap is useful for finding new devices within a given IP range or subnet as well as checking whether a known device is still online. Nmap offers operating system and running services detection based on responses from the host as well in addition to XML output, which can be parsed by machines.

  Disadvantages The Nmap operating system and service detection is not accurate. As Nmap does not have access to devices itself, but can only deduce information from communication responses, it can not reliably detect the correct information. Nmap is also not able to detect relations between devices. It can only detect whether a device is online or not and communicate with it, but Nmap does not scan network traffic between other devices.

  Using Nmap is useful for detecting devices and acquiring basic information about a host, but when more accurate information is needed, Nmap is not sufficient.

- **Snort**
  Snort is a passive scanner and intrusion detection system. As an intrusion detection system, it is an important tool for maintaining network security. For the collection information for a model of the network, Snort is useful as it shows communication between devices in the network. Depending on the network traffic, Snort can also detect what kind of service is used.

  Advantages Snort can be used to identify relations between devices as the output of Snort shows source and destination IP addresses and ports. In addition, these ports can be used to guess which services are running on the corresponding devices. Snort offers multiple forms of output, but the default console output can be filtered using the `grep` command line utility. The result can be parsed by a custom parser.

  Disadvantages Snort shows almost no information about the devices it monitors traffic from. This is unavoidable as Snort is a passive scanner and therefore Snort can only collect information that is within the network traffic. Being a passive scanner also

limits the locations where Snort must listen, it must be a location where all or at least most of the network traffic must travel.

Using Snort is useful for detecting relations between devices, but it must be located at a point where all network traffic comes through.

- **Observium**
  Observium is a monitoring tool that uses the SNMP protocol to monitor devices in the network. Observium does not automatically scan the network; the user has to add each device themselves. This process can be automated using a REST API or via command line access using scripts.

  The information collected by Observium is primarily about the hardware of the monitored devices. Observium collects information about CPU usage, memory usage, disk usage, etcetera, as well as the properties of the CPU, memory and hard drives. Additionally, the operating system is also detected.

  Advantages Observium is able to detect the operating system of monitored hosts. It also provides a REST API for convenient access to its information. It is also possible to manipulate Observium via direct command line access.

  Disadvantages The main disadvantage is that an SMTP agent is needed on each device that is to be monitored by Observium. There is no support for agent-less devices, which makes Observium insufficient for monitoring all devices in the network.

  Even though detecting the operating system is an advantage, it is also a disadvantage, because operating system detection relies on what the SNMP agent reports. Unfortunately, the SNMP agent may report generic information that is not enough to accurately identify the operating system. Given that an agent is required, Observium does not add much value compared to Nmap in such cases. However, there are also cases where the operating system is identified more accurately than Nmap.

  As an active scanner, like Nmap, Observium does not detect relations between devices.

  Observium is useful for monitoring devices and keeping an eye on their CPU, memory and disk usage. However, for building a model, there is not enough information to gather to make it an interesting tool for building a model. As Observium requires an agent to work, it is more interesting to use a tool that collects much more information by using an agent.

- **Open-AudIT**
  Open-AudIT is an active scanner that is built to tell the user what is on their network, how it is configured and when it changes [43]. Adding devices is done by starting a discovery with an IP range, subnet or a single IP address. Open-AudIT will scan the given range, subnet or address and add all devices it may find. Using SSH access and audit scripts, Open-AudIT collects information from each device it finds.

Advantages Open-AudIT can detect new devices within a given IP range or subnet. In fact, Open-AudIT uses Nmap for the discovery of devices and then proceeds with its own information collecting steps.

Open-AudIT is able to accurately detect the operating system and version of scanned devices, as it can just collect that information using SSH and its audit scripts.

Due to the SSH access and audit scripts, Open-AudIT is able to detect all installed software[29] and services as well as what kind of servers are running on the host.

As an alternative to SSH access, it is also possible to have the network devices themselves run the audit script and report its results to Open-AudIT. Using this setup, the devices actively report themselves to Open-AudIT instead of Open-AudIT requesting information from the devices.

All this information is accessible by using Open-AudIT's REST/JSON API.

Disadvantages Just like Observium, Open-AudIT depends on some form of access to the scanned devices. In this case, Open-AudIT needs SSH access or audit scripts. As a result, valid authentication credentials must be present in the Open-AudIT database for the discovery process to try when setting up an SSH connection. Furthermore, depending on the operating system, there must be an audit script present on each scanned device.

Open-AudIT still works when SSH access and audit scripts are not available, but in that case, the information that is collected by Open-AudIT is limited and often inaccurate.

The final limitation is that Open-AudIT is not able to detect relations between devices, as it only looks at hosts and not at network traffic.

Open-AudIT is a very useful tool that provides a lot of information that can be used to build a model of a network, but it requires extensive access to the devices that are to be part of the model.

- **Nagios XI**
  Nagios XI is a monitoring tool. It is possible to add devices to the system by providing an IP range or subnet. Once given, Nagios discovers any live device and the user can add them to be monitored.

  If a Nagios agent is installed, it can detect which services are running and monitor them.

  Advantages If a separate tool (Nagios Network Analyzer) is installed, Nagios XI can detect relations between devices. However, for that purpose, one could also just install Nagios Network Analyzer without Nagios XI.

  Nagios XI also has a JSON API, which can be used to control the system.

[29] Given that is is installed in default locations or using a package manager.

Disadvantages Nagios XI does not show operating system information about hosts. Only when devices are discovered and presented to the user to be added to the system, the operating system is mentioned. However, this information is based on Nmap and therefore does not increase reliability. However, the information about the operating system is nowhere to be found after the devices are added to the system.

Even though Nagios XI shows services, it does not show what services are running. It makes a guess based on the port number, but does not identify the services, other than reporting a response, which might, but does not necessarily, reveal the software running behind the port. To really identify the services running, an agent is needed on the hosts themselves.

Nagios XI is useful for monitoring devices and keeping an eye on their service. However, for building a model, there is not enough information to gather to make it an interesting tool for building a model.

C Data Models

The tools mentioned above each have their own way of presenting the collected data. For example, Nmap can output to a normal text file or an XML file, Open-AudIT provides a REST API to access its data and Snort provides textual output when using it as a sniffer.

To create a single model of these three sources, there must be some uniform way of accessing the collected data. This is done by creating a database that holds all the data. This section describes the data models that describe how the collected data is stored.

C.1 Nmap Scan Result

The result of an Nmap scan is stored in the database as an Nmap scan. The Nmap scan has a scan config and a scan result, as well as an Nmap version. Figure 20 shows the data model of how the Nmap scan is stored with its configuration and result. A relational view of the resulting database can be seen in appendix D.

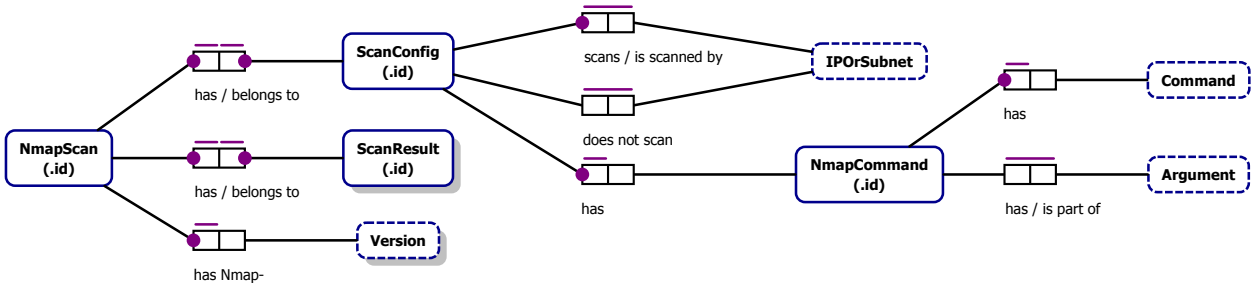


Figure 20: Data model of Nmap scan

It starts with an `NmapScan` class. This class has a scan config (`ScanConfig`), a scan result (`ScanResult`) and an Nmap version (`Version`). The Nmap version is recorded, because a new version could bring changes that can impact the scan result significantly. Knowing the version can impact decisions when dealing with the result and detecting changes compared to previous models or results.

The `ScanConfig` class stores information about the scan. It keeps records of which IP addresses and/or subnets were scanned as well as which IP addresses and/or subnets were excluded from the scan. Additionally, it records a list of arguments, which were given to the Nmap application as well as the entire Nmap command string.

The `ScanResult` stores information about the retrieved information. Each Nmap Scan result is stored in the database and is represented by the `ScanResult` class. A scan result has one, mandatory, timestamp and at least one or more hosts, as seen in figure 21.

These hosts are represented by the `Host` class. This host object has several properties. It has an address, network distance and uptime, which are all mandatory and each host has exactly one address, distance and uptime. It is however possible that multiple hosts have the same address as they can be part of the same scan result. It is assumed that Nmap will not list conflicting addresses within the same scan result.

Furthermore, each host has at least one or more Operating System Guesses. Each Operating System Guess, represented by the `OperatingSystemGuess` class, has a description and a confidence percentage. This percentage indicates how sure Nmap is that a host runs the given operating system. It is possible that multiple hosts have the same `OperatingSystemGuess`. The reasoning behind that possibility is that there is not much variation between the guesses

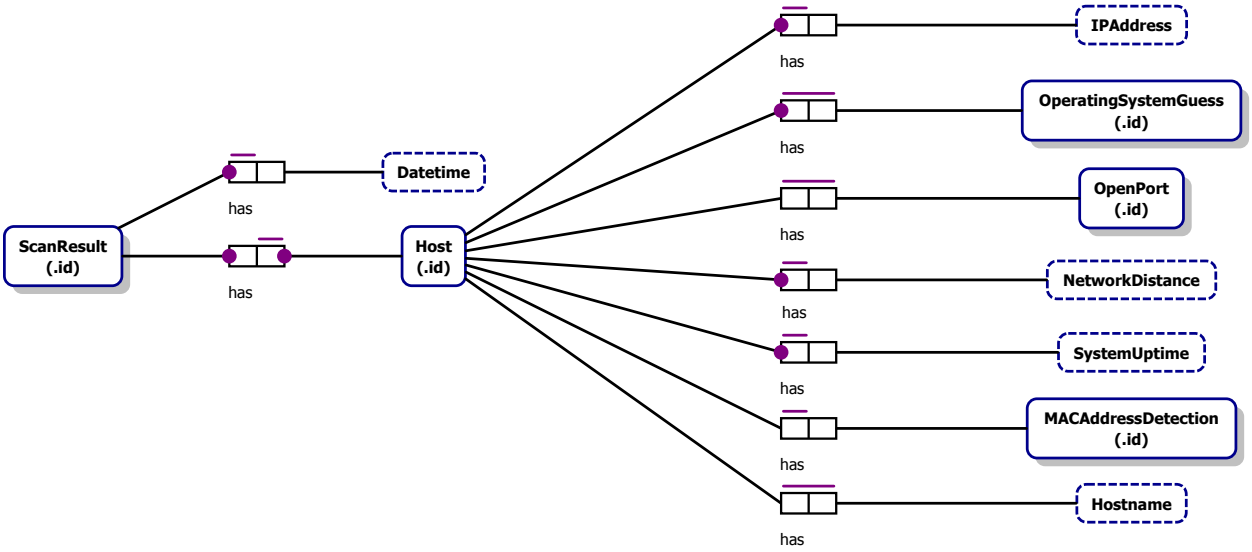


Figure 21: Data model of Nmap scan result

and it saves space to only store an operating system description and confidence percentage pair once.

As seen in figure 22, each operating system guess has one ore more `OperatingSystemClass`'es. These contain more specific information about the operating system, such as the OS family, vendor and version. They also provide the type of the operating system as well as the accuracy, which is often the same as the accuracy of the parent operating system guess.

A host can also have open ports, represented by the `OpenPort` class. As it is possible that there are no open ports on a host, the presence of one or more open ports is not required by the data model. As with the Operating System Guesses, multiple hosts can have the same `OpenPort`. An open port has a port number, a service, a product, a version and extra info. The service field indicates the protocol that is usually associated with the port number. The product, version and extra info fields contain the product and version information as well as the extra info reported by the listening application. All these fields, except the port number, are not mandatory as not all applications necessarily provide information for these fields.

If the machine running Nmap is within the same subnets of the devices it discovers, it also finds the MAC addresses of the devices. As it does not necessarily find these MAC addresses, a `Host` is not required to have one. However, it can have at most 1 `MACAddressDetection`. In contrast, it is possible that multiple hosts have the same `MACAddressDetection`. This may seem unintuitive, but with each scan, hosts may be scanned once more. This results in another host, coupled to a different scan result, that has the same MAC address. Additionally, each `MACAddressDetection` has a vendor, as seen in figure 22, determined by Nmap.

To save storage space, each `OpenPort`, `MACAddressDetection`, `OperatingSystemGuess` and `OperatingSystemClass` must be unique. If one of these with the same information was already stored in the database while processing a previous scan, the old record can be reused. The connection between a `Host` and for example an `OpenPort` is stored using a combination of keys (indexes), which allows easy reusing of old records.

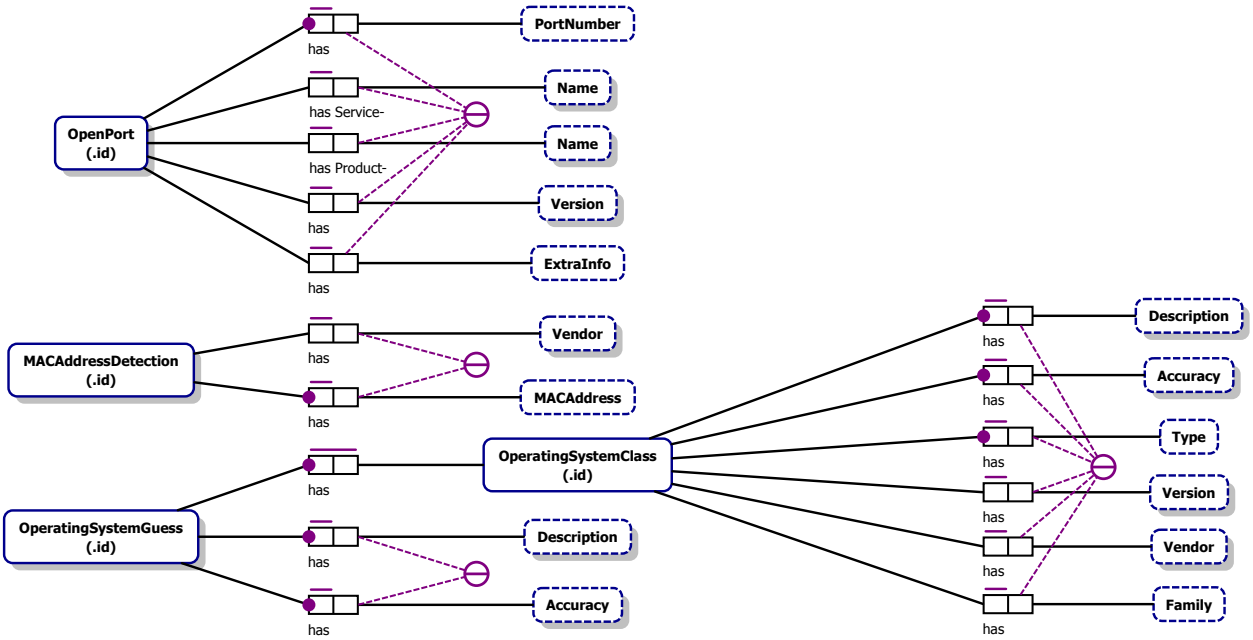


Figure 22: Data model of Nmap open port, MAC address and operating system guess

C.2 Open-AudIT Scan Result

The result of an Open-AudIT scan is stored as an `OAScanResult`, as seen in figure 23. A scan result has exactly one date and time which is the moment the scan result was retrieved from the REST API, which allows access to data from the Open-AudIT database. In addition, a scan result has a list of devices and a list of networks (see figure 27).

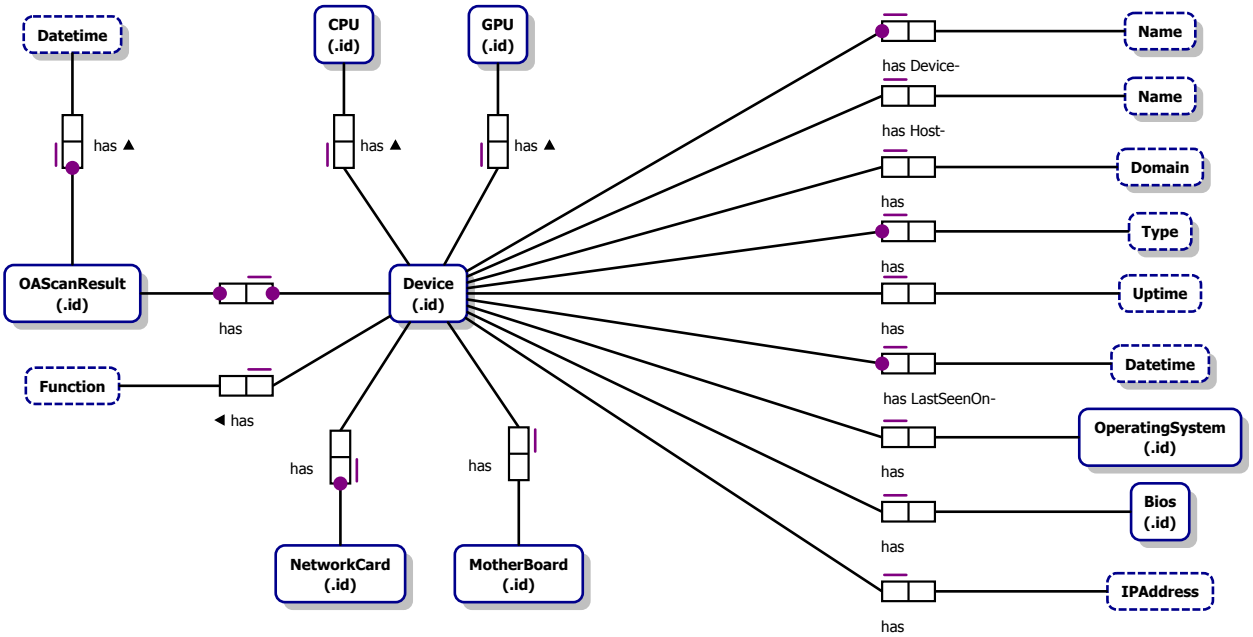


Figure 23: Data model of Open-AudIT scan result

Each device has a lot of properties, which are listed below. Not all properties are described in detail in this section. Some properties are in the data model, because they can be useful for the purpose of using the model for cyber security analysis. However, they are not used in the final model and described separately in appendix E.

- **DeviceName**
  Name of the device. Can be customised by the operator of the Open-AudIT instance.
- **HostName**
  Hostname of the device.
- **Domain**
  Network domain to which the device belongs. Relevant for Windows devices.
- **Type**
  What type of device this is. Type can be things like Computer, Firewall, Router, etcetera.
- **Uptime**
  Shows how long the device is already running.
- **LastSeenOnDatetime**
  The moment that the device was last scanned or queried by Open-AudIT.
- **IPAddress**
  The IP address of the device.
- **Function**
  The function of the device. Must be manually inputted by the operator of the Open-AudIT instance.
- **OperatingSystem**
  Information about the operating system that runs on the device. See figure 24.
- **NetworkCard**
  Information about the network cards of the device. See figure 25.
- **Bios**
  Information about the BIOS of the device. See figure 31 in appendix E.
- **CPU**
  Information about the CPU of the device. See figure 32 in appendix E.
- **GPU**
  Information about the GPU of the device. See figure 32 in appendix E.
- **MotherBoard**
  Information about the mother board of the device. See figure 33 in appendix E.

For all these properties, it is always the case that each device can have at most one value for this property, or if a property is required, exactly one. The only exception is `NetworkCard`, for which only one Device can have that network card, though a device can have more than one network card.

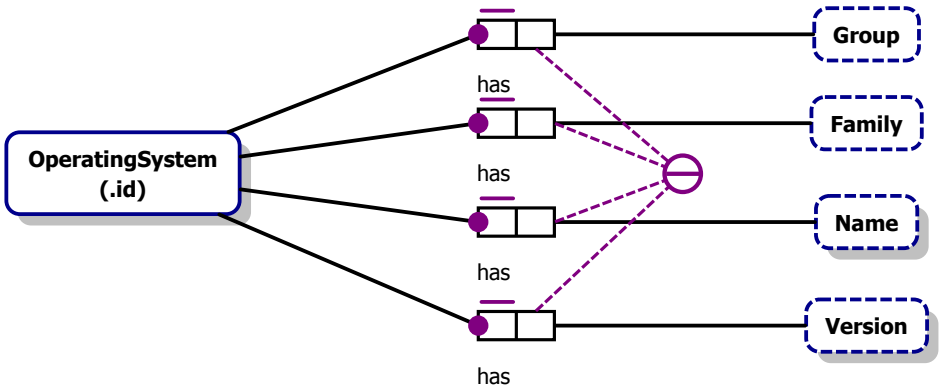


Figure 24: Data model of Open-AudIT operating system

Open-AudIT detects the name, version, operating system group and operating system family of a device. Figure 24 shows how this information is represented in the data model. This information is only available if Open-AudIT had access to the device itself. If the information is found, all fields of the Operating System must be filled. Additionally, if an operating system is already present in the database, the existing record is reused. If Open-AudIT did not have access to the device, Open-AudIT is not able to determine the operating system information.

If Open-AudIT has access to a device, it can also collect information about the network cards of the device, which is for example useful to detect whether the device is connected to multiple networks. Furthermore, the network card also provides extra information about each connected network.

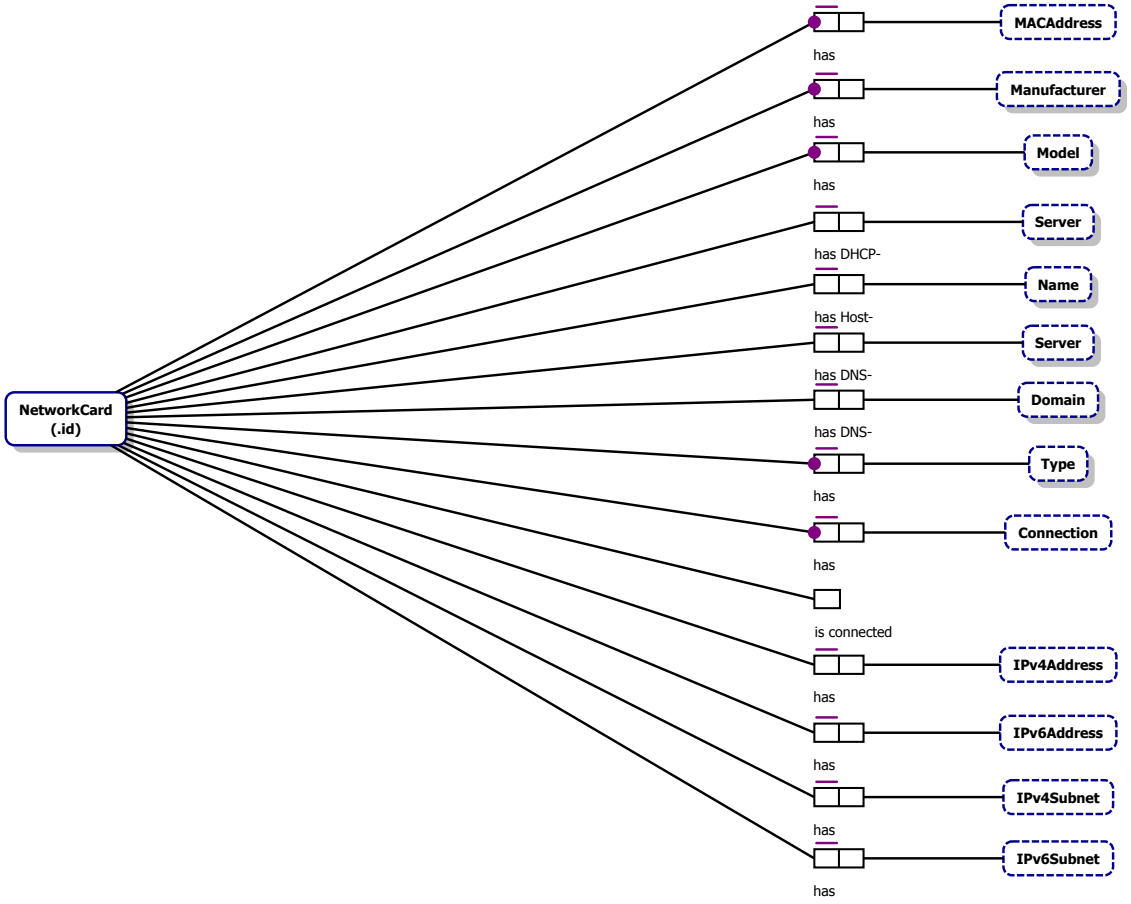


Figure 25: Data model of Open-AudIT network card

The NetworkCard class offers a lot of information about the network cards of a device, as seen in figure 25, such as IP information: IP address, subnet, DNS server and domain, DHCP server, Ethernet type. Furthermore, the Connection field stores the physical hardware device (such as `en0`). Also the manufacturer and model are available, as well as the MAC address. Finally, there is a boolean indicating whether the network card is connected to a network. In all cases, a NetworkCard can have at most one value for each property or exactly one value for the required properties.

In addition to hardware information, Open-AudIT also offers information about software running on a device, which is represented in figure 26. There are four types of software that is recorded: regular applications, services, ODBC drivers and software updates. The latter two are only collected on Windows devices.

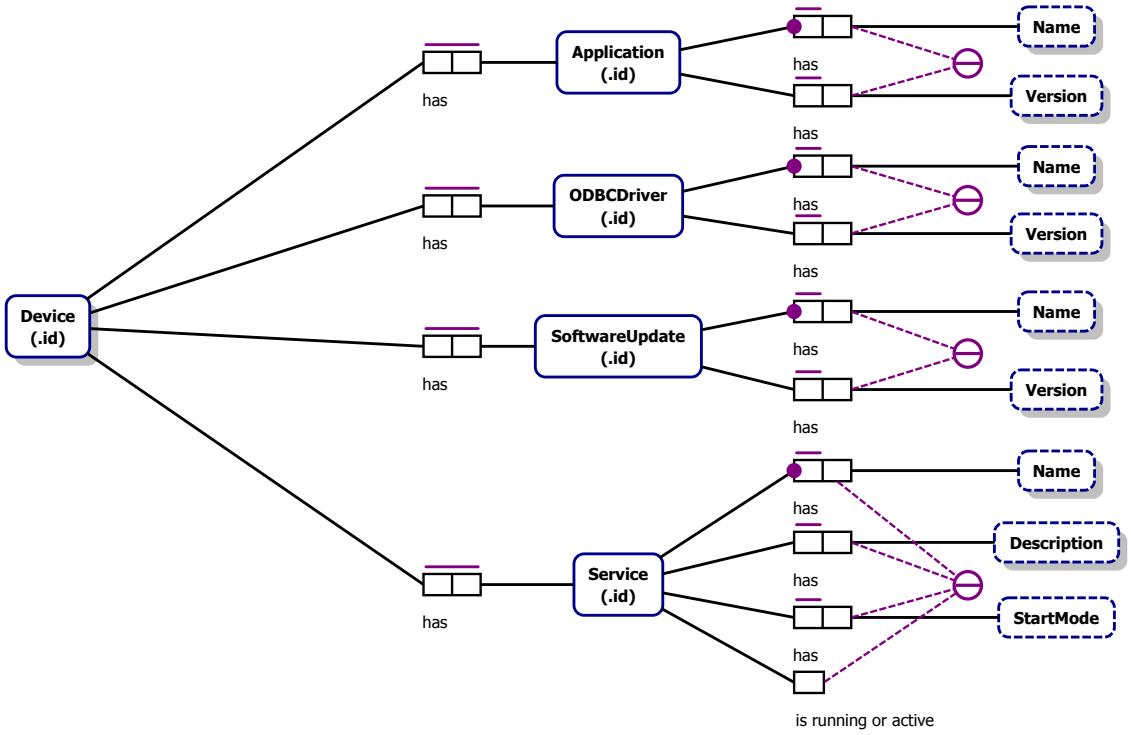


Figure 26: Data model of Open-AudIT software detections

Open-AudIT stores a name and version of each of the types of software it detects, except services. Services have a name and description and a start mode, which is either the file name of the service definition (Linux) or whether the service is running automatically or needs manual starting and stopping (Windows). Services also have a boolean indicating whether the service is currently active (Linux) or running (Windows).

For all these software types, there can at most be one value or each property or exactly one if it is required. Additionally, if there already exists a record in the database with identical information, the old record is reused.

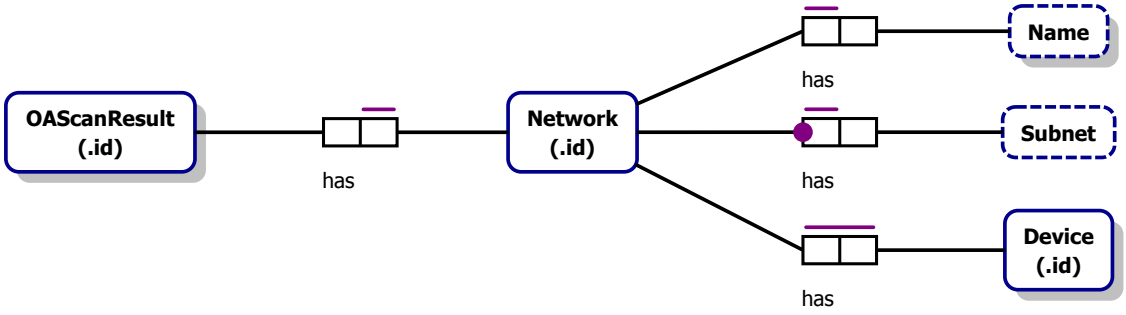


Figure 27: Data model of Open-AudIT networks

In addition to a list of devices, an Open-AudIT scan result also has a list of networks, as represented in figure 27. These networks have a name, which can be set by the operator of the Open-AudIT instance and the subnet that defines the network. Additionally, it is also possible to retrieve a list of devices that belong to that network.

Networks with the same name and the same subnet are allowed in the database, because they belong to a different scan result. They can not be reused, because the list of devices in the network is not necessarily the same over different scan results.

C.3 Snort Scan Result

The results of a Snort scan are stored as a `SnortScan` in the database. A Snort scan has a start and end date and time, which indicate the time window in which the (passive) scan was running. A Snort scan also has an interface address, which is the interface address on the network card that Snort was listening on. This interface address can be used to acquire the subnet to which the interface belongs, which can be used to identify devices being part of a network.

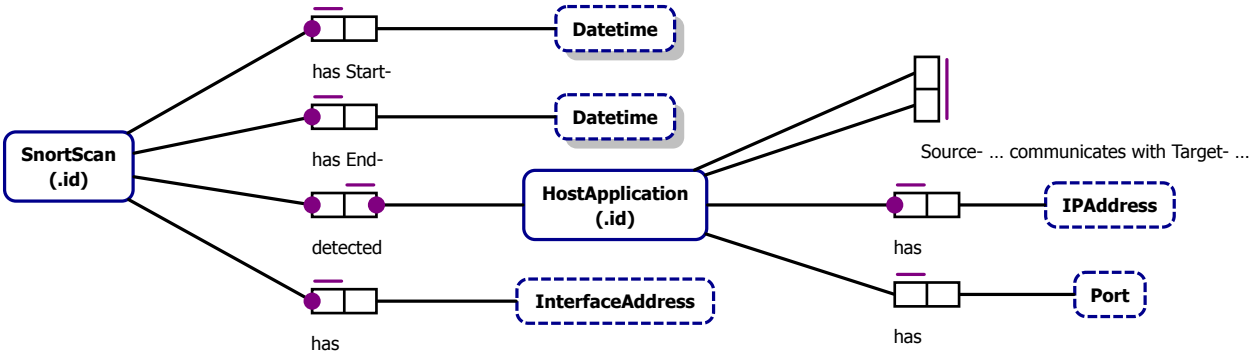


Figure 28: Data model of Snort scan result

In the proof of concept, we are only interested in the communication between devices when we use Snort. Therefore, the data that Snort produces is reduced to the form shown in listing 1. This data contains a timestamp, showing when the communication happened and the information about the source and target device, which both have an IP address and port number.

Listing 1: Example of Snort data

```
05/07-11:30:06.591994 192.168.11.12:8301 -> 192.168.10.200:8301
05/07-11:30:06.699137 192.168.11.10 -> 172.217.17.131
05/07-11:30:06.707212 172.217.17.131 -> 192.168.11.10
05/07-11:30:06.810523 10.0.0.6:8301 -> 192.168.11.9:8301
05/07-11:30:06.878647 192.168.11.10:8301 -> 10.0.0.5:8301
05/07-11:30:07.043559 192.168.11.11:8301 -> 10.0.0.5:8301
05/07-11:30:07.092909 192.168.11.12:8301 -> 10.0.0.5:8301
05/07-11:30:07.093375 192.168.11.12:8301 -> 10.0.0.6:8301
```

In addition, a Snort scan has a list of host applications. These are all the IP address and port combinations that were detected in the sniffed traffic. Both the source and target device are represented as a `HostApplication` in the data model, because it is an application that causes the communication to happen. Each host application communicates with another host application, which is represented by the 'Source- ... communicates with Target- ...' relation. This relation can be used to retrieve a list of host applications that the current host application communicated with.

C.4 Device Management Report

It is possible to use a device management system to get a more complete model. For example, the function of a device is something that can not be determined by a scanner. This information can however be stored in a device management system (of which Open-AudIT is an example) and be used to populate the model.

The Device Management Report is used to simulate a report from an arbitrary device management system. The Device Management Report is a CSV file with the corresponding information. Figure 29 shows the simple data model for a Device Management Report.

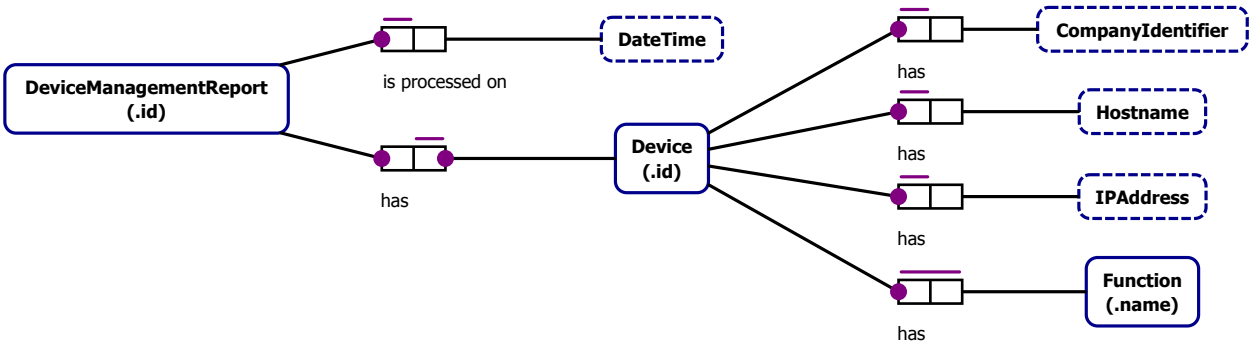


Figure 29: Data model of Device Management Report

A Device Management Report is stored as a `DeviceManagementReport` and has a date and time that indicate the moment the file was processed. In addition, it has a list of devices, all of which have a company identifier, hostname, IP address and one or more functions.

Listing 2: Example of Device management CSV file

```
CompanyIdentifier,Hostname,IPAddress,Functions
nmapScanner,nmapScanner,10.0.0.3,Scanning network
openAudITScanner,openAudITScanner,10.0.0.5,Scanning network
snortScanner,snortScanner,10.0.0.201,Passive scanning|Being a gateway
snortScanner,snortScanner,192.168.11.201,Passive scanning|Being a gateway
snortScanner,snortScanner,192.168.10.201,Passive scanning|Being a gateway
ordinaryHost2,ordinaryHost2,192.168.11.10,Webserver
ordinaryHost4,ordinaryHost4,192.168.11.12,Mailserver
```

Listing 2 shows an example CSV file that could be used as input for the model generation. The company identifier is the same as the hostname is this case, but that is not necessary for the CSV file to be used. Multiple functions can be separated by a vertical bar character.

D Nmap Relational View

Figure 30 shows a relational view of the Nmap Scan database. It shows the tables and columns that form the database.

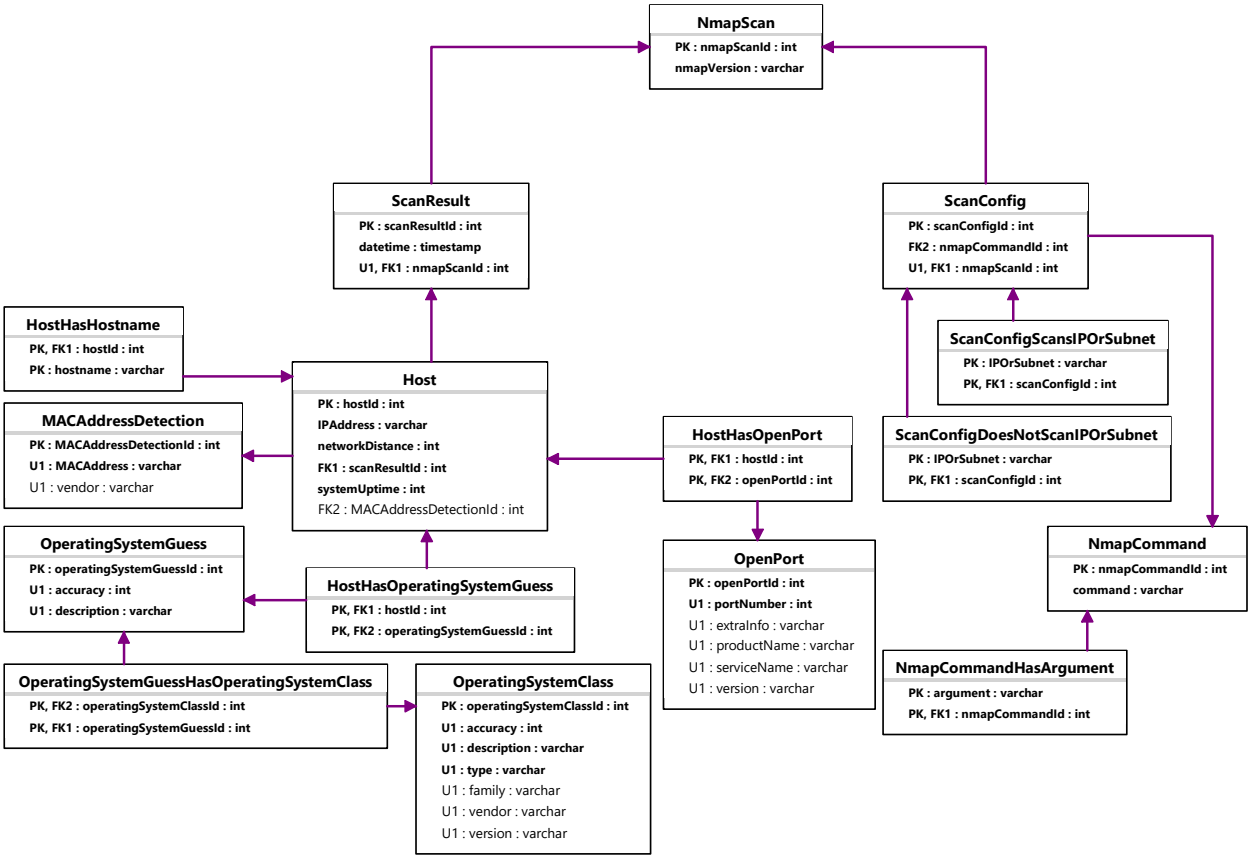


Figure 30: Relational view of the Nmap Scan database

E Open-AudIT Scan Result extra properties

The Open-AudIT scan result is already described in appendix C.2. The purpose of this appendix is to describe the device properties that are not directly relevant to the thesis proof of concept.

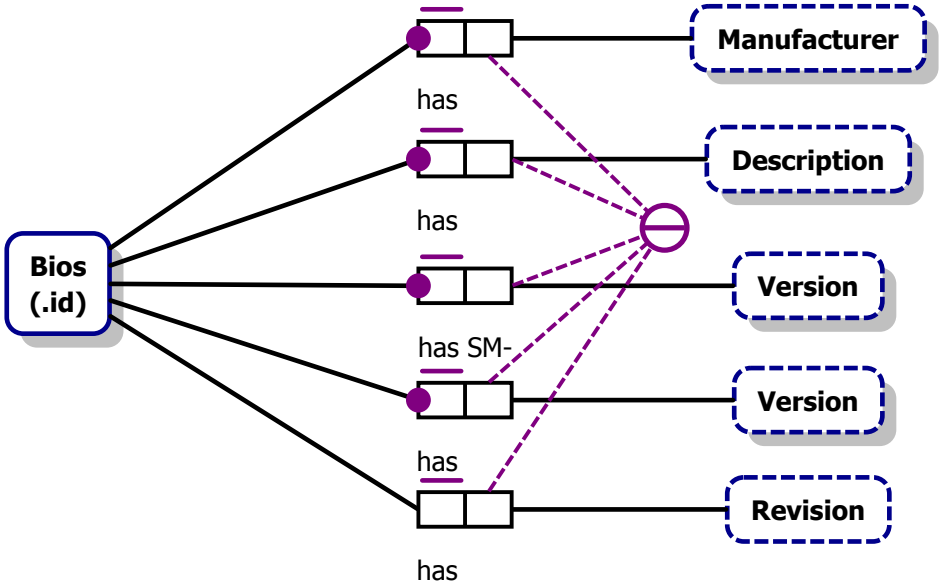


Figure 31: Data model of Open-AudIT bios

As seen in figure 31, Open-AudIT stores the manufacturer and a description about the BIOS. Furthermore, there is version information, such as BIOS version, SM version and which revision the BIOS is.

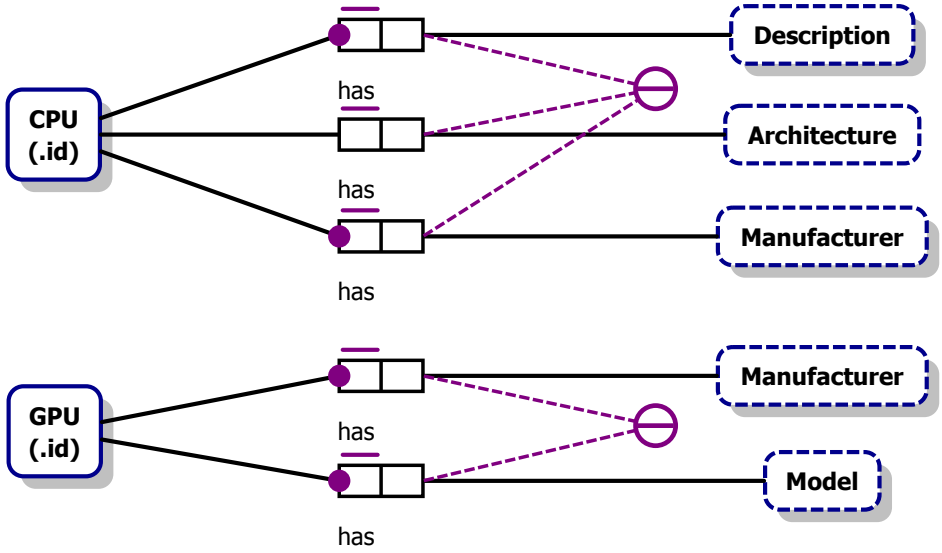


Figure 32: Data model of Open-AudIT CPU and GPU

Open-AudIT also collects CPU and GPU information, as seen in figure 32. From the CPU it collects the description, architecture and manufacturer and from the GPU it collects the manufacturer and model. Open-AudIT also collects the manufacturer, model and version of the motherboard that is used by the device (figure 33).

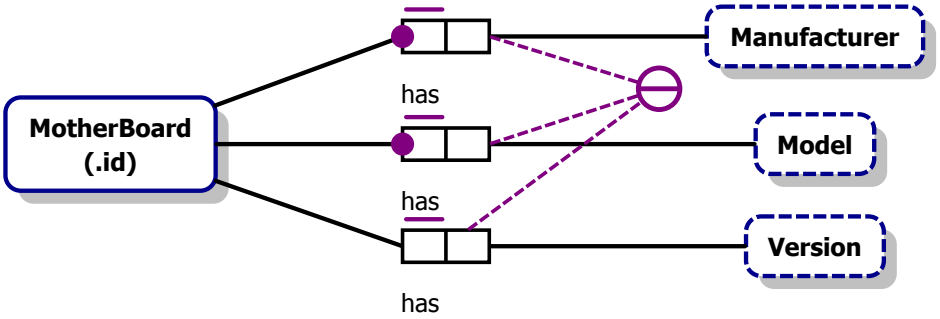


Figure 33: Data model of Open-AudIT motherboard

It is also possible to collect information about users from the Open-AudIT instance (figure 34). Open-AudIT has a list of all users on the device and their properties. For Windows, their is more information to collect, such as password policies, but a username and type is available for both systems.

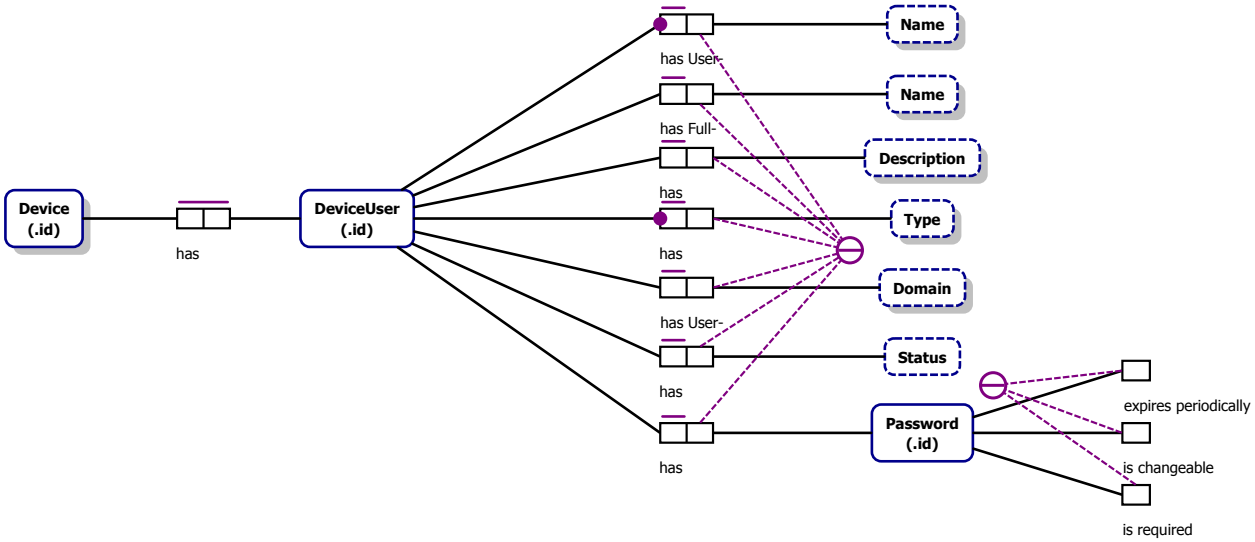


Figure 34: Data model of Open-AudIT user

In all of the above cases, their is an external uniqueness constraint to indicate that a record may only exist once in the database. As the information collected about the hardware and users is static information, it can be reused by subsequent scan results. Therefore, duplicate information is avoided to save storage space.

F Open-AudIT Relational View

Figure 35 shows a relational view of the Open-AudIT Scan database. It shows the tables and columns that form the database.

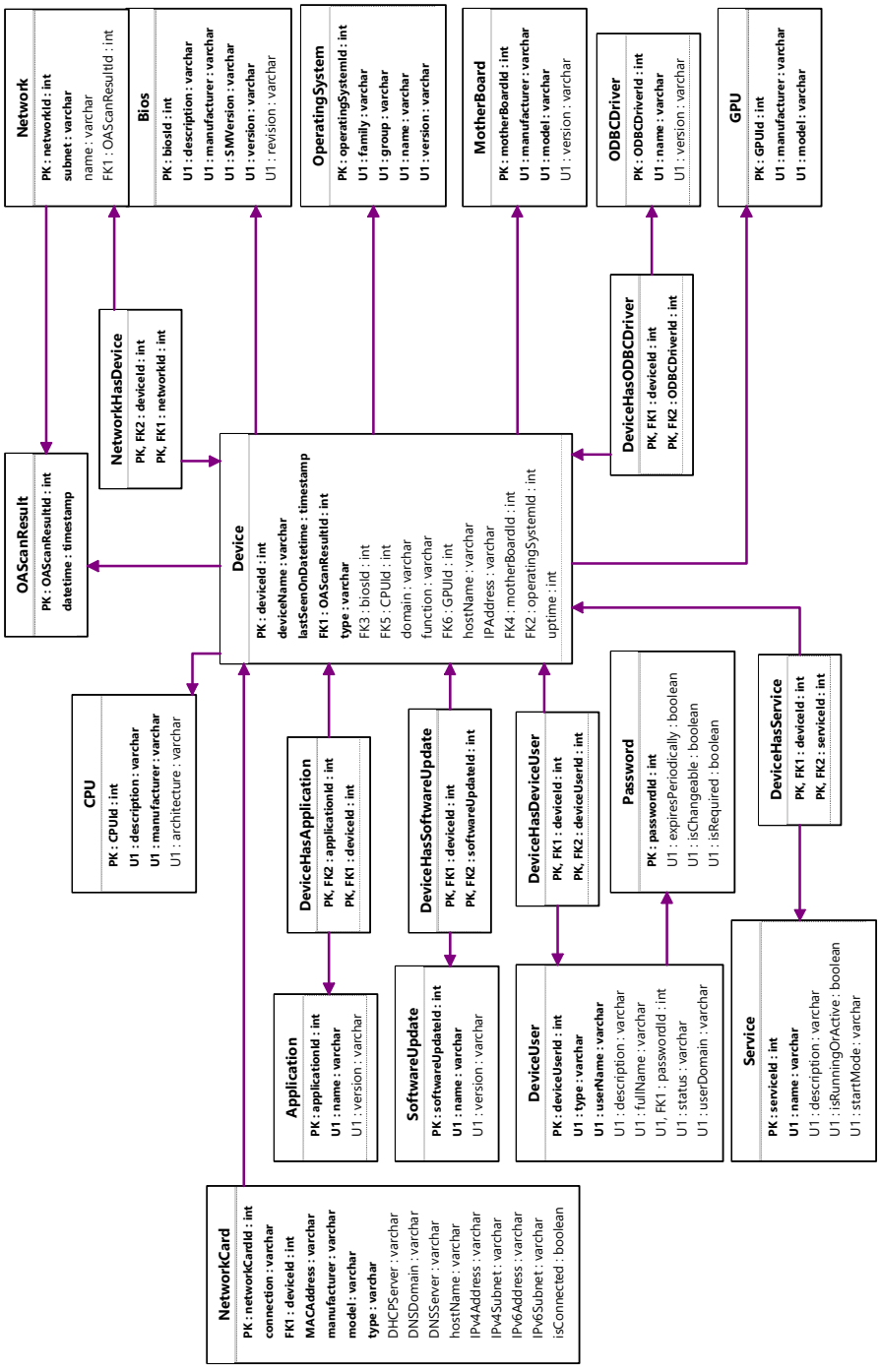


Figure 35: Relational view of the Open-AudIT Scan database

G Snort Relational View

Figure 36 shows a relational view of the Snort Scan database. It shows the tables and columns that form the database.

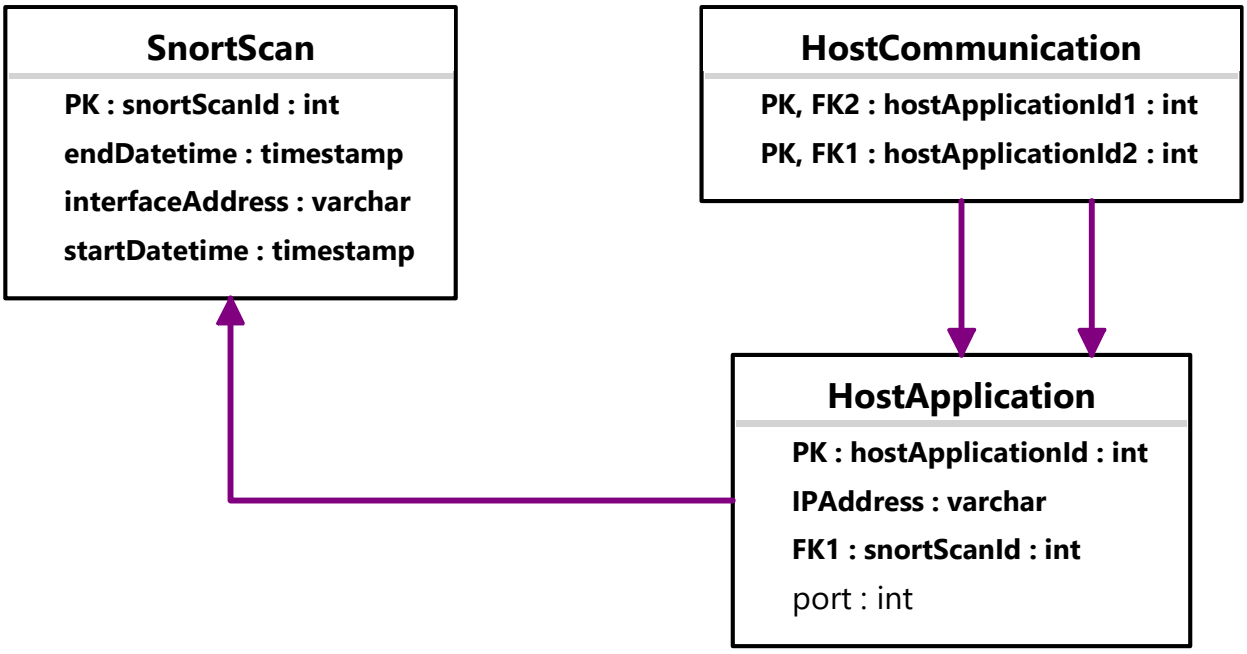


Figure 36: Relational view of the Snort Scan database

List of Figures

List of Tables

List of Listings

References

[1] Amazon Web Services Inc. *What is a Document Database?* Wayback archive: http://web.archive.org/web/20190822125244/https://aws.amazon.com/nosql/document/. URL: https://aws.amazon.com/nosql/document/ (visited on 08/22/2019).

[2] Amazon Web Services Inc. *What Is a Key-Value Database?* Wayback archive: http://web.archive.org/web/20181113133246/https://webcache.googleusercontent.com/search?q=cache:FJ6ScdF1sLMJ:https://aws.amazon.com/nosql/key-value/+&cd=5&hl=nl&ct=clnk&gl=nl. URL: https://aws.amazon.com/nosql/key-value/ (visited on 11/13/2018).

[3] A. Andreev and I. Bogoiavlenskii. "An algorithm for building an enterprise network topology using widespread data sources". In: *2017 21st Conference of Open Innovations Association (FRUCT)*. Nov. 2017, pp. 34–43. DOI: 10.23919/FRUCT.2017.8250162.

[4] Ars Technica. *Confirmed: US and Israel created Stuxnet, lost control of it*. Wayback archive: http://web.archive.org/web/20190424092854/https://arstechnica.com/tech-policy/2012/06/confirmed-us-israel-created-stuxnet-lost-control-of-it/. June 2012. URL: https://arstechnica.com/tech-policy/2012/06/confirmed-us-israel-created-stuxnet-lost-control-of-it/ (visited on 04/24/2019).

[5] M. Ali Aydın, A. Halim Zaim, and K. Gökhan Ceylan. "A hybrid intrusion detection system design for computer network security". In: *Computers & Electrical Engineering* 35.3 (2009), pp. 517–526. ISSN: 0045-7906. DOI: https://doi.org/10.1016/j.compeleceng.2008.12.005. URL: http://www.sciencedirect.com/science/article/pii/S0045790609000020.

[6] T. Binz et al. "Automated Discovery and Maintenance of Enterprise Topology Graphs". In: *2013 IEEE 6th International Conference on Service-Oriented Computing and Applications*. Dec. 2013, pp. 126–134. DOI: 10.1109/SOCA.2013.29.

[7] H. Birkholz et al. "Enhancing Security Testing via Automated Replication of IT-Asset Topologies". In: *2013 International Conference on Availability, Reliability and Security*. Sept. 2013, pp. 341–349. DOI: 10.1109/ARES.2013.46.

[8] Jens Bleiholder and Felix Naumann. *Conflict Handling Strategies in an Integrated Information System*. Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II, Institut für Informatik, 2006. DOI: https://dx.doi.org/10.18452/2460.

[9] Nathan W. Burke. *Continuous Asset Management and Cybersecurity: How We Got Here and Where We're Going*. Wayback archive: http://web.archive.org/web/20181220125133/https://medium.com/axonius/continuous-asset-management-and-cybersecurity-how-we-got-here-and-where-were-going-6d0856001058. Apr. 2018. URL: https://medium.com/axonius/continuous-asset-management-and-cybersecurity-how-we-got-here-and-where-were-going-6d0856001058 (visited on 12/20/2018).

[10] Markus Buschle et al. "A Tool for Automatic Enterprise Architecture Modeling". In: *IS Olympics: Information Systems in a Diverse World*. Ed. by Selmin Nurcan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1–15. ISBN: 978-3-642-29749-6.

[11] Center for Internet Security. *CIS Controls*. Wayback archive: http://web.archive.org/web/20181218140810/https://www.cisecurity.org/controls/. URL: https://www.cisecurity.org/controls/ (visited on 12/18/2018).

[12] Center for Internet Security. *Inventory and Control of Hardware Assets*. Wayback archive: http://web.archive.org/web/20190508090838/https://www.cisecurity.org/controls/inventory-and-control-of-hardware-assets/. URL: https://www.cisecurity.org/controls/inventory-and-control-of-hardware-assets/ (visited on 05/08/2019).

[13] Center for Internet Security. *Inventory and Control of Software Assets*. Wayback archive: http://web.archive.org/web/20190508091131/https://www.cisecurity.org/controls/inventory-and-control-of-software-assets/. URL: https://www.cisecurity.org/controls/inventory-and-control-of-software-assets/ (visited on 05/08/2019).

[14] Cisco Systems Inc. *Enterprise Campus 3.0 Architecture: Overview and Framework*. Wayback archive: http://web.archive.org/web/20181213082540/https://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Campus/campover.html. Apr. 2008. URL: https://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Campus/campover.html (visited on 12/13/2018).

[15] Cisco Systems Inc. *Medium Enterprise Design Profile Reference Guide*. Wayback archive: http://web.archive.org/web/20190724100816/https://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Medium_Enterprise_Design_Profile/MEDP/chap2.html. URL: https://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Medium_Enterprise_Design_Profile/MEDP/chap2.html (visited on 07/24/2019).

[16] De Volkskrant. *In gesprek met Jelle S. (18), die met ddos-aanvallen de Belastingdienst en banken zou hebben platgelegd*. Feb. 2018. URL: https://www.volkskrant.nl/wetenschap/in-gesprek-met-jelle-s-18-die-met-ddos-aanvallen-de-belastingdienst-en-banken-zou-hebben-platgelegd~b3887d7b/ (visited on 05/08/2018).

[17] Edgewise Networks. *Edgewise for Data Flow Mapping*. Wayback archive: http://web.archive.org/web/20190306092550/https://www.edgewise.net/hubfs/Edgewise%20for%20Data%20Flow%20Mapping%20Data%20Sheet%20-%20Web.pdf. URL: https://www.edgewise.net/hubfs/Edgewise%5C%20for%5C%20Data%5C%20Flow%5C%20Mapping%5C%20Data%5C%20Sheet%5C%20-%5C%20Web.pdf (visited on 03/06/2019).

[18] M. Ekstedt et al. "Securi CAD by Foreseeti: A CAD Tool for Enterprise Cyber Security Management". In: *2015 IEEE 19th International Enterprise Distributed Object Computing Workshop*. Sept. 2015, pp. 152–155. DOI: 10.1109/EDOCW.2015.40.

[19] Justin Ellingwood. *An Introduction to SNMP (Simple Network Management Protocol)*. Wayback archive: http://web.archive.org/web/20181130105423/https://www.digitalocean.com/community/tutorials/an-introduction-to-snmp-simple-network-management-protocol. Aug. 2014. URL: https://www.digitalocean.com/community/tutorials/an-introduction-to-snmp-simple-network-management-protocol (visited on 11/30/2018).

[20] Matthias Farwick et al. "REQUIREMENTS FOR AUTOMATED ENTERPRISE ARCHITECTURE MODEL MAINTENANCE - A Requirements Analysis based on a Literature Review and an Exploratory Survey". In: *Proceedings of the 13th International Conference on Enterprise Information Systems - Volume 4: ICEIS,* INSTICC. SciTePress, 2011, pp. 325–337. ISBN: 978-989-8425-56-0. DOI: 10.5220/0003429203250337.

[21] Frank Fransen, Richard Kerkdijk, and Robert Seepers. "Security at machine speed". In: *European Cyber Security Perspectives, TNO* (2019), pp. 38–41.

[22] Ed Gastle. *Enterprise Network Overview [Video file]*. May 2013. URL: https://www.youtube.com/watch?v=7tJKR6z0uW8 (visited on 01/02/2019).

[23] Greg Ghia. *Why Companies Struggle to Manage Their IT Assets*. Wayback archive: http://web.archive.org/web/20181220120735/https://blog.samanage.com/it-asset-management/why-companies-struggle-to-manage-their-it-assets/. July 2009. URL: https://blog.samanage.com/it-asset-management/why-companies-struggle-to-manage-their-it-assets/ (visited on 12/20/2018).

[24] Terry Halpin. "Object-Role Modeling". In: *Encyclopedia of Database Systems*. Ed. by LING LIU and M. TAMER ÖZSU. Boston, MA: Springer US, 2009, pp. 1941–1946. ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_251. URL: https://doi.org/10.1007/978-0-387-39940-9_251.

[25] Terry Halpin and Tony Morgan. *Information Modeling and Relational Databases*. 2nd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008. ISBN: 0123735688, 9780123735683.

[26] Jan L. Harrington. *Relational Database Design and Implementation : Clearly Explained.* Vol. 3rd ed. Morgan Kaufmann, 2009. ISBN: 9780123747303.

[27] Guy Harrison. *Next Generation Databases*. Apress, 2015. ISBN: 9781484213292. DOI: 10.1007/978-1-4842-1329-2. URL: http://dx.doi.org/10.1007/978-1-4842-1329-2.

[28] Hannes Holm et al. "A quantitative evaluation of vulnerability scanning". In: *Information Management & Computer Security* 19.4 (2011), pp. 231–247. DOI: 10.1108/09685221111173058. eprint: https://doi.org/10.1108/09685221111173058. URL: https://doi.org/10.1108/09685221111173058.

[29] Hannes Holm et al. "Automatic data collection for enterprise architecture models". In: *Software & Systems Modeling* 13.2 (May 2014), pp. 825–841. ISSN: 1619-1374. DOI: 10.1007/s10270-012-0252-1. URL: https://doi.org/10.1007/s10270-012-0252-1.

[30] IBM. *An architectiral blueprint for autonomic computing*. Wayback archive: http://web.archive.org/web/20190426121227/https://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf. June 2005. URL: https://www-03.ibm.com/autonomic/pdfs/AC%5C%20Blueprint%5C%20White%5C%20Paper%5C%20V7.pdf (visited on 04/26/2019).

[31] S. Jha, O. Sheyner, and J. Wing. "Two formal analyses of attack graphs". In: *Proceedings 15th IEEE Computer Security Foundations Workshop. CSFW-15*. June 2002, pp. 49–63. DOI: 10.1109/CSFW.2002.1021806.

[32] Mariott International, Inc. *Marriott Announces Starwood Guest Reservation Database Security Incident*. Wayback archive: http://web.archive.org/web/20190424114540/https://news.marriott.com/2018/11/marriott-announces-starwood-guest-reservation-database-security-incident/. Nov. 2018. URL: https://news.marriott.com/2018/11/marriott-announces-starwood-guest-reservation-database-security-incident/ (visited on 04/24/2019).

[33] C. Mavrakis. *Passive asset discovery and operating system fingerprinting in industrial control system networks*. Wayback archive: http://web.archive.org/web/20190307110951/https://pure.tue.nl/ws/files/46916656/840171-1.pdf. Nov. 2015. URL: https://pure.tue.nl/ws/files/46916656/840171-1.pdf.

[34] Microsoft. *Extract, transform, and load (ETL)*. Wayback archive: http://web.archive.org/web/20190726094111/https://docs.microsoft.com/en-us/azure/architecture/data-guide/relational-data/etl. Feb. 2018. URL: https://docs.microsoft.com/en-us/azure/architecture/data-guide/relational-data/etl (visited on 07/26/2019).

[35] MongoDB, Inc. *Document Databases*. Wayback archive: http://web.archive.org/web/20190822122910/https://www.mongodb.com/document-databases. URL: https://www.mongodb.com/document-databases (visited on 08/22/2019).

[36] National Coordinator for Security and Counterterrorism. *Cyber Security Assessment Netherlands 2018*. Aug. 2018. URL: https://www.ncsc.nl/english/current-topics/Cyber+Security+Assessment+Netherlands/cyber-security-assessment-netherlands-2018.html.

[37] Thanh Trung Nguyen and Minh Hieu Nguyen. "Zing Database: high-performance key-value store for large-scale storage service". In: *Vietnam Journal of Computer Science* 2.1 (Feb. 2015), pp. 13–23. ISSN: 2196-8896. DOI: 10.1007/s40595-014-0027-4. URL: https://doi.org/10.1007/s40595-014-0027-4.

[38] NLTimes.nl. *DUTCH BANKS ABN AMRO, ING HIT IN CYBER ATTACK*. Wayback archive: http://web.archive.org/web/20190508084610/https://nltimes.nl/2018/01/29/dutch-banks-abn-amro-ing-hit-cyber-attack. Jan. 2018. URL: https://nltimes.nl/2018/01/29/dutch-banks-abn-amro-ing-hit-cyber-attack (visited on 05/08/2019).

[39] NLTimes.nl. *RABOBANK SERVICES OFFLINE AFTER DDOS ATTACK; THIRD DUTCH BANK HIT*. Wayback archive: http://web.archive.org/web/20190508082122/https://nltimes.nl/2018/01/29/rabobank-services-offline-ddos-attack-third-dutch-bank-hit. Jan. 2018. URL: https://nltimes.nl/2018/01/29/rabobank-services-offline-ddos-attack-third-dutch-bank-hit (visited on 05/08/2019).

[40] NLTimes.nl. *TEEN SUSPECTED OF DDOS ATTACKS ON DUTCH FINANCIAL SERVICES WANTED TO PROVE A POINT*. Wayback archive: http://web.archive.org/web/20190508085017/https://nltimes.nl/2018/02/07/teen-suspected-ddos-attacks-dutch-financial-services-wanted-prove-point. Feb. 2018. URL: https://nltimes.nl/2018/02/07/teen-suspected-ddos-attacks-dutch-financial-services-wanted-prove-point (visited on 05/08/2019).

[41] Nmap. *Port Scanning Techniques*. Wayback archive: http://web.archive.org/web/20190307103934/https://nmap.org/book/man-port-scanning-techniques.html. URL: https://nmap.org/book/man-port-scanning-techniques.html (visited on 03/07/2019).

[42] NOS. *Banken weer paar uur getroffen door DDoS-aanvallen*. Wayback archive: http://web.archive.org/web/20190508082728/https://nos.nl/artikel/2233318-banken-weer-paar-uur-getroffen-door-ddos-aanvallen.html. May 2018. URL: https://nos.nl/artikel/2233318-banken-weer-paar-uur-getroffen-door-ddos-aanvallen.html (visited on 05/08/2019).

[43] Opmantek. *Open-AudIT Documentation*. Wayback archive: http://web.archive.org/web/20190305130757/https://community.opmantek.com/display/OA/Home. URL: https://community.opmantek.com/display/OA/Home (visited on 03/05/2019).

[44] Angela Orebaugh and Becky Pinkard. *Nmap in the Enterprise: Your Guide to Network Scanning*. Syngress Publishing, 2008. ISBN: 9780080558745, 9781597492416.

[45] Stephen Pimentel. *The rise of the multimodel database*. Wayback archive: http://web.archive.org/web/20181114091033/https://www.infoworld.com/article/2861579/database/the-rise-of-the-multimodel-database.html. Jan. 2015. URL: https://www.infoworld.com/article/2861579/database/the-rise-of-the-multimodel-database.html (visited on 11/14/2018).

[46] Ewa Płuciennik and Kamil Zgorzałek. "The Multi-model Databases – A Review". In: *Beyond Databases, Architectures and Structures. Towards Efficient Solutions for Data Analysis and Knowledge Representation*. Ed. by Stanisław Kozielski et al. Cham: Springer International Publishing, 2017, pp. 141–152. ISBN: 978-3-319-58274-0.

[47] Muhammad Azizur Rahman, Algirdas Pakštas, and Frank Zhigang Wang. "Network modelling and simulation tools". In: *Simulation Modelling Practice and Theory* 17.6 (2009), pp. 1011–1031. ISSN: 1569-190X. DOI: https://doi.org/10.1016/j.simpat.2009.02.005. URL: http://www.sciencedirect.com/science/article/pii/S1569190X09000197.

[48] Reuters. *Ukraine points finger at Russian security services in recent cyber attack*. Wayback archive: http://web.archive.org/web/20190424093602/https://www.reuters.com/article/us-cyber-attack-ukraine/ukraine-points-finger-at-russian-security-services-in-recent-cyber-attack-idUSKBN19M39P. July 2017. URL: https://www.reuters.com/article/us-cyber-attack-ukraine/ukraine-points-finger-at-russian-security-services-in-recent-cyber-attack-idUSKBN19M39P (visited on 04/24/2019).

[49] K. Sahatqija et al. "Comparison between relational and NOSQL databases". In: *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. May 2018, pp. 0216–0221. DOI: 10.23919/MIPRO.2018.8400041.

[50] Momina Sohail. *The Biggest Mistakes Businesses Make When Managing IT Assets*. Wayback archive: http://web.archive.org/web/20181220124031/https://technologyadvice.com/blog/information-technology/biggest-mistakes-managing-assets/. Mar. 2018. URL: https://technologyadvice.com/blog/information-technology/biggest-mistakes-managing-assets/ (visited on 12/20/2018).

[51] T. Sommestad, M. Ekstedt, and H. Holm. "The Cyber Security Modeling Language: A Tool for Assessing the Vulnerability of Enterprise System Architectures". In: *IEEE Systems Journal* 7.3 (Sept. 2013), pp. 363–373. ISSN: 1932-8184. DOI: 10.1109/JSYST.2012.2221853.

[52] A.S. Tanenbaum and D. Wetherall. *Computer Networks*. Pearson Prentice Hall, 2011. ISBN: 9780132126953.

[53] Techcrunch. *A hotspot finder app exposed 2 million Wi-Fi network passwords*. Wayback archive: http://web.archive.org/web/20190424120132/https://techcrunch.com/2019/04/22/hotspot-password-leak/. Apr. 2019. URL: https://techcrunch.com/2019/04/22/hotspot-password-leak/ (visited on 04/24/2019).

[54] The Washington Post. *Russian military was behind 'NotPetya' cyberattack in Ukraine, CIA concludes*. Wayback archive: http://web.archive.org/web/20190424093751/https://www.washingtonpost.com/world/national-security/russian-military-was-behind-notpetya-cyberattack-in-ukraine-cia-concludes/2018/01/12/048d8506-f7ca-11e7-b34a-b85626af34ef_story.html?noredirect=on&utm_term=.f0b240b12478. Jan. 2018. URL: https://www.washingtonpost.com/world/national-security/russian-military-was-behind-notpetya-cyberattack-in-ukraine-cia-concludes/2018/01/12/048d8506-f7ca-11e7-b34a-b85626af34ef_story.html?noredirect=on&utm_term=.f0b240b12478 (visited on 04/24/2019).

[55] United States Department of Justice. *Criminal Charges Filed in Los Angeles and Alaska in Conjunction with Seizures Of 15 Websites Offering DDoS-For-Hire Services*. Wayback archive: http://web.archive.org/web/20190424132356/https://www.justice.gov/opa/pr/criminal-charges-filed-los-angeles-and-alaska-conjunction-seizures-15-websites-offering-ddos. Dec. 2018. URL: https://www.justice.gov/opa/pr/criminal-charges-filed-los-angeles-and-alaska-conjunction-seizures-15-websites-offering-ddos (visited on 04/24/2019).

[56] Gaurav Vaish. *Getting Started with NoSQL : Your Guide to the World and Technology of NoSQL*. Packt Publishing, 2013. ISBN: 9781849694988. URL: http://search.ebscohost.com.proxy-ub.rug.nl/login.aspx?direct=true&db=nlebk&AN=562024&site=ehost-live&scope=site.

[57] M. Välja et al. "A Requirements Based Approach for Automating Enterprise IT Architecture Modeling Using Multiple Data Sources". In: *2015 IEEE 19th International Enterprise Distributed Object Computing Workshop*. Sept. 2015, pp. 79–87. DOI: 10.1109/EDOCW.2015.33.

[58] M. Välja et al. "Automated architecture modeling for enterprise technology manageme using principles from data fusion: A security analysis case". In: *2016 Portland International Conference on Management of Engineering and Technology (PICMET)*. Sept. 2016, pp. 14–22. DOI: 10.1109/PICMET.2016.7806662.

[59] Margus Välja. "Improving IT Architecture Modeling Through Automation : Cyber Security Analysis of Smart Grids". QC 20180924. PhD thesis. KTH, Network and Systems engineering, 2018, p. 44. ISBN: 978-91-7729-931-8.

[60] Adam Wedgbury and Kevin Jones. "Automated Asset Discovery in Industrial Control Systems - Exploring the Problem". In: *ICS-CSR*. 2015.

[61] Wikipedia. *Network topology*. Wayback archive: http://web.archive.org/web/20190104114837/https://en.wikipedia.org/wiki/Network_topology. Dec. 2018. URL: https://en.wikipedia.org/wiki/Network_topology (visited on 01/02/2019).

[62] Wireshark Foundation. *Ethernet capture setup*. Wayback archive: https://web.archive.org/web/20181108093045/https://wiki.wireshark.org/CaptureSetup/Ethernet. 2018. URL: https://wiki.wireshark.org/CaptureSetup/Ethernet (visited on 11/08/2018).