Synthetic Packet Generation and Characterization of Nmap Scans

Submitted in partial fulfilment
of the requirements of the degree of

Bachelor in Cyber Security

of Noroff University College

Ørjan Alexander Jacobsen

Bodø, Norway
June 2022

Declaration

I declare that the work presented for assessment in this submission is my own, that it has not previously been presented for another assessment, and that work completed by others has been appropriately acknowledged.

Name: Ørjan Alexander Jacobsen **Date**: June 20, 2022

Abstract

Today the increased use and peoples dependence on the internet have resulted in a significant attack surface for threat actors around the world. Scanning is the most important activity regarding information gathering for use in potential cyber attacks. People today use network scanning for multiple purposes; conducting audit towards own manageable networks, researching purposes or conducting cyber attacks, and during autonomous malicious activity. It's important to clearly have a base knowledge with how a network scanner works, which various types of scanning can be conducted and how this can be identified. Related to pending security action being directed, a base knowledge of which types of tools to conduct various tasks such as audit is crucial to mitigate risks of network scanning. Characterising network activity is a important step of mitigating future risks of cyber attacks. A very important thing is to detect such activity, and it exists multiple mechanisms that can be applied. This thesis will describe the framework for synthetic packet generation and characterization of Nmap scans, together with the evolution and historical background of network scanning methods from the early internet age until today. The framework includes tools for automating generation and processing of packets, together with templates for conducting empirical analysis.

Keywords: *Network scanning, NMAP, network enumeration, detection, scanning*

Acknowledgements

I would like to thank my supervisor, Prof. Barry Irwin, for guidance through this thesis and the valuable continuously feedback.

A much appreciated close friend, wished to be anonymous, have been highly valuable regarding technical guidance and constructive discussions during the work with this thesis.

Thirdly, a much appreciated close friend, have supported and encouraged me with the work of this thesis.

Finally, I would like to thank my fellow student, and friend, Dan Solberg for the valuable constructive discussions in the final runner-up of the thesis.

Contents

List of Figures

List of Tables

List of source codes and commands

1

Introduction

This research report elaborates on the automatic generation of synthetic network scanning packet captures using various types of scan modes and templates using the Nmap scanner. By generating synthetic data sets, where the control of the tasks conducted against each worker host running packet capturing, the analysis of these data sets could be used in a comparative analysis to classify and detect various types of network scans. Following this method elaborated in the paper would allow synthetic data generation for use in a comparative analysis.

The main objective of this research is to automate network scans in order to generate synthetic data sets, which in the future would optimize the method of collecting quality data for analysis purposes. The research outcome is expected to generate data sets containing synthetic data, which would be used as a step in the preparation process of signatures as a final research output.

1.1 Problem Statement

A system administrator would have full control of a network, and network scanning is a large proportion of noise towards networks all around. Today there exists intrusion detection systems capable of detecting such activity and firewalls for blocking malicious network traffic. Network scanning is a step of collecting intelligence and information about a target regarding open and locked ports and is considered the first step in Lockheed Martin's framework Cyber Kill Chain (Lockheed Martin 2015). Reconnaissance activities are important to gain knowledge of potential ways of breaking into a system or network. These threats should ideally be detected and mitigated within the first step of the Cyber Kill Chain.

Not only do threat actors use network scanning, but penetration testers and system administrators also use network scanning as a tool for mapping out exposed services that either can be poorly configured or services running by default after an operating system installation. These arguments emphasize the importance of having proper intrusion detection systems in place, hardened systems, and strictly configured systems. Important knowledge for implementing rules and configurations to such systems is to understand patterns and threats of a cyber attack to enable early detection of these threats. An important step to accomplish this is to primarily detect widely known scanners as an important step for both ongoing scanning activity and prevention of cyberattacks. Understanding of these patterns can be achieved with empirical analysis.

1.2 Research Objectives

The primary objective for this thesis will focus on the creation of a framework for synthetic packet generation and characterization of Nmap scans. The result outcome of this research is to generate synthetic packet captures which is used in a empirical analysis. These packet captures are used in comparison to determine patterns and similarities for each type of scan. Since there is a need of having a certain number of scans to determine patterns and similarities, one of the main outcomes of this research is to generate code for automating the generation of synthetic packet captures in a fully controlled test environment. By doing so, this enables the generation of a given number of scans which leads to a fully scalable environment depending on which number of scans conducted.

By generating synthetic packet captures for comparison, this thesis will as a outcome result in a analysis of various types of Nmap scans compared against various evasion techniques used in Nmap called timing templates. These synthetic packet captures can be used for contribution to a larger network repository and empirical analysis of a chosen number of Nmap scans. Nmap scans based on artefacts and characteristics in packet captures will be investigated in this thesis. In order to process and normalize packet capture, code will be produced in order to extract as many fields as possible from the collected packet capture data. Other outcome of this research is a template for analysis of these packet captures used for comparison and visualisation of the various types of scans.

1.3 Approach/Methodology

Within this section, the approach taken to the research is explained, and the chosen research strategy is elaborated, together with the chosen project methodology and the approach to accomplish the objectives for this research. A Scrum methodology mixed with a Kanban board was the chosen method to streamline the process of accomplishing the objectives in a structured manner. Considering the challenge regarding time constraints, the combination with a clear step-to-step approach is crucial to achieving the main goals.

1.3.1 Research strategy

A separated, fully controlled lab environment must be set up to achieve the full potential of this research, considering where the research is taking place. This includes one or multiple operating systems to compare if different operating systems return different sets of data. Also, multiple scanning software will be used to develop intrusion detection system signatures which will lead to verifying

these against other larger data sets and identification and classification of scans in the wild. The primary goal is to produce signatures and verify these against other data sets. Secondly, these signatures could be used to identify scanning software and potential various scanning types. Included in this controlled lab environment is the generation of data sets, which will be collected through packet captures with multiple scanning tools. The results of this could be used in further research in the future.

The chosen research strategy is via empirical data generation and analysis. Controlled since the whole research is in a virtual environment, where all traffic is within a virtual network. The effects of outside factors are minimized. This research contains analyzing large proportions of data, which especially focuses on the development and verification of IDS signatures which qualifies this research for a quantitative methodology, not unlike the research by Liu and Kensuke Fukuda (2018) where data were collected within an unused */18* IPv4 address block in Japan. Mazel, Fontugne, and Kensuke Fukuda (2014) conducted systematic research where taxonomy was first built, followed by refined anomaly descriptions, analyzing flagged events, and building appropriate signatures for uncharacterized behavior. A systematic methodology is chosen based on the successful research results by Mazel, Fontugne, and Kensuke Fukuda (2014). From the research conducted, metrics are to be presented doing this empirical research.

1.3.2 Project methodology

Chosen project methodology for this project is Scrum with the use of Kanban board. Kanban will streamline tasks and optimize a flow during sprints, which will usually last no longer than a month. Introducing the Scrum framework to this project will create a good overview for breaking larger milestones into smaller goals that can be completed during a sprint.

1.3.3 Approach

The first approach for the project is to research the field regarding scanning, investigating Nmap[1] regarding usage and go through earlier research to gain more in-depth competence in conducting scanning. This will be documented in the first deliverable in the form of a literature review. Secondly is to successfully install virtual machines with the required tools to conduct the research. The next step is to collect data sets by running scans against the different workers and log the data. The output of this will result in data that are going to be used in the process of conducting analysis. It is required to become familiar with various types of scans during this step as well. During all mentioned steps, it is required to document the procedure for possible future research could replicate the environment used during this research.

[1] https://nmap.org

1.4 Scope and Limits

In this thesis the scope and limitations are limited to the network scanner called Nmap. Within the research the limitations are set for using six scanning types in Nmap:

- Connect scan - discussed in section 2.16.4
- Fin scan - discussed in section 2.16.3
- Null scan - discussed in section 2.16.3
- Ping scan - discussed in section 2.16.2
- Service scan - discussed in section 2.16.1
- Xmas scan - discussed in section 2.16.3

There are no restrictions regarding timing templates since this research evaluates all six Nmap timing templates. Time constraints are a significant limitation regarding the scope of this research which involves a risk to the quality of the research outcome. Another limit and risk is the lack of previous exact similar research to examine. The research is limited to one local virtual lab environment not connected to the cloud, which requires a local hands-on appearance. Ethical issues and risks regarding port scanning are mitigated by conducting the research in a local closed lab environment. Finally, limitations and risks for data loss due to hardware crashes are a factor in the research. This is mitigated through regular backups to external hosts.

1.5 Document Structure

The thesis is structured as follows:

- **Chapter 2** Literature Review:
  Within chapter 2 the related work is presented.
- **Chapter 3** Design & Implementation:
  In chapter 3 is the design of the framework presented together with the implementation.
  The setup of the lab environment, worker configurations, developed scripts and preservation of data files are discussed.
- **Chapter 4** Data Analysis:
  Within chapter 4 is the process of generating, processing and analysing data elaborated.
- **Chapter 5** Results:
  In chapter 5 are the results of the analysis presented
- **Chapter 6** Conclusion:
  Chapter 6 concludes the research together with future work

2

Literature Review

2.1 Introduction

In this chapter the literature review is presented together with background resources for the research.

Scanning attacks are often applied to collect information about a network, mainly in the first step of the Cyber Kill Chain (CKC) known as the reconnaissance phase. The Lockheed Martin Cyber Kill Chain is a framework used for identification and prevention of cyber intrusions activity, which is a part of the Intelligence Driven Defense model (Lockheed Martin 2015). The framework describes seven phases of a kill chain to accomplish a successful attack. The first step of the CKC is reconnaissance involving identification of targets such as gathering email addresses and social relationships for important persons in a targeted company, running services or other technical information relevant to moving on with a potential cyber attack (Hutchins, Cloppert, Amin, et al. 2011). Applying mitigation actions within the steps in the CKC framework proves to be efficient since during a cyber attack all seven phases needs to be successful to allow a cyber attack to occur (Lockheed Martin 2015).

During the phase of intelligence gathering, three large steps exists; foot printing, scanning and enumeration (Arkin 1999). These are methods used for intelligence gathering and used as an initial step to gain access to a network. Other malicious activities such as scanning in a paper by Treurniet (2011) argues that it is difficult to detect scanning since it's neither well-behaved nor well-understood. Insight can be given by classifying known activities, giving visibility to threats and extending awareness of network activities (Treurniet 2011).

Scanning are also a central part of a penetration testing, where vulnerabilities are identified. During the phase of scanning and information gathering activities such as port and vulnerability scanning

are conducted, as a normal thing for gathering intelligence. This proves the importance of detecting potential cyber attacks in the early stage by applying early warning such as scanning detection and categorisation.

Popular tools conducting penetration testing is the large open-source project Metasploit, which comes default with a large database of already tested exploits, and is currently one of the biggest open-source projects within the field of penetration testing and information security (Singh 2018). Metasploit is further described in section 2.11. The port scanning tool Network Mapper (nmap), elaborated in section 2.6, is a useful network scanning tool for information gathering involving targets running services and open ports (Pinkard and Orebaugh 2008). Nessus is a tool used for vulnerability assessments, described in section 2.7.

Within the sections in the literature review the historical development are presented, various types of scanners are elaborated, how research have classified network scans, how network scan works and how to detect it.

2.2 The historical development of network scanners

On the 2nd November 1988, a worm with the name of Morris Worm was created by Robert Morris. The main objective was to steal military information. The result out of this worm was a conviction against Robert Morris, who targeted 6.000 computers on the Internet. Morris was the first person ever getting convicted for breaking the law of gaining unauthorized access to protected computers. This led to the creation of what we today know as a Computer Emergency Response Team (Daya 2013; FBI 2019).

Later in the 1990s, Kevin Mitnick committed several crimes, where tools such as scanners were a central part of the activities committed. Other persons included in these activities were Zyklon Burns, Patrick Gregory, and Chad Davis, where Burns received 15 months in prison, while the others received 26 months. During a hack against the White House, Zyklon initially used a CGI scanner, which especially targets vulnerabilities for common gateway interface. In the book, a port scanner is mentioned as a central breaking-in point to gather intelligence about the open ports during a bank break-in (Mitnick and Simon 2009). Mitnick used nmap to scan for all open ports during an intrusion later in London, where the network were mapped, identified a router and open ports (Mitnick and Simon 2009), which proves that nmap fast grew becoming a popular tool.

On 27th February 1995, Julian Assange initially released a version of the TCP port scanner Strobe (Assange 1995). Strobe evaluates listening TCP ports on hosts segmenting bandwidth between hosts in an efficient matter (Assange 1999) and finds open ports on a system (Higgins 2011; Blyth 1999). Julian Assange had earlier a history of hackings taken place, where he on the 5th December 1996 pleaded guilty for hacking Nortel, where he was sentenced to three years of probation and a fine on $2.300 (Higgins 2011).

In September 1997 the first release of the port scanner nmap was released. Nmap was mainly created to conduct an audit of systems for system administrators to keep one step ahead of hackers. It has a lot of features created to perform scans from basic ICMP pings to check for alive hosts, to conducting operating system detection, and it also supports TCP and UDP (Pinkard and Orebaugh 2008).

The history moves on, and other tools with the main purpose of conducting port scanning were created such as NetScanTools, Superscan, Angry IP Scanner, and Unicornscan. The working area and feature description of these tools would be further described in section 2.4. These tools were initially released within the period of 1998 and 2006.

During 1998, the large focus on network security grow. At the time the Security Administrator Tool for Analyzing Networks (SATAN) were shown to be obsolete and other commercial scanners shown to be more comprehensive. Nessus were created in 1998, growing from a small community to a community with 149 participants and 114 plugin writers in 2008. During a licence change of Nessus version 3 at 5th October 2005 changing from open-source to proprietary multiple forks where created as a consequence. These two projects are Porz-Wahn and OpenVAS (Rogers 2011).

In the late 2009, John Matherly asked friends trying out his new project called Shodan, an idea conceived in 2003. The goal of the project were to map all devices connected to the internet. The results from the project were extraordinary, and Shodan shown it's possibilities to find water plans, power grids, the cyclotron at Lawrence Berkeley National Laboratory, SCADA systems and thousands of unsecured Cisco equipment among other things. The scanning service has shown useful for security researchers, threat actors and hackers since the results of a scan could be used to gain unauthorised access to critical systems. Some say that barely no competence is needed to take advantage of the results if the targeted host has not protected their systems good enough started working on a project named Shodan, where the goal was to map out devices connected to the internet (O'Harrow Jr 2012).

2.3 Scanning activity and security audit towards own network

A system administrator could implement network scanning towards its network to identify alive hosts within the network, figuring out identification of operating system in use and service information. A vulnerability scan could be conducted to identify weak spots within the network, like a comparison between gathered service information and a vulnerability database. This is often conducted during an audit of a network that aims to identify vulnerabilities without interfering with running services. This method is called vulnerability analysis. The gathered information could be further used in the evaluation of the vulnerable parts of the network. The assessments during the network and vulnerability scan base the signatures on which operating system is used, running services and vulnerabilities (Holm et al. 2011). The popular scanner Nmap are widely used by security professionals during compliance testing, asset management, and system inventory. It has multiple usage areas such as verifying firewall filters, host discovery, policy compliance, and system monitor uptime (Pinkard and Orebaugh 2008).

2.4 Scanning activity towards others networks

Today organizations exist and offer services providing data sets generated out of Internet scanning. Two of the largest organizations offering this are Censys[1] and Shodan[2]. These two organizations provide a search engine-like interface to their datasets and operate on a subscription-based model. There search engines index large databases of the results of their scanning activities of networks and devices connected to the Internet. Shodan is a widely known search engine finding devices connected to the Internet. It can provide a specific version of a software (e.g. Apache version x) running on a host connected to the internet. Shodan gathers banners containing versioning information, identifying a given host with a given IP address together with version of software running (Matherly 2015).

2.5 Autonomous scanning activity by malware

After multiple large attacks at the end of 2016, the Mirai botnet got known. Mirai spreads through scanning a large set of IP addresses across the Internet, looking for vulnerable devices to exploit (Antonakakis et al. 2017). It then tries to gain control of the device by applying default credentials. This continues until the network contains controllable devices that are set up successfully. Mirai communicates to a command and control server where the malware is deployed to the host and then further spread out (Gallopeni et al. 2020). This shows that the Mirai botnet initially uses autonomous scans to find exploitable hosts to deploy its malware too.

2.6 Network Mapper (nmap)

Network mapper (nmap) is free software released in September 1997, mainly created for use to scan networks under security audits by system administrators and other IT professionals. It has a wide range of capabilities, such as scanning single hosts for detecting the versioning of software running on open ports. Another capability is creating IP packets whose utilised to perform more specific scans towards a single or multiple hosts. The software has various usage areas, such as asset management, system, and network inventory, compliance testing, and security auditing. During audits, it could be useful to conduct OS versioning to detect the need for patching and updates. Since nmap could usually be used for this purpose, it could also be used during harmful intrusions to map the infrastructure of a network. Furthermore, this could be used to exploit vulnerabilities within the network.

Nmap is free and runs on multiple operating systems, most used on Windows, Linux, and MacOSX. Regarding types of scan that could be conducted in nmap, section 2.16 describes the various ways of behaviour for scans. Nmap have a lot of scanning capabilities, such as ICMP echo request, stealth scan, XMAS scan, null scan and connect scan (Pinkard and Orebaugh 2008).

[1] https://censys.io
[2] https://www.shodan.io

2.7 Nessus

Nessus is a enterprise proprietary vulnerability assessment software aimed for the enterprise market. The software is used often to provide internal scans when conducting audit to help with vulnerability assessments. The community for Nessus is growing, and it has many contributors writing audit plugins for the software (Rogers 2011). Nessus comes in various price ranges depending on the use of the software[3].

2.8 Unicornscan

Unicornscan has the capabilities of conducting port scanning, which works as an asynchronous scanner. It's able to gather information quickly and uses distributed TCP/IP stack for fast responses (Hoque et al. 2014). Unicornscan has also capabilities for banner grabbing, port scanning, logging, and identification of operating systems and applications (Inc. 2004).

2.9 Shodan

Shodan is the result of a idea from John Matherly. It's used as a search engine looking for devices connected to the internet. The primarily part of data that Shodan collects is the banner result from a scanned host. Information given back to Shodan after a scan where a port is open, could return service version information and other specifics for the given service (Matherly 2015).

In the late 2009, John Matherly asked friends trying out his new project called Shodan, an idea conceived in 2003. The goal of the project were to map all devices connected to the internet. The results from the project were extraordinary, and Shodan shown it's possibilities to find water plans, power grids, the cyclotron at Lawrence Berkeley National Laboratory, SCADA systems and thousands of unsecured Cisco equipment among other things. The scanning service has shown useful for security researchers, threat actors and hackers since the results of a scan could be used to gain unauthorised access to critical systems. Some say that barely no competence is needed to take advantage of the results if the targeted host has not protected their systems good enough started working on a project named Shodan, where the goal was to map out devices connected to the internet (O'Harrow Jr 2012).

2.10 Zmap

In 2013 the Zmap project was developed, a network scanner designed to scan the whole IPv4 address space from one single host. It is capable of doing so within 45 minutes. Compared to Nmap, Zmap is capable of scanning the whole IPv4 public address space 1300 times faster (Durumeric, Wustrow, and Halderman 2013).

[3] https://www.tenable.com/products/nessus

2.11 Metasploit

The Metaspolit framework[4] is a open source penetration testing framework with a large database of tested exploits, currently one of the biggest open-source projects within penetration testing and information security. Within the framework it exists a enormous database with exploits and vulnerability assessment which allows an pentester to compromise the system security where vulnerabilities exists within the network stack, operating system or services. Other useful features is payload which is the running code doing the work after it's conducted the exploitation on a system. This could be traffic to conduct command and control (C2) to establish a connection between the remote host and an threat actor. Metasploit are designed in a way to easily integrate new exploit code to the framework since the modules of the complete system are small building blocks (Singh 2018).

2.12 Classification of network scanning activities

In the cyber space today threats have become a major concern for businesses. A paper by Fekolkin (2015) discusses what is IDS and IPS and which security measures needed to be implemented other than anti-virus and firewall. In network intrusion a network scan is a common reconnaissance activity (Barnett and Irwin 2008). The paper further describes a classification of network scanning and also illustrates the complexity of the activity. Liu and K. Fukuda (2014) shows that over 96% of all anomalous events can be detected and labelled. This results would make unlabelled source rate very low.

For the mission of detecting abnormal activity within a network, a taxonomy of anomalies could be created to detect such behaviour. There exist multiple techniques for detection, such as creating signatures for pattern recognition, statistical methods, among other things. These methods provide information that further could be used for anomaly detection. Traffic metrics could be used for detecting anomalies, such as scanning activity and DDoS. Labelling each occurring event could be applied to categorise and differ between abnormal traffic and normal traffic. Categorisation of traffic could be separated into subcategories such as unknown, anomalies, or normal (Mazel, Fontugne, and Kensuke Fukuda 2014).

Scans are one type of event among others, which could be represented through characteristics and patterns. It might be several hosts or single machines that are scanned. Scans are conducted to gain knowledge about a target. This can be opened ports, which operating system running, or versions of software that are using the open ports. In a network scan, hosts alive could be detected, among services and ports open. During a host scan, the same results could be retrieved, through checks towards the allowance of packets to a specific port. Network scans can be used to gather intelligence of how the network is structured, how many clients are connected, and which services at which version are located on the network. After gathering intelligence about clients in the network, the intelligence itself can tell something about which exploitation could be used against the various services at which specific versions (Mazel, Fontugne, and Kensuke Fukuda 2014).

Various traffic characteristics during scanning would be caught, such as the following. During TCP scanning, SYN is used to initiate connection while ACK is mapping ports by filtering port answers through ICMP (Mazel, Fontugne, and Kensuke Fukuda 2014). A signature is created with rules for

[4] https://www.metasploit.com

different attributes of the traffic. Abnormal characteristics such as scanning attempts can be detected and generated a signature out of the behaviour of the traffic recently conducted (Mazel, Fontugne, and Kensuke Fukuda 2014).

While identifying scanning techniques, a quantitative amount of data is required to conduct identification and evaluation of scans conducted. During the research of Barnett and Irwin (2008) the port scanning tool nmap was used to construct the scanning, and tcpdump was used to capture packets on the network interface of a FreeBSD host. Scripts were constructed to speed up the analysis and be able to analyze one month of data in about five minutes.

2.13 Horizontal and vertical scanning

A horizontal scan sends probes to each host connected to the Internet connectable outside the firewall (Leonard et al. 2012). This could also be conducted on a local network. During a horizontal scan, one single port are targeted on multiple hosts. The most use case of this scanning method is to identify vulnerabilities for a service running on a specific port (Barnett and Irwin 2008).

Vertical scanning sends probes on various ports against one specific host on the network (Barnett and Irwin 2008). If a known vulnerability is out and a user using a scanner are interested to know how many or who are using the explicit version of a software, a vertical scan could be conducted to map this out.

2.14 Detection of scanning activities

It exists software today which are able to detect scanning activity. The technical term often used for this activity is Intrusion Detection Systems (IDS).

Snort is used as a standard Intrusion Detection System since it has a high level of customisation and easy to use. According to a study by K. Chumachenko and D. Chumachenko (2017), intrusion detection system (IDS) and intrusion prevention system (IPS) can detect zero-days, but also prevent known attacks and other steps after the initial attack (e.g. port scans). Jammes and Papadaki (2013) points to the same argument regarding detection of port scanning and exploits if Snort is correctly configured and has the signatures for given exploits. There exists various types of techniques to execute a scan and in some cases these techniques can be stealth according to a paper by Patel and Sonker (2016), though the challenge is to detect these types of scans. Snort has according to the paper the capabilities to handle such activity. Thapa and Mailewa (2020) review various IDS and IPS tools, such as Snort [5], Suricata[6], Zeek[7] among others. Day and Burns (2011) discuss tests comparing if Suricata shows increase in system performance and accuracy compared with Snort, which is single threaded.

Snort aren't able to detect port scanning while the flag $sfPortscan$ are enabled. The frag3 preprocessor defragments the IP packet to prevent malicious packets to escape detection. Packets could be intentionally fragmented to avoid detection, Snort therefore needs to defragment the packet to check

[5] https://www.snort.org
[6] https://suricata.io
[7] https://zeek.org

the contents. The stream5 preprocessor is another component which is important to detect nmap scanning. The stream5 preprocessor reconstructs the TCP flow and has capabitilies for reconstructing UDP sessions. This enabled rules to be applied to the data stream. Snort cannot detect port scanning without this feature (Jammes and Papadaki 2013).

According to a study by Liu and Kensuke Fukuda (2018) the observation of cyber attacks and anomalies have among security researchers made them able to developed cyberspace monitoring approaches to detect such anomalies. The study also shows that darknet provides effective passive monitoring approaches, which often refers to globally routable and unused IP addresses that are used to monitor unexpected incoming network traffic. This is an effective approach for viewing network security activities remotely. The main goal of the study was to create and propose a simple taxonomy of darknet traffic.

There exists multiple other ways to create taxonomies of network anomalies. Such as a study by Mazel, Fontugne, and Kensuke Fukuda (2014) where previously documented anomalous events are classified using data collected through six years. The report points out that previously unknown events are now accountable for a substantial number of UDP network scans, scan responses and port scans. With their proposed taxonomy the number of unknown events have decreased from 20% to 10% of all events compared to the heuristic approach (Mazel, Fontugne, and Kensuke Fukuda 2014). Such scans can be handled by nmap (Pinkard and Orebaugh 2008), which is a network scanner which has many features, used for administration, security auditing and network discovery. Subverting firewalls and IDS, optimisation of nmap performance and automating common network tasks could also be done with the nmap.

2.15 Avoiding detection (evasion)

According to the study of Jammes and Papadaki (2013) Snort were able to detect five out of six of the different timing templates used for Nmap, which is default delivered by Nmap. The last template, the paranoid template, were Snort not able to detect according to the study (Jammes and Papadaki 2013). These templates are named insane, aggressive, normal, polite, sneaky and paranoid. The difference of these are the minutes waited between sending a probe to the target host. During the paranoid mode, Nnmap waits 5 minutes while at the sneaky mode it waits 15 seconds (Pinkard and Orebaugh 2008; Jammes and Papadaki 2013). During a scan execution, the parameter $-T$ could be used to set the template. The higher the number are, the faster the scan are (Pinkard and Orebaugh 2008).

During a Nmap scan, other parameters could also be applied such as the Time-to-Live parameter. Other methods for avoiding detection is to randomize the order hosts are scanned in. Nmap have capabilities for sending UDP and TCP packets with invalid checksums, which often firewalls and other security controls drops due to the fact the checksum aren't checked. If a reply is given, these are often from firewalls or other security controls.

2.16 Theory behind network scanners and exploration of scanning methods

Network scanning is a reconnaissance activity conducted within the field of network intrusion (Barnett and Irwin 2008). During an intelligence-gathering step, network scanning is one of the first steps before an attack is about to happen (Houmz et al. 2021). Conducting a network scan would reveal information about hosts on a network, which includes running services and applications, open ports, and operating system details. The main techniques applied during network scanning are version and service detection, scanning ports, operating system detection, and network mapping (Pinkard and Orebaugh 2008). During a network scan devices and services vulnerable for attacks could be unveiled (Houmz et al. 2021).

2.16.1 Port scan and service scanning

By conducting port scanning, information about running services and open ports could be retrieved. This could further be used to determine misconfiguration on a system or version of the software which could be mapped against known exploitable vulnerabilities. It exists multiple types of port scanning methods, further elaborated below (Pinkard and Orebaugh 2008). Port scanning is a reconnaissance technique used to discover services running on open ports which could be exploited to gain access to a system. Port scanning is primarily divided into vertically and horizontally scans, further elaborated in section 2.13. In vertical attacks, described in section 2.13, multiple ports are scanned on one host. This is a common process when a threat actor is interested in one specific host to attack (Dabbagh et al. 2011).

2.16.2 Host discovery using ICMP echo requests (ping scan)

ICMP echo requests could be used for the purpose of determining whether a host is online or offline. According to the RFC 792 (Postel 1981) it exists multiple types of ICMP types which are messages for a request or reply message, such as timestamp, redirect, echo, timestamp, information, source quench, destination unreachable and time exceeded. The echo type, type number 8 is a echo request message sent to a target to determine if an reply could be received. The echo reply message has type number 0. The ICMP echo requests can be used for determine whether an host is online or not (Postel 1981; Lyon 2009). Many firewalls have by default configured to deny echo ICMP requests (Arkin 1999). The nmap scanner can conduct this sort of operation by using the parameters $-PE$ in a scan (Lyon 2009).

2.16.3 Stealth scans

A stealth scan is what the name suggests it to be, to be a scan that should not be detected. Some of the evasion techniques used for a stealth scan is using various flag settings and fragmentation. Fragmentation means splitting up the scan requests over multiple packets by attempting to not getting detected while scanning. This applies to the TCP-based scanning requests (Pinkard and Orebaugh 2008).

Other evasion techniques such as slow and low scanning can be applied in order to not getting detected by an intrusion detection system (IDS). The technique limits the number of ports targeted for

a larger timespan than a fast scan. By reducing the time of a scan reduces the risk of being detected. Some of the stealthy scanning techniques is NULL scan, XMAS scan, FIN scan, SYN/ACK scan and ACK scan (Pinkard and Orebaugh 2008).

XMAS Scan

A Nmap XMAS scan sets the FIN (finish), PSH (push) and URG (urgent) flag during a scan. These flags should not occur in normal traffic and are therefore called invalid flags. The conventions for these types of packets are not established meaning various stacks in TCP would respond differently. Some TCP stacks would return a RST (reset) packet from both open and closed ports, while other systems can return no reply at all. The most typical is a reply with RST/ACK (reset/acknowledge) for closed ports, while open ports not replies at all and drops the received packet. The parameter for conducting a TCP XMAS Scan in nmap is $-sX$ (Pinkard and Orebaugh 2008; Lyon 2009). The XMAS scan can based on this determine open and closed ports.

Null Scan

A nmap null scan does not set any flags, it sets the flag header to 0. The name null scan abbreviates from this. Nmap null scan has the ability of being stealthy by avoiding intrusion detection systems, firewalls and system logging. A open port will, like the XMAS scan, drop the received packet and not reply to the scan request. Meanwhile, an closed port replies with RST/ACK (reset/acknowledge), where some systems replies with RST (reset) for all ports, and other systems might not reply at all to any received packets (Pinkard and Orebaugh 2008; Lyon 2009). This is similar to the XMAS scan, elaborated in section 2.16.3. The parameter for conducting a null scan in nmap is $-sN$ (Pinkard and Orebaugh 2008).

FIN Scan

The FIN scan are similar to the XMAS scan described in section 2.16.3 and the Null scan described in section 2.16.3. Though, it only sets the FIN (finish) flag during a scan compared to the two mentioned scanning methods. A port is considered closed if a RST (reset) is given in the reply. Considered as a open port is no reply (Pinkard and Orebaugh 2008; Lyon 2009). The parameter for FIN scan in nmap is $-sF$.

2.16.4 Connect scan

During a connect scan, a three-way handshake is performed to open a connection to the given target. The connect scan is abbreviated in this research to *TCP full scan*. A regular reply would be an SYN/ACK (synchronize/acknowledge) packet if a TCP port is listening on the given port, otherwise, a host would respond with an RST/ACK (reset/acknowledge) (Pinkard and Orebaugh 2008; Dabbagh et al. 2011). This depends also on other factors such as the firewall configurations, where hosts would reply with RST/ACK if the firewall blocks the connection. TCP connect scans have a higher likelihood to be identified and logged compared to stealth scans, described in section 2.16.3. By using the Nmap connect scan, Nmap uses the operating system for connection establishment rather than customising the raw packet when sending it to a target. This type of scan creates a large number of connection attempts to various target ports, and this often indicates a ongoing port scanning. Due to this, the risk of being detected is significantly higher than other types of scans. Systems that do not close TCP

connections in a proper manner can suffer from denial of service (DoS) conditions. The parameter to use in Nmap for conducting a connect scan is $-sT$ (Pinkard and Orebaugh 2008).

2.17 Summary

The related work for this thesis is presented in chapter 2. Important to know is how a network scanner works and how can history tell about the development from the first day until today. The theory of when a network scanner are applied and whose using it is important knowing. There does not exist only one network scanner, therefore are multiple network scanners discussed within the chapter. Various types of Nmap specific scanning types, timing templates and evasion techniques are elaborated within section 2.16. The chosen Nmap scanning types for this research are described in the limitations in section 1.4.

3

Design & Implementation

3.1 Introduction

This chapter presents the overview of the design, together with the technical implementation of the lab environment and code used to conduct the data generation.

The importance of automatically generating synthetic packet captures is further discussed in this chapter. Having a standard of conducting each network scan enhances the reliability of each scan being conducted in a similar standardized way compared to manually running these Nmap scans, where differences might appear depending on the user inputs while executing the commands and human error. In cases where this is conducted manually, the risk of not standardizing the scans can lead to incomparable data as an output during the analysis phase. Therefore, the importance of automating the scans with a standard is crucial. The automation is also time-saving compared to a manual scanning procedure. This automation is described in this chapter, where the main components are described. Among these is the important scanner script elaborated in section 3.4.1. By the use of various timing templates, the workers finish the scans at different times, which makes a manual scanning method very time-consuming and not effective. This design and implementation chapter will describe the setup of such an automation packet capture generation.

3.2 Lab Environment

A segmented lab environment was needed in order to accomplish the objectives of generating synthetic scanning data for comparison. Within this chapter, the design choices and implementation details are elaborated. Chosen virtualization platform is VMware Workstation in combination with the

operating systems Ubuntu Server for scanning targets and Kali Linux for the scanning host. A guide for installation of Kali Linux can be followed on the official Kali web page [1].

3.2.1 Software and operating system implementation

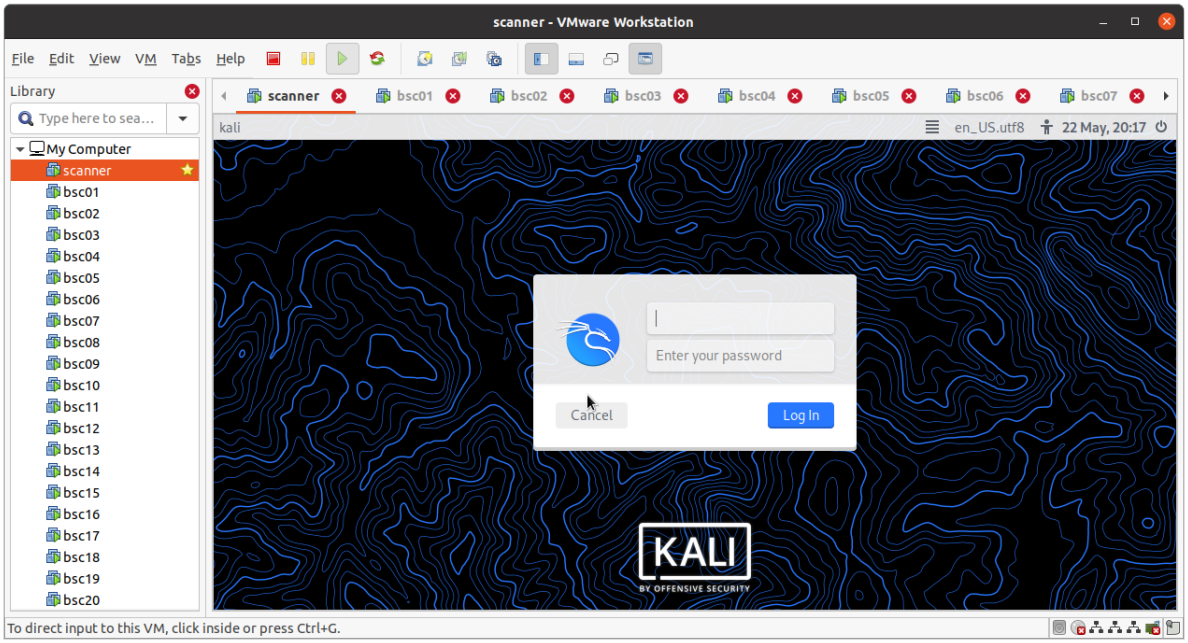


Figure 3.1: Lab environment shown in VMware Workstation

VMware Workstation 16.1 was used to set up 20 virtual hosts (referred to as workers) using Ubuntu 20.04 (Linux kernel 5.4.0-81-generic #91) with the only target of capturing packet captures from the scanner host. The host running the VMware Workstation is using Ubuntu Linux 20.04.4 (Linux kernel 5.13.0-35-generic #40). One virtual host (referred to as the scanner host) were installed with Kali Linux 2021.4 (Linux kernel 5.14.0-kali4-amd64 #1) for conducting a various type of scans, issuing tasks to the workers, and monitoring ongoing scans. Both these operating systems are based on Debian, which simplifies setup, configuration, and troubleshooting during the research. These operating systems are also easy to install and use without a large need for customization, which reduces the time consumption during the research as well.

3.2.2 Virtual network settings

Two closed internal virtual networks are set up for the purpose of data generation and management. These two networks had no connectivity to either the local network or to the internet. The network diagram for the lab environment is shown in figure 3.2.

The scanner host were also setup with an initial NAT network interface for retrieval of additional packets needed during the research and operating system updates primarily for the scanner host (as an initial step of the procedure while setting up the lab environment). The virtual network settings for this connection is shown in figure 3.3. This connection were mainly disabled during the research to

[1] https://www.kali.org/docs/installation/hard-disk-install/

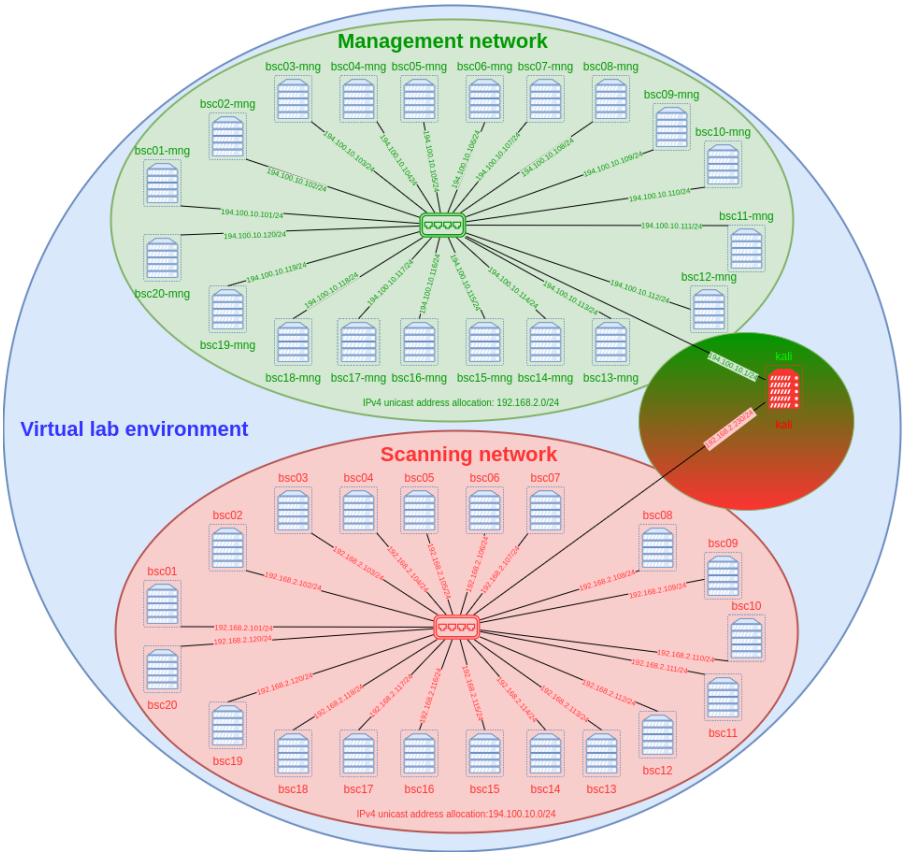


Figure 3.2: Capture of network diagram for the virtual environment.

not interfer with newer versions of software during the research which could result in various results making the generated data not comparable with earlier captured data.

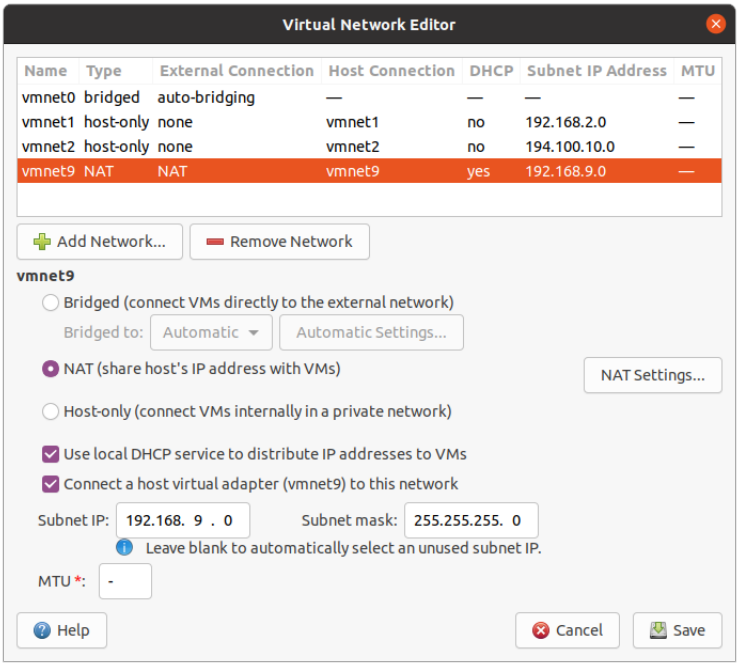


Figure 3.3: Virtual network settings for connectivity to internet.

This is for the production of raw packet captures in accordance with generating comparative data sets, which could be used for comparison between the generated synthetic data and data in the wild. Through such a comparison, these synthetic data sets could be used to classify various types of scanners and scanning types. The chosen IPv4 unicast address block for virtual network 1 (referred to as "scanning network") is $192.168.2.0/24$, as seen in figure 3.4. Though, the address blocks $192.0.2.0/24$ is the standard for TEST-NET-1 according to RFC 5737 (Arkko, Cotton, and Vegoda 2010), which describes the usage of the network during this research. The $192.168.2.0/24$ subnet was chosen, instead of following the RFC 5737 due to other networks within the local network using this exact subnet for another experiment, to minimize confusion during the research. Each worker host has its increment asset name where the hostname is in accordance with the last two digits in the last IP octet (e.g., $bsc01$ resolving to $192.168.2.101$). The scanning host has a higher last IP octet to ensure no confusion and IP collision regarding the static allocation for worker hosts. Within this research, the scanning host has the address $192.168.2.230$. This address allocation logic only shows flaws when the number of workers is above 99. As this is a proof of concept and the low number of workers, this is not a problem.

Secondly, another virtual network is set up for the usage of managing worker hosts in the network. The main reason is to reduce noise traffic on the scanning network and to segment management of each worker, such as managing tasks, monitoring ongoing tasks, and retrieving raw packet capture data. Chosen IPv4 unicast address block for the virtual network 2 (referred to as "management network") is $194.100.10.0/24$, settings shown in figure 3.5. Within RFC 1466 (Gerich 1993) a network in the address space $194.0.0.0$ - $195.255.255.255$ is allocated for management use in Europe. The same logic for address allocation used in the scanning network is also applied to the management network, where

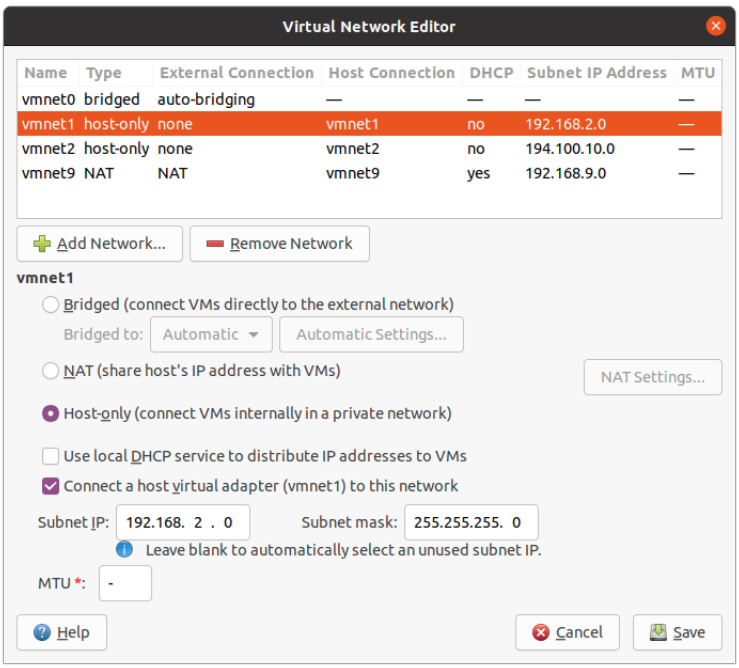


Figure 3.4: Virtual network settings for scanning network.

the last IP octet represents each worker host. Both these networks are visualized in 3.2. Another network characteristic is DHCP, which in both networks is disabled to ensure the logical structural setup of the network, such as static address allocation to each virtual host in accordance with a hostname. Basing the network setup on the RFC standards reduces the possibility of collision and confusion regarding usage of the networks (Gerich 1993; Arkko, Cotton, and Vegoda 2010).

3.3 Worker operating system configurations

During the installation and initial setup of a worker, multiple configuration changes need to be applied in order to decrease the amount of noise traffic. This noise traffic is primarily generated by the Ubuntu Linux packet managers APT and Snap, further elaborated in the section 3.3.1. Another noise generator is the network time protocol, described in section 3.3.2. The use of these packet managers and an updated time through NTP is not relevant to use during this research and are therefore disabled.

3.3.1 Disabling automatic upgrades and stop package managers

Default delivered in Ubuntu Linux 20.04 is the older APT (Advanced Package Tool) and the newer Snap package manager. Packages delivered through the SNAP PACKAGE MANAGER are called snaps. There exist multiple methods of package management in Ubuntu, such as using Synaptic Package Manager, Ubuntu Software, apt, apt-get, or snap. It is also possible to download raw deb files and install them using the pkg tool. In the Ubuntu Software in Ubuntu 20.04, Snap is prioritized when installing packages in front of the commonly used APT package manager. DEB (Debian package format) packages are the older format for packages in Ubuntu Linux, which APT uses. Snaps are formatted differently and cannot be installed through APT but instead installed through the Snap

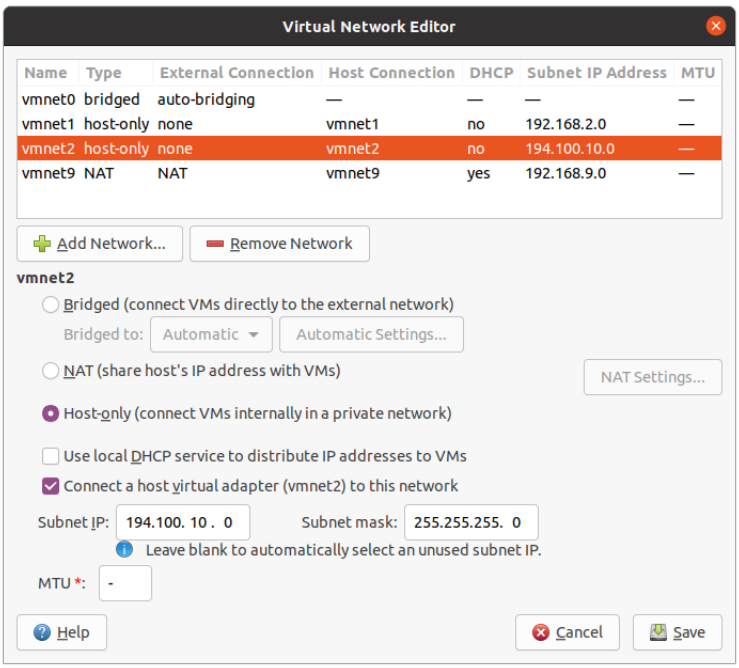


Figure 3.5: Virtual network settings for management network.

package manager. The Snap package manager uses snapd daemon for package management. The Snap package format will replace the older DEB package in Ubuntu (Petersen 2020).

Noise traffic generated by the automatic update polling for the package managers can be reduced. A few configuration parameter changes must be implemented to disable the automatic updates. Disabling automatic updates like this is a security risk for systems connected either to the internet or a local network. The workers are segmented on their own separated virtual network on one virtual host and not directly connected either to the internet or the local network. Therefore the risk during this research is acceptable. By changing the given parameters for the auto upgrades seen in listing 1 from 1 to 0 the automatic upgrades for the APT package manager are disabled.

```
# File contents for /etc/apt/apt.conf.d/20auto-upgrades
APT::Periodic::Update-Package-Lists "0";
APT::Periodic::Unattended-Upgrade "0";
```

Listing 1: Disabling automatic updates for APT

The snap package manager has a number of services running in Ubuntu 20.04 by default. These services can be disabled by running two commands for each service. The $systemctl\ stop$ command stops the running service, while $systemctl\ disable$ disables the service from running on startups. The commands in listing 2 must be run as $root$ on the worker host in the Bash shell command prompt.

By implementing these changes the automatic updates will be disabled both for the APT package manager and the Snap package manager, and it will reduce the noise traffic generated.

```
systemctl stop snapd
systemctl disable snapd
systemctl stop snapd.socket
systemctl disable snapd.socket
systemctl stop snapd.service
systemctl disable snapd.service
systemctl stop snapd.seeded
systemctl disable snapd.seeded
systemctl stop snapd.snap-repair.timer
systemctl disable snapd.snap-repair.timer
systemctl stop snapd.apparmor
systemctl disable snapd.apparmor
```

Listing 2: Disabling Snap services

3.3.2 Disabling NTP

Default in a Ubuntu distribution, like the 20.04 used in this research, a time and date service is active for the purpose of keeping the system updated with the correct date and time. The timedatectl would generate noise traffic such as NTP requests to ntp.ubuntu.com for synchronizing the time. During the initial setup, before noise reduction measurements were taken, a significant number of NTP requests were sent from the worker in order to reach ntp.ubuntu.com. The following command was executed to disable the network time synchronization[2].

```
timedatectl set-ntp 0
```

Listing 3: Command for disabling NTP synchronisation

This command stops the synchronisation services which enables the virtual machine to use the host's clock.

3.3.3 Enabling ordinary users to execute tcpdump

tcpdump requires higher privileges to capture incoming and outgoing traffic on the network interface. Since tcpdump is the program used for packet capturing in this research, it would be more efficient to give an ordinary user access to this program file and traffic capturing. The design of the capture is that the scanner uses SSH to issue tasks to each worker host. If tcpdump requires root access, this will lead to more complications in configurations on the worker host while issuing tasks to a worker. This issue is mitigated by setting the set-user-ID bit on the tcpdump program, enabling unprivileged users to capture traffic on the worker host's network interface, which normally requires privileged access.

Since the worker hosts are located on an isolated virtual network, the risk of unauthorized access is reduced and enabling simplistic capture of packets without any advanced configuration changes. These commands shown in listing 4 must be executed as $root$ on the worker to gain effect. The command in line one would return the full path for tcpdump, as seen in line two. Line three changes the modifications on the program, setting the set-user-ID bit. To assure that this is executed successfully, line four would list the permissions for the file shown inline five. Here we see that the s flag is set for the owner ($-rws$) and the group ($r-s$), while a normal user has read and execution permissions ($r-x$).

[2]https://www.man7.org/linux/man-pages/man1/timedatectl.1.html

```
which tcpdump
/usr/sbin/tcpdump
chmod +s /usr/sbin/tcpdump
ls -l /usr/sbin/tcpdump
-rwsr-sr-x 1 root root 1044232 Dec 31  2019 /usr/sbin/tcpdump
```

Listing 4: Command for changing mode on tcpdump file

After setting the set-user-ID bit on the file, the file rights looks similar like mentioned in the listing above.

3.3.4 Network configuration

Within the setup of the lab environment, each worker host must have its unique network configuration. For starters, a unique hostname needs to be set. By executing the command as $root$ in listing 5, the hostname for the worker host is set.

```
hostnamectl set-hostname <hostname>
```

Listing 5: Command for setting hostname

Within the research, the hostname convention used is $bscXY$, where the XY symbolizes an incrementing number starting on *01*.

Furthermore, the network configurations for each worker need to be set in the netplan configuration file shown in listing 6. Within this file, both network interfaces are static configured. One NIC is used for the scanning traffic, and the second NIC is used for management traffic. To retrieve the name for each of the NICs's, the command $ip\ a$ can be executed in the terminal. Result seen in listing 7.

The IP subnet for the management network is chosen in accordance with RFC 1466 (Gerich 1993). The IP address block for the scanning network is chosen partly in accordance with RFC 5737 (Arkko, Cotton, and Vegoda 2010), enlisted as $TEST - NET - 1$ in the RFC and within this paper as SCANNING NETWORK. Different from the RFC standard is the subnet $192.0.2.0/24$ given in the RFC 5737 is not chosen since it's already used for another project within the local network. To not create confusion, the $192.168.2.0/24$ subnet is chosen for this purpose, enlisted as SCANNING NETWORK.

Within each of the netplan configuration files, a unique IP address needs to be set for each of the workers, as seen in listing 6. To reload the network configuration, execute $netplan\ apply$ on the worker host as root.

These network settings can be verified after the settings are applied by running the command $ip\ a$ on the worker host, resulting in a similar output as seen in listing 7. Within listing 7 we see that the scanning interface ($ens33$) have the IP address 192.168.2.110, which is validated against the configuration in listing 6, while the management network is set to $ens36$ with the IP address 194.100.10.110.

In Linux, there is a file located in $/etc/hosts$, shown in listing 8, which is a local DNS file for looking up locally defined DNS records. On the scanning host for this research, this file contains the IP addresses of each of the workers IP addresses. The file has entries for both network interfaces, scanning network, and management network as described in section 3.3.4.

```
# File contents of /etc/netplan/00-installer-config.yaml
network:
  ethernets:
    ens33:
      addresses:
      - 192.168.2.110/24
      gateway4: 192.168.2.1
      nameservers:
        addresses:
        - 192.168.2.1
        search: []
    ens36:
      addresses:
      - 194.100.10.110/24
      gateway4: 194.100.10.1
      nameservers:
        addresses:
        - 194.100.10.1
        search: []
  version: 2
```

Listing 6: Network configuration for worker host

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:11:22:33:44:55 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.110/24 brd 192.168.2.255 scope global ens33
       valid_lft forever preferred_lft forever
3: ens36: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:22:33:aa:ff:44 brd ff:ff:ff:ff:ff:ff
    inet 194.100.10.110/24 brd 194.100.10.255 scope global ens36
       valid_lft forever preferred_lft forever
```

Listing 7: Listing of network interfaces

3.3.5 SSH configurations

By default the SSH service are listening on $0.0.0.0$, meaning all interfaces, as seen in the red marked two lines in figure 3.6. To achieve the objective regarding minimizing noise traffic to the scanning network interface, the SSH service used for task management must be set to listen to the correct network interface. The usage of SSH on the scanning network interface are not relevant and are therefore strictly limited to the management interface. The listen address parameter in the SSHd configuration file needs to be changed to the IP address for the management network interface, as seen in listing 9. This parameter change is conducted on each worker through opening the virtual machine in VMware Workstation. The $root$ user must conduct this, since the SSHd configuration file is a file owned by root. The IP address in line number 2 in listing 9 must be changed accordingly to the IP address assigned to the management network interface on the given worker host.

After this change is done, the SSH service must be restarted. This must be done by elevating the privileges to $root$ and execute the command in listing 10.

After the SSH server are restarted, the listen address must be validated. This could be done by executing the command in listing 11, and the result of this command is shown in figure 3.6.

```
127.0.0.1       localhost
127.0.1.1       kali
192.168.2.101   bsc01
192.168.2.102   bsc02
192.168.2.103   bsc03
192.168.2.104   bsc04
192.168.2.105   bsc05
192.168.2.106   bsc06
192.168.2.107   bsc07
192.168.2.108   bsc08
192.168.2.109   bsc09
192.168.2.110   bsc10
192.168.2.111   bsc11
192.168.2.112   bsc12
192.168.2.113   bsc13
192.168.2.114   bsc14
192.168.2.115   bsc15
192.168.2.116   bsc16
192.168.2.117   bsc17
192.168.2.118   bsc18
192.168.2.119   bsc19
192.168.2.120   bsc20
194.100.10.101   bsc01-mng
194.100.10.102   bsc02-mng
194.100.10.103   bsc03-mng
194.100.10.104   bsc04-mng
194.100.10.105   bsc05-mng
194.100.10.106   bsc06-mng
194.100.10.107   bsc07-mng
194.100.10.108   bsc08-mng
194.100.10.109   bsc09-mng
194.100.10.110   bsc10-mng
194.100.10.111   bsc11-mng
194.100.10.112   bsc12-mng
194.100.10.113   bsc13-mng
194.100.10.114   bsc14-mng
194.100.10.115   bsc15-mng
194.100.10.116   bsc16-mng
194.100.10.117   bsc17-mng
194.100.10.118   bsc18-mng
194.100.10.119   bsc19-mng
194.100.10.120   bsc20-mng
```

Listing 8: Contents of /etc/hosts on scanner host

```
1 # Changed parameters from default in file /etc/ssh/sshd_config
2 ListenAddress 194.100.10.110
```

Listing 9: Limiting SSH to listen only to the management NIC

```
systemctl restart ssh
```

Listing 10: Restart SSH server

```
ss -lntu
```

Listing 11: List of listening services

The last red marked line in figure 3.6 shows that the server only listens to incoming traffic on port 22 when the address is $194.100.10.130$. This means the SSH listening address is verified to only accept incoming connections on the mentioned address.

Furthermore, the key exchange between scanner and worker must be configured. As a first-time procedure, the key generation for the scanner host needs to be conducted in order to achieve automatic login to a worker without a password prompt. The command in listing 12 is executed on the scanner

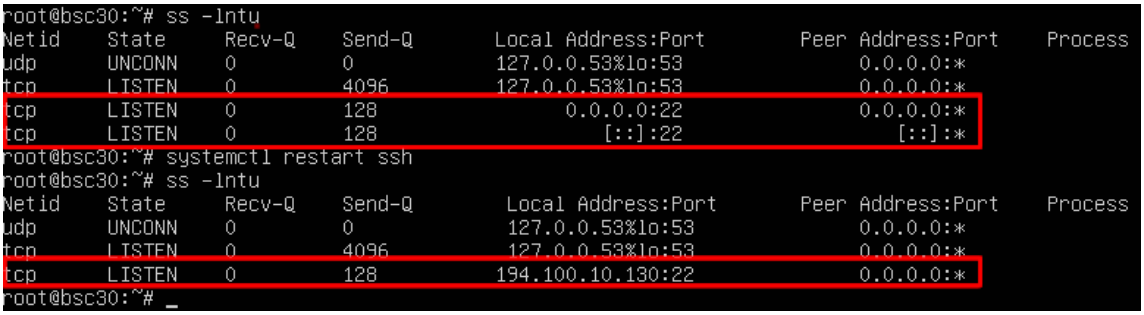


Figure 3.6: Verifying SSH configuration ListenAddress for management network.

host to generate a key pair with both private and public keys.

```
ssh-keygen -t rsa
```

Listing 12: SSH key generation

When the key pair is generated, a simple SSH client configuration needs to be implemented in order to achieve automatic login through SSH without receiving a password prompt. This configuration checks if the worker's hostname matches the criteria $bsc*-mng$, shown in line 2 in listing 13. In case of a match of the criteria, the configuration defines which default usernames are used to successfully connect to the worker host, shown in line 3 in listing 13. Also, the given private key to use for the connection is defined in line 4 in listing 13.

The configuration in listing 13 must be implemented to the file referred to in line 1 on the scanner host to achieve this.

```
1 # Implement the following lines to the file /root/.ssh/config
2 Host bsc*-mng
3     User bscadm
4     IdentityFile ~/.ssh/id_rsa
```

Listing 13: SSH client configuration for scanner host

When these settings are implemented, the public key needs to be authorized on the worker host. This is step two of the process of enabling authorization using a public key instead of a password prompt.

```
1 ssh-copy-id -i ~/.ssh/id_rsa bscadm@bsc30-mng
2 ssh bsc30-mng
```

Listing 14: SSH key authorization on worker host

The command in line 1 in listing 14 adds the key to the authorized keys list in the home directory for the user $bscadm$, as defined in line 3 in listing 13. Secondly, in line 2 in listing 14, the connection is tested. The first time logging in on this worker would trigger a prompt for key authorization, which must be accepted before continuing. From this point forward, the ability to automatically log in to a worker using public-key authorization is enabled.

3.3.6 Lab environment

Through an initiation worker preparation script, there are implemented commands to reduce traffic noise affecting the scan results. This script disables certain services known for querying servers on the internet for the purpose of updating time through NTP, automatic operating system updates, and operating system package updates. This script must be run during the setup of the environment to ensure noise reduction. Another very important step during this initial configuration is change moving the tcpdump command to suid ($chmod + s$), meaning a regular user is able to run tcpdump as a regular user. The command must be executed by root. During the initial configuration of each worker, the SSH public key for the scanner host is added to the worker's authorized keys in $\$HOME/.ssh/authorized\_keys$. It allows the scanner host to remotely access each worker with the use of its SSH private key to manage tasks for the worker through SSH. Each worker has its own $bscadm$ local user for this purpose. SSH listens only to the management network interface on each worker.

3.4 Developed scripts

For the purpose of automating the issuing of tasks to each worker, automatic collection of packet captures, and retrieval of all packet captures from each worker, code had to be produced to achieve this goal. The scripts developed in this research are further described in each respective subsection in this section.

3.4.1 Scanner

Given task files are iterated until a task with the status new is found. From this point, the scanner host deploys a task to the worker through the management network, commanding it to dump traffic on the incoming interface pointed to the scanning network. When this task is successfully deployed, the scanner hosts start the scan on the scanning network towards the worker with the given scanning characteristics. As a default in this script, using the Nmap scanner, the result of the scan from the scanner host is outputted to an XML result file represented with the given task name. This XML file is not used for analysis within this research, though the generated XML file could be used in future work, further described in section 6.5. The status for the representative task in the task file is changed from new to $ongoing$. This operation is iterated through all workers until all workers are busy when the scanning host cannot deploy more tasks until one or more of the workers are available for work again. A capture of this is shown in figure 3.7.

The scanner script is built up with a number of functions for later use within the script.

Standardising the message output

When the script is running from the prompt by a user, it is important that the output is human-readable. The function in listing å15 colorizes the messages depending on error messages, success messages, or ordinary messages. A visualization of this output is shown in figure 3.7. The figure also shows the message output when conducting task allocation, further elaborated in section 3.4.1.

```
colored_message() {
  COLOR=$1
  MESSAGE="$2"
  if [[ $COLOR == "red" ]]; then
    echo -e "[ \e[31m\e[1mError\e[0m ] $MESSAGE"
  elif [[ $COLOR == "green" ]]; then
    echo -e "[ \e[32m\e[1mSuccess\e[0m ] $MESSAGE"
  else
    echo -e "$MESSAGE"
  fi
}
```

Listing 15: Colorizing messages

```
bsc07-mng is busy (error code: 1).
[ Success ] bsc08-mng is available.
[ Success ] Capturing traffic on bsc08-mng on task nmap_fin_scan_insane
tcpdump: listening on ens33, link-type EN10MB (Ethernet), capture size 262144 bytes
[ Success ] Scanning bsc08-mng on task nmap_fin_scan_insane
[ Success ] bsc09-mng is available.
[ Success ] Capturing traffic on bsc09-mng on task nmap_null_scan_insane
tcpdump: listening on ens33, link-type EN10MB (Ethernet), capture size 262144 bytes
[ Success ] Scanning bsc09-mng on task nmap_null_scan_insane
[ Success ] bsc10-mng is available.
```

Figure 3.7: Capture of task allocation and a standardized colorized message format.

Checking task status

The input for this function is the unique task name. The function would then retrieve the process information for the given task name. It will then check if the indicated process ID on the scanner machine exists, and the result would be returned in the format where *1* symbolize that the task is still in progress, while *0* would identify a task complete.

```
check_scan_status() {
  TASK_NAME=$1
  SCANNER_PROCESS_INFO=$(ps aux | grep "${RESULT_DIR}/${TASK_NAME}" | grep -v grep)
  SCANNER_LOCAL_PID=$(echo $SCANNER_PROCESS_INFO | awk '{ print $2 }')

  if [[ ! -z $SCANNER_LOCAL_PID ]]; then
    return 1 # return false - still working
  else
    return 0 # return true - work done
  fi
}
```

Listing 16: Checking scan status

As seen in listing 16 the if statement checks if a process ID is set with the statement that the process ID should not (using !) be a zero value (z). If the process ID is not set, it will return an error code 1 ($false$ in this case), while the error code 0 ($true$ in this case) is returned if the process ID is zero. This goes against the general logic in programming, where usually 1 is $true$ and 0 is $false$. The reason for returning 1 in Bash is that it returns the error code given from the script. A 0 error code indicates no errors and will therefore be true, while any other positive error codes would indicate an error. This could be not only applied for the Bash shell but also in system error codes used in Windows, where 0 ($0x0$) is a successful operation (Microsoft 2021).

Terminate a given task

The input for this function is the unique task name. The function would retrieve the process information for the given task name. If the given process ID is found, the script will continue to retrieve the local PID on the scanner host and retrieve the remote process information running on the respective worker host. The local process on the scanner host would be killed, and the remote process ID's allocated to the tcpdump running with the unique task name would be gracefully killed as well.

```
terminate_tcpdump() {
  TASK_NAME=$1
  PROCESS_INFO=$(ps aux | grep -E "(tcpdump).*($TASK_NAME)" | grep -v grep)
  if [[ ! -z $PROCESS_INFO ]]; then
    LOCAL_PID=$(echo $PROCESS_INFO | awk '{ print $2 }')
    WORKER_HOST=$(echo $PROCESS_INFO | awk -F 'ssh' '{ print $2 }' | \
    awk '{ print $1 }')
    REMOTE_PIDS=$(ssh $WORKER_HOST ps aux | grep -E "(tcpdump).*($TASK_NAME)" | \
    awk '{ print $2 }')
    REMOTE_PIDS_VIEW=$(echo $REMOTE_PIDS | tr "\n" " ")

    kill -15 $LOCAL_PID ; ssh $WORKER_HOST kill -15 $REMOTE_PIDS_VIEW
    STATUS=$?
    return $STATUS
  fi
}
```

Listing 17: Terminate tcpdump

After the process is gracefully killed, the function would return the status of the executed command for use where the function is executed in the script.

Availability check

The primary mission of this function is to determine if a worker is busy or available for work. The input for this function is the given hostname, which will be collated to the tcpdump process on the respective worker. There exist multiple status checks for the worker. Mainly the function returns three various types of status; unreachable, busy, or available. Since the scanner is attempting to SSH into the given worker host, and no reply is given, an unreachable status is returned from the function. If SSH is successful and a process with tcpdump is running on the given worker, the busy status is returned. If none of these checks are true, the status available would be returned.

```
available_worker() {
  HOSTNAME=$1
  PID_TCPDUMP=$(ssh -q $HOSTNAME pgrep tcpdump)
  if ! ssh -q -T $HOSTNAME exit &> /dev/null; then
    return 2 # unreachable
  elif [[ ! -z $PID_TCPDUMP ]]; then
    return 1 # busy
  else
    return 0 # available
  fi
}
```

Listing 18: Retrieving available workers

Task status change

Since the scanner script iterates through the task list, a function for changing the given task status was crucial to integrate for task management. There exist four different inputs to this function; unique task name, priority, old status, and new status. This function would generate an old and new CSV formatted entry. This format is later used to replace the given old string in the CSV task file with the new string. When successfully completed, a success return is given, while an error return is given if the command fails.

```
task_change() {
  TASK_NAME=$1
  PRIORITY=$2
  OLD_STATUS=$3
  NEW_STATUS=$4

  OLD_CSV_ENTRY="$PRIORITY,$TASK_NAME,$OLD_STATUS"
  NEW_CSV_ENTRY="$PRIORITY,$TASK_NAME,$NEW_STATUS"

  if sed -i -e "s/$OLD_CSV_ENTRY/$NEW_CSV_ENTRY/g" $TASK_FILE; then
    return 0 # true (success)
  else
    return 1 # false (error)
  fi
}
```

Listing 19: Changing task status

Find interface name and start tcpdump

The need to know which interface tcpdump should listen to is solved through a function. Input to this function is the given worker hostname. The function logs into the worker and extracts the interface name from the given subnet.

```
find_listening_scanner_if() {
  WORKER_HOST=$1
  IF=$(ssh $WORKER_HOST ip r | grep $SUBNET | awk '{print $3}')
  echo $IF
}
```

Listing 20: Find scanning network interface on worker host

With the knowledge of the network interface name on the worker, tcpdump could be started. Into this function are the worker's hostname and the output packet capture filename. This function starts the tcpdump on the given worker through SSH with the output to the given packet capture file. It returns then the status for the task, where '1' is returned as successfully started or '0' if the command failed to execute. Within the start of the script, a filter parameter is given as a default filter for all the conducted scans, denoted in the first line below.

```
TCPDUMP_FILTER="ip and dst not 192.168.2.1 and src not 192.168.2.1" # no GW tfc

start_tcpdump() {
  WORKER_HOST=$1
  PCAP_FILENAME=$2
```

```
  INTERFACE=$(find_listening_scanner_if $WORKER_HOST)

  if ssh $WORKER_HOST "tcpdump -U -i $INTERFACE -w $PCAP_FILENAME $TCPDUMP_FILTER \
  2>&1" & > /dev/null; then
    return 0 # success!
  else
    return 1
  fi
}
```

Scanner function

The scan itself is conducted within this function. Inputs to this function are which scanner is to be used (e.g., Nmap), the worker's hostname, the output path for the XML generated from the scanner, and additional arguments used for the scan. The function starts the scanner and returns an error code to symbolize successfully started or failed to start.

```
scan() {
  SCANNER=$1
  TARGET_HOST=$2
  OUTPUT_PATH=$3
  SCANNER_ARGS="$4" # must do something like £{4.<endline>}
  SCANNER_ARGS=$(echo $SCANNER_ARGS | sed 's/"//g')

  # Nmap
  if [ $SCANNER == "nmap" ]; then
    nmap -oX $OUTPUT_PATH.xml $TARGET_HOST $SCANNER_ARGS --system-dns 2>&1 \
    > /dev/null &
    ERR_CODE=$?
    return $ERR_CODE
  fi
}
```

Listing 21: Scanning function

Task list update

This function iterates through the task list, reading the values for each task from a CSV task file. It checks if a valid task name is given and detects if a header is found. The status for the given task name is checked through the task status check function 3.4.1. If the status '0' is returned, which symbolizes work done, this function would terminate tcpdump, as seen in listing 17.

Deploying tasks to a worker

Within this function of the scanner script, the deployment of tasks is conducted to a given worker. This function requires the input of a given worker host. Furthermore, it iterates through the task list and determines which tasks have the status new. Tcpdump is then attempted to be started on the given worker. When tcpdump is successfully started on the worker, it starts the scanning towards the given worker. The status in the task list is changed from new to ongoing. If the steps of either starting tcpdump or scan fail, an error message is given. When a task is deployed successfully, the iteration

```
update_tasklist() {
  # Read through the tasks in the task file
  while IFS=, read -r PRIORITY TASK_NAME TASK_STATUS SCANNER EXTRA_ARGS
  do
    # Make sure that the task is not the header in the task file
    if [[ ! -z $TASK_NAME ]] && [[ $TASK_NAME != "task_name" ]]; then
      check_scan_status $TASK_NAME
      SCAN_STATUS=$?

      if [[ $SCAN_STATUS -eq 0 ]]; then
        if terminate_tcpdump $TASK_NAME; then
          task_change $TASK_NAME $PRIORITY "ongoing" "completed"
        else
          colored_message "red" \
          "Unable to terminate traffic capture on task \e[1m${TASK_NAME}\e[0m."
        fi
      fi
    fi
  done < $TASK_FILE
}
```

Listing 22: Update tasklist and terminate TCPDump when scan is complete

will break and stop.

Crawl through workers, check availability and deploy tasks

Within this final step of the script, the worker host array defined at the start of the script would be iterated. This iteration would check the current status of the given worker returning the status of busy, unreachable, or available. If the available status is retrieved, the task is deployed to the given worker through the deploy task function described in section 3.4.1.

3.4.2 Task manager

The main objective of this script is to maintain the tasklist, changing the status of tasks from ongoing to completed. This script is executed through the terminal as a user and is not designed to run through a cronjob. The main input to this script is the given task file.

Process comparison

This function extracts the executed command for a process matching tcpdump and pipes the output to a temporary file. The temporary file is iterated through, and the PID for each running tcpdump is retrieved. From here, the function compares the task name to processes initiated through SSH to each worker. The function then checks if there is a scanning process towards the given work with the correlating task name. If there is no PID returned from this check, the function terminates the tcpdump on the worker.

Reused functions from the scanner script

Compared to the scanner script, functions have been reused within this script. These are important functions for colorizing messages, checking scan status, terminating tcpdump, changing task status,

```
deploy_tasks_to_worker() {

  # Define worker host
  WORKER_HOST=$1

  # Read through the tasks in the task file
#    update_tasklist

  while IFS=, read -r PRIORITY TASK_NAME TASK_STATUS SCANNER EXTRA_ARGS
  do
    # Make sure that the task is not the header in the task file
    if [[ ! -z $TASK_NAME ]] && [[ $TASK_NAME != "task_name" ]]; then

      # Do not scan the management side
      TARGET_HOST=$(echo $WORKER_HOST | sed 's/-mng//g')
      # --------------------
      # Task status: NEW - deploy tasks to worker
      # --------------------
      if [[ $TASK_STATUS == "new" ]]; then
        TIMESTAMP=$(date +%Y%m%d%H%M)
        OUTPUT_PATH="${RESULT_DIR}/${TASK_NAME}_${TIMESTAMP}"
        # Start dumping traffic on a worker
        if start_tcpdump $WORKER_HOST "${TASK_NAME}_${TIMESTAMP}.pcap"; then
          colored_message "green" "Capturing traffic on \
          \e[1m${WORKER_HOST}\e[0m on task \e[1m${TASK_NAME}\e[0m"

          # Start scan
          sleep 3 # make sure tcpdump is spawned
          if scan $SCANNER $TARGET_HOST $OUTPUT_PATH "$EXTRA_ARGS"; then
            colored_message "green" \
            "Scanning \e[1m${WORKER_HOST}\e[0m on task \e[1m${TASK_NAME}\e[0m"

            # Change the task status
            task_change $TASK_NAME $PRIORITY "new" "ongoing"
          else
            colored_message "red" \
            "Unable to scan \e[1m${TARGET_HOST}\e[0m on task \e[1m${TASK_NAME}\e[0m"
          fi
        else
          colored_message "red" "Unable to capture traffic on \
          \e[1m${WORKER_HOST}\e[0m on task \e[1m${TASK_NAME}\e[0m"
          break
        fi

        echo "" > /dev/null # for the fun of it
        break # break out since the worker already have received a task
      fi
    fi
  done < $TASK_FILE
  sleep 5
}
```

Listing 23: Deploy tasks to a worker host

and updating task list. These functions are described within the scanner section 3.4.1.

```
for WORKER_HOST in ${WORKER_HOSTS[@]};
do

  # Check if the host is available for work
  available_worker $WORKER_HOST
  ERR_CODE=$?

  # available for work
  if [[ $ERR_CODE -eq 0 ]]; then
    colored_message "green" "$WORKER_HOST is available."
    deploy_tasks_to_worker $WORKER_HOST


  # unreachable
  elif [[ $ERR_CODE -eq 2 ]]; then
    colored_message "red" "$WORKER_HOST is unreachable (error code: $ERR_CODE)"

  # unavailable (working on something)
  elif [[ $ERR_CODE -eq 1 ]]; then
    echo "$WORKER_HOST is busy (error code: $ERR_CODE)."
  fi
done
```

Listing 24: Worker iteration and task deployment

```
process_comparision_clean() {
  YEAR=$(date +%Y)
  ps -eo command | grep tcpdump | grep -v grep | sed 's/^ssh.*-w //g' | \
  sed "s/_$YEAR.*$//g" > ps_comparision

  while read TASK_NAME
  do
    SCANNER_PID=$(ps -eo pid,command | grep "$TASK_NAME" | grep -Ev 'tcpdump|grep' | \
    awk '{ print $1 }')
    if [[ -z $SCANNER_PID ]]; then
      terminate_tcpdump $TASK_NAME
    fi

  done < ps_comparision
}
```

Listing 25: Process comparison and TCPDump termination

Further on, this script contains a true while loop for continuously running checks to update the tasks. The reason behind this is to reduce the amount of irrelevant traffic generated after a scan is completed.

3.4.3 Task population

A script was developed for populating a unique tasklist for streamlining the process of creating tasks. The script reads one input parameter, which is a template task file created by a user. This input file is then iterated through, and an incrementing number is added to each task for the identification of unique tasks later during a scan.

To populate a new task list file, the following command needs to be executed in a Bash shell.

```
while true
do
  if update_tasklist; then
    DATE=$(date "+%d.%m.%Y %H:%M:%S")
    printf "\r[ $DATE ] Cleaned taskfile: $TASK_FILE"
    sleep 2
  else
    printf "\r[ $DATE ] Failed cleaning taskfile: $TASK_FILE"
  fi
done
```

Listing 26: Update tasklist continously

```
#!/bin/bash
INPUT_FILE=$1

if [[ -z $INPUT_FILE ]]; then
  echo "Usage: $0 <task-file>"
  echo "Default number of task param is \"50\" - must be changed in for loop in line 12"
  exit
fi

while IFS=, read -r PRIORITY TASK_NAME TASK_STATUS SCANNER EXTRA_ARGS
do
  for i in {1..50}
  do
    echo "${PRIORITY},${TASK_NAME}_${i},${TASK_STATUS},${SCANNER},${EXTRA_ARGS}"
  done
done < $INPUT_FILE
```

Listing 27: Populating tasks to task file

```
user@host:~# bash populate.sh template-taskfile.csv >> new-taskfile.csv
```

This will append each line generated to the ‘new-taskfile.csv‘.

3.4.4 Packet capture retrieval

This script is created for easier retrieval of packet captures from each worker host. During the initial research, packet captures, including noise traffic, were generated. Within this script, a filter is applied to reduce the retrieval of these packet captures. During the research, this filter mainly consisted of a timestamp starting with year and month to retrieve the correct files. This script uses rsync to retrieve packet captures from each worker host.

3.4.5 Task management scripts

A task processing script is running to maintain the task list with the correct given status of tasks. This meaning is only comparing running scan processes, identified by task name on the scanning host, against running tcpdump tasks on the given worker. When this differs, and only the tcpdump process together with the task name are found in the process list, it terminates the tcpdump on the worker and updates the task status in the task list.

A monitor script is also created to monitor the ongoing tasks. This monitor updates every 2 seconds,

```
#!/bin/bash
WORKER_HOSTS=(bsc01-mng bsc02-mng bsc03-mng bsc04-mng bsc05-mng bsc06-mng bsc07-mng
bsc08-mng bsc09-mng bsc10-mng bsc11-mng bsc12-mng bsc13-mng bsc14-mng bsc15-mng
bsc16-mng bsc17-mng bsc18-mng bsc19-mng bsc20-mng)
PCAP_FILTER_NAME=$1
OUTPUT_DIRECTORY=$2

if [[ -z $PCAP_FILTER_NAME ]] || [[ -z $OUTPUT_DIRECTORY ]]; then
  echo "Usage: $0 <pcap file filter name> <sync destination>"
  exit
fi

for WORKER_HOST in ${WORKER_HOSTS[@]};
do
  rsync -azvv -e ssh "bscadm@$WORKER_HOST:$PCAP_FILTER_NAME*.pcap" $OUTPUT_DIRECTORY/
done
```

Listing 28: Retrieving packet captures from worker hosts

capturing scanning specific tasks and tcpdump tasks returning data for worker corresponding with task name, shown in figure 3.8.

Worker_Host	Process_Name	Task
bsc01	nmap	nmap_fin_scan_paranoid
bsc01-mng	tcpdump	nmap_fin_scan_paranoid
bsc02	nmap	nmap_null_scan_paranoid
bsc02-mng	tcpdump	nmap_null_scan_paranoid
bsc03	nmap	nmap_os_svc_discovery_paranoid
bsc03-mng	tcpdump	nmap_os_svc_discovery_paranoid
bsc04	nmap	nmap_ping_echo_scan_paranoid_1
bsc04-mng	tcpdump	nmap_ping_echo_scan_paranoid_1
bsc05	nmap	nmap_svc_discovery_sneaky_7
bsc05-mng	tcpdump	nmap_svc_discovery_sneaky_7
bsc06	nmap	nmap_svc_discovery_paranoid
bsc06-mng	tcpdump	nmap_svc_discovery_paranoid
bsc07	nmap	nmap_tcp_full_scan_paranoid
bsc07-mng	tcpdump	nmap_tcp_full_scan_paranoid

Figure 3.8: Capture of scan monitoring tool

Other useful tools are developed, such as the TASK POPULATOR. The populator iterates through a given task file containing tasks that want to be run a number of times in order to generate a comparable synthetic data set in the end. An incrementing number is added to the task name (e.g., $nmap\_xmas\_scan\_1$) to uniquely identify a task when iterated through by the scan deployment stage.

A simplistic collection of captured packet captures run through a retrieve script on the scanning host, which iterates through each worker host matching a specific filter (e.g., task name) and remotely synchronizes packet captures to the local output directory on the scanning host.

3.5 Data files archived and stored

A naming convention for files and directories was used to separate each scan type from the other and each number of a scan. In figure 3.9 the structure of naming convention for files are shown. The red color marks the task name, the blue marks the scan number, and the green contains the timestamp for the scan.

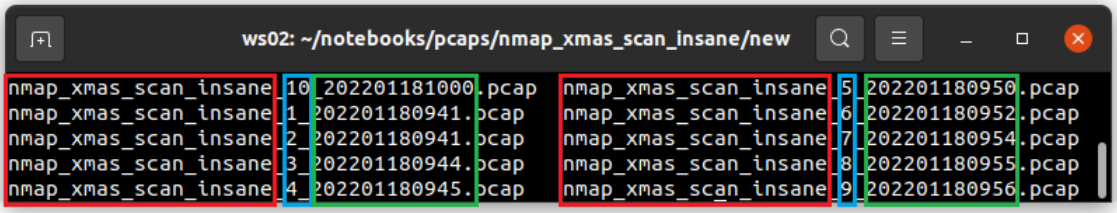


Figure 3.9: File naming convention.

The directory structure shown in figure 3.10 was created manually on the host running VMware Workstation, since the analysis using Jupyter was conducted on this host. This was done mainly to separate each type of scan into respective subdirectories to prevent mixups of scan files and to keep a clean structure when conducting analysis.

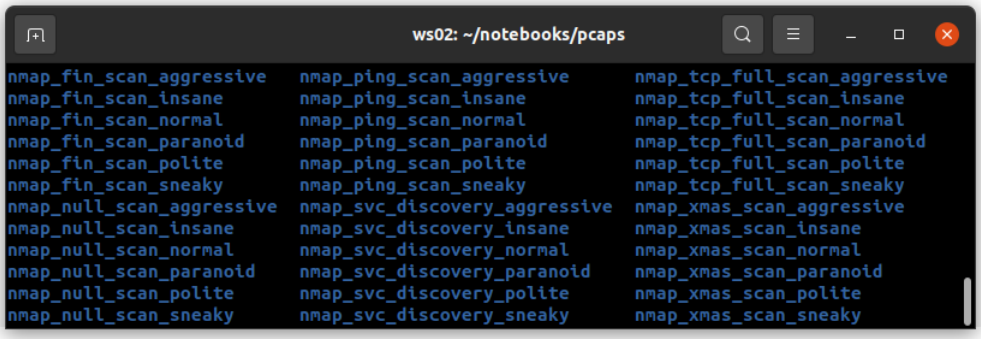


Figure 3.10: Directory naming convention.

The file listings on each of the worker hosts have no functionality to consider and sort the packet captures into respective subdirectories. Each worker only saves the packet capture into their home directory with the naming convention shown in figure 3.9.

When the task list for all scans was completed, the PCAP RETRIEVAL, further elaborated in section 3.4.4 were executed manually. All packet captures from the worker hosts were now transferred to the scanner host. The packet captures retrieved could from here be retrieved through SSH from the host running VMware Workstation. Before this retrieval can find place, the packet captures must be moved from the $root$ home to the $kali$ user's home directory, and the owner for directories and files must be changed to the $kali$ user before being able to retrieve these files. These steps are described in lines 2 and 3 in listing 29.

Within this research, the VMware host OS is Ubuntu Linux, which makes the transfer process of these files simple, and only the command in line 11 was executed to retrieve all packet captures.

```
# The following commands must be executed as root on the scanner host
mv <directory where the packet captures are synched to> /home/kali/
chown -hR kali:kali /home/kali/<directory name mentioned above>

# If the VMware Workstation host is running Linux, the following commmand
# could be executed to retrieve all packet captures.
# This command must be executed at the VMware host machine
rsync -azvv -e ssh kali@<ip to scanner host>:<directory containing pcaps> \
<destination on scanner host>
# Example:
rsync -azvv -e ssh kali@192.168.2.230:pcap-results pcaps/
```

Listing 29: Limiting SSH to listen only to the management NIC

Alternative tools for retrieving packet captures from the scanner host if the VMware host OS were Windows, WinSCP[3] and Filezilla[4] are decent tools to use for this purpose. WinSCP has an easy GUI for transferring files, shown in figure 3.11.

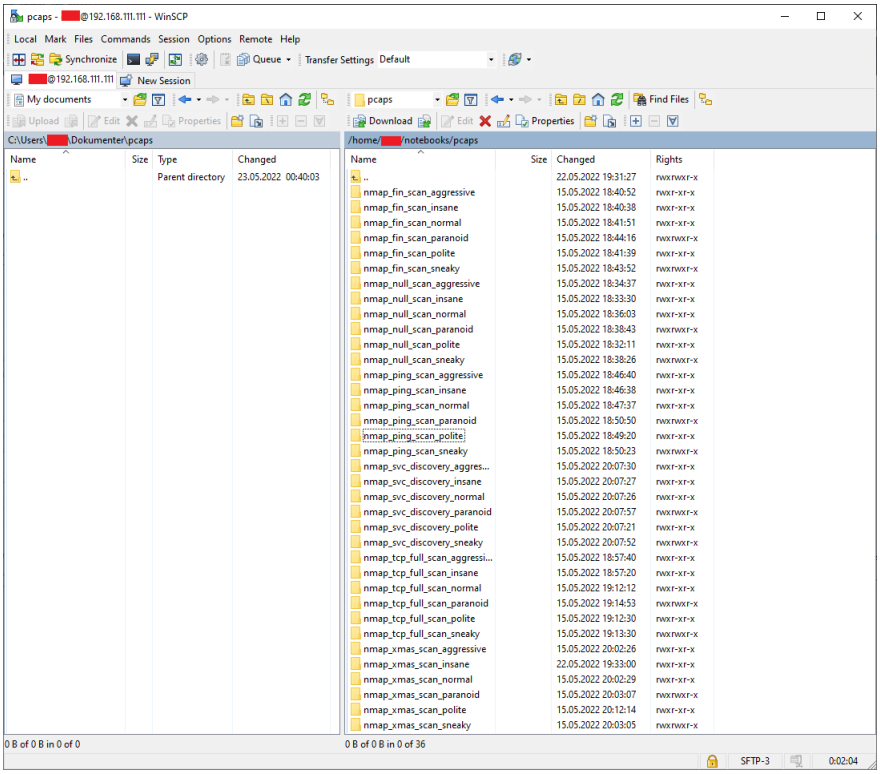


Figure 3.11: WinSCP GUI

3.6 Summary

To conduct this research without setting up a lab environment and automating these types of scans would not be a beneficial design choice considering time management due to the fact that these scans then had to be conducted manually. The amount of various types of scans within this research is the following six (m), each described in its respective section:

[3] https://winscp.net/
[4] https://filezilla-project.org/

- Section 2.16.1: Port and Service scan
- Section 2.16.2: Host Discovery using ICMP echo requests
- Section 2.16.3: XMAS scan
- Section 2.16.3: Null scan
- Section 2.16.3: FIN scan
- Section 2.16.4: Connect scan

As described in section 2.15, it exists six different timing templates (t). Each of the different scanning types with the respective timing template has ten scans (n). This means the total sum (s) of packet captures would be calculated as follows:

$s = (m * t) * n$

This results in *360* packet captures in total. Since each Nmap scan would use different times compared to each other, a user executing these scans manually would use a significant amount of time conducting this research manually. This would require the user to regularly check the scanning status and start a new scan depending on the status of the scan. Other factors such as human error would increase by manually scanning as well. Depending on the intervals the researcher checks the scan status, the time conducting this research would be significantly higher compared with an automated scanning framework like this lab environment and scripts included.

This framework makes use of the workers immediately when they're done with the last task, streamlining the scanning process significantly. Development of scripts to conduct scanning, cleaning processes, and conducting task management were made to automate this process. These scripts are described in detail in section 3.4.

4

Data Analysis

4.1 Introduction

In this chapter, the generation of synthetic packet captures, together with the development and usage of the Jupyter analysis notebook, is elaborated.

The importance of automatically generating synthetic packet captures is stated in this chapter. Having a standard of conducting each network scan enhances the reliability of each scan being conducted in a similar standardized way compared to manually running these Nmap scans, where differences might appear depending on the user inputs while executing the commands and human error. In cases where this is conducted manually, the risk of not standardizing the scans can lead to incomparable data as an output during the analysis phase. Therefore, the importance of automating the scans with a standard is crucial. The automation is also time-saving compared to a manual scanning procedure. By the use of various timing templates, the workers finish the scans at different times, which makes a manual scanning method very time-consuming and not effective. This design and implementation chapter will describe the setup of such an automation packet capture generation.

A virtual lab environment was needed to conduct this research, with the purpose of segmenting captured scanning traffic from real internal network traffic to ensure the validity of the synthetic generated packet capture data, meanwhile having the capability of managing each worker host. The importance of generating synthetic data is to have a clean data set, with minimal noise, for comparison against real-world data containing a substantial amount of noise traffic, as seen in figure 4.1.

```
05:40:35.270969 IP 192.168.2.118.36024 > ws02.domain: 18368+ A? api.snapcraft.io. (34)
05:40:35.272581 IP 192.168.2.118.53962 > ws02.domain: 31480+ AAAA? api.snapcraft.io. (34)
05:40:39.279036 IP 192.168.2.118.42495 > ws02.domain: 45912+ AAAA? ntp.ubuntu.com. (32)
05:40:39.279557 IP 192.168.2.118.40442 > ws02.domain: 7634+ A? ntp.ubuntu.com. (32)
05:40:39.283978 IP 192.168.2.118.49600 > ws02.domain: 7634+ A? ntp.ubuntu.com. (32)
05:40:39.284360 IP 192.168.2.118.40698 > ws02.domain: 45912+ AAAA? ntp.ubuntu.com. (32)
05:40:39.284640 IP 192.168.2.118.49600 > ws02.domain: 7634+ A? ntp.ubuntu.com. (32)
05:40:39.284830 IP 192.168.2.118.40698 > ws02.domain: 45912+ AAAA? ntp.ubuntu.com. (32)
05:40:39.286010 IP 192.168.2.118.57192 > ws02.domain: 53498+ A? ntp.ubuntu.com. (32)
05:40:39.286299 IP 192.168.2.118.57338 > ws02.domain: 24806+ AAAA? ntp.ubuntu.com. (32)
05:40:40.282046 IP 192.168.2.118.43407 > ws02.domain: 31480+ AAAA? api.snapcraft.io. (34)
05:40:40.282245 IP 192.168.2.118.39693 > ws02.domain: 18368+ A? api.snapcraft.io. (34)
05:40:40.282333 IP 192.168.2.118.43407 > ws02.domain: 31480+ AAAA? api.snapcraft.io. (34)
05:40:44.289648 IP 192.168.2.118.57338 > ws02.domain: 24806+ AAAA? ntp.ubuntu.com. (32)
05:40:44.293813 IP 192.168.2.118.48717 > ws02.domain: 24806+ AAAA? ntp.ubuntu.com. (32)
05:40:44.293957 IP 192.168.2.118.40430 > ws02.domain: 53498+ A? ntp.ubuntu.com. (32)
05:40:44.294090 IP 192.168.2.118.48717 > ws02.domain: 24806+ AAAA? ntp.ubuntu.com. (32)
05:40:44.294202 IP 192.168.2.118.40430 > ws02.domain: 53498+ A? ntp.ubuntu.com. (32)
```

Figure 4.1: Capture of noise in packet capture.

4.2 Generating and collecting synthetic data

Management of each worker is conducted from the scanner host, as this is the host initiating the scanning tasks. The task issuing is conducted through a Bash script described in section 3.4.1.

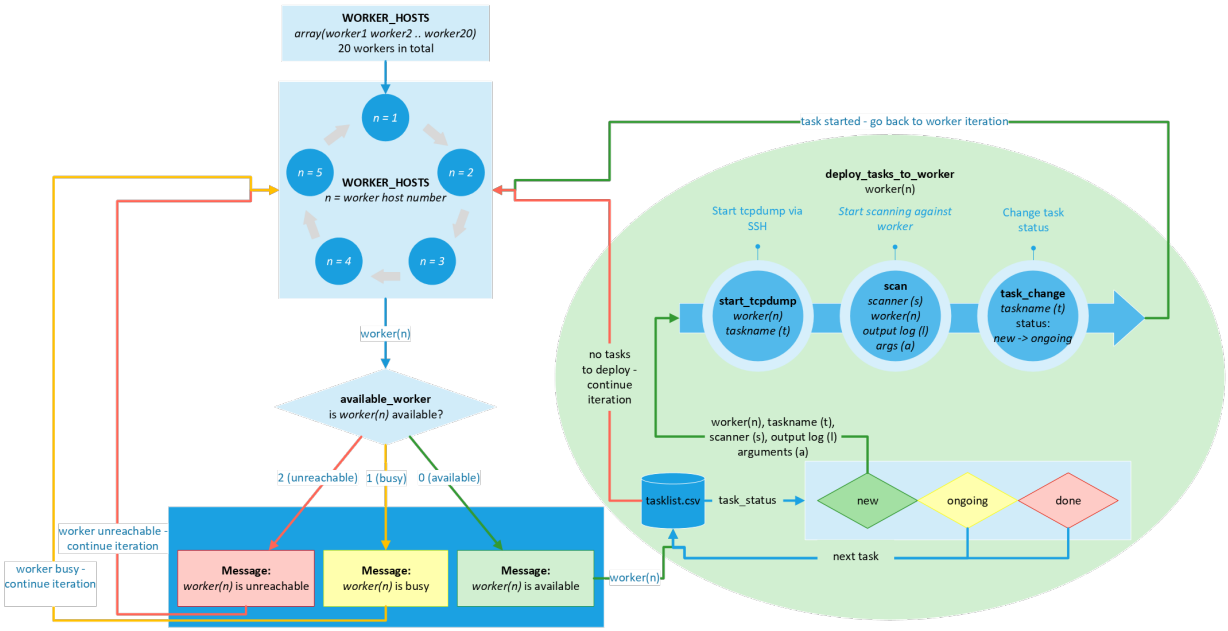


Figure 4.2: Scanning environment flow chart

As seen in figure 4.2 an array containing all the worker's hostname are enlisted as $WORKER\_HOSTS$. This is an input to the loop seen below, also marked as $WORKER\_HOSTS$, and n is defined as the identification number for each worker host. The array is iterated, starting on $worker1$ where the function $available\_worker$ checks if the given worker is available. An error code is returned as the status of the worker. If the worker is unreachable, code 2 is returned, and a message is printed saying the worker is unreachable. If the worker is busy, code 1 is returned, and a message is printed saying the worker is busy. With the given error code 1 or 2, the iteration in the loop is continued with an increment of the number with $+1$, and the mentioned process is repeated with the worker now set to $worker2$. When the $available\_worker$ function finds an available worker, it returns a 0 error code and returns a message informing the availability of a worker. Now a new iteration in function

$deploy\_tasks\_to\_worker$, code shown in listing 23, are started. It iterates through each line in the tasklist, checking the status of the given task. If the given task is either *ongoing* or *done* it continues the iteration until it finds a task with the status *new*. If a task is found with the status *new* the script starts tcpdump on the worker with the given task name. Then it starts the scan on the scanning host towards the given work with the scanner arguments. Further it changes the status for the task from *new* to *ongoing*. The iteration of workers now continues until it reaches the last worker, which in this research is $worker20$. During the research, this script has been run as a cronjob with the entry shown in 30. This cronjob is configured to run each 3rd minute. To edit a cronjob, run the command $crontab -e$ as root on the scanner host.

```
*/3 * * * * /bin/bash /root/scan.sh tasklist.csv > /dev/null 2>&1
```

Listing 30: Cronjob entry for running scanner each third minute

It can also be executed manually, and the same operations will be conducted. The task list is enlisted in a comma-separated file for enhanced readability for the user populating the task list, as seen in figure 4.3.

```
└─# cat comparative_data.csv | sed 's/_[0-9]*,/,/g' | sort -r | uniq | head -n 8
priority,task_name,task_staus,scanner,extra_args
3,nmap_xmas_scan_polite,completed,nmap,"-T2 -sX"
3,nmap_tcp_full_scan_polite,completed,nmap,"-T2 -sT"
3,nmap_svc_discovery_polite,completed,nmap,"-T2 -sV"
3,nmap_ping_scan_polite,new,nmap,"-T2 -PE"
3,nmap_ping_scan_polite,completed,nmap,"-T2 -PE"
3,nmap_os_svc_discovery_polite,completed,nmap,"-T2 -A"
3,nmap_null_scan_polite,completed,nmap,"-T2 -sN"
```

Figure 4.3: Capture of task list.

During this process, a task manager described in 3.4.2, is maintaining the tasklist changing the status on ongoing tasks to complete when they finish. Within this process, the running tcpdump on the worker is terminated, and the tasklist is updated with the correct status. This is an iterative process that requires the script to be manually executed and runs until it is terminated either by a user or unexpectedly.

These are the two most important components of the framework. Additional components are created to efficiently populate a new tasklist from a template tasklist described in listing 27, and a monitoring component which shows details of a running task shown in figure 3.8.

4.3 Data processing

One of the components developed in this research is the PACKET CAPTURE PARSER, further described in section 4.3.1. Inputs to this parser are the result directory containing packet captures captured by each worker during each separate scan task and also an IP ignore list to filter out unwanted IP addresses. The retrieval component synchronizes packet captures to an output directory given by an input parameter. To separate each task from the other for use in the analysis, the directory tree structure for the retrieved packet capture is shown in figure 4.4.

The parser then iterates through the given result directory identifying each respective task by name and the respective files related to the task. Scapy is used to read each representative packet captured

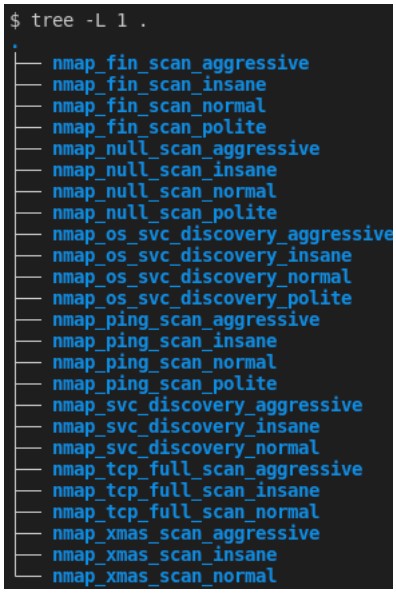


Figure 4.4: Capture of tree directory structure for retrieved packet captures.

into an object. Furthermore, header fields are defined for IP, TCP, UDP, and ICMP header, which are directly written to a file, each resulting in 3 files (TCP, UDP, and ICMP), with the pattern enlisted as *protocol_taskname_scannumber_timestamp.csv*. Each packet is iterated through, where only IP packets are processed. The parser creates a list of each packet during the iteration and detects the protocol before writing the packet details into its representative CSV file. Another Python library used during iteration is the binascii library, which decodes the payload to hex. A comparison of the raw packet capture and the parsed CSV file is shown in figure 4.5. Within the figure, the top 5 lines, including the read message in tcpdump of the raw packet capture file, and the top 5 lines, including the header of the CSV file, are shown as a comparison of the formats. Colorized in the figure are corresponding services in the packet capture file with the port number outlined in the CSV file. As we can see, the telnet service is outputted in the CSV file as port 23, enlisted with the green color.

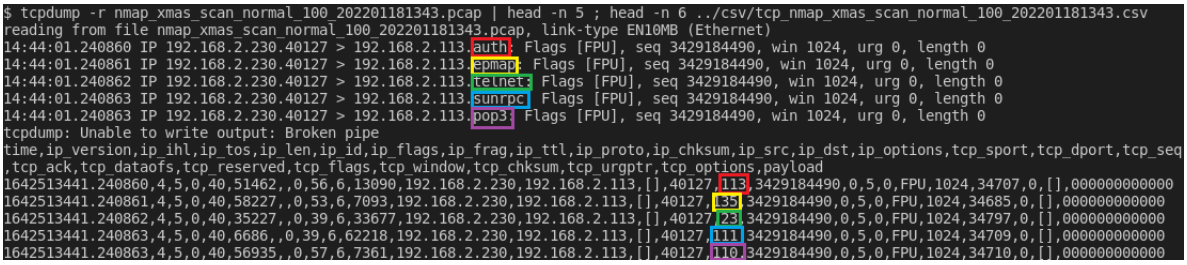


Figure 4.5: Capture of comparison of raw packet capture and parsed CSV output.

4.3.1 Packet capture parser

A packet capture parser was needed for parsing raw packet capture files into comma-separated value (CSV) files. This format is easier to work with and extract data later in the process. The parser is written in Python and uses one user input argument for setting the path for the given packet capture file that is going to be parsed.

Python libraries

Python libraries used in the parser are CSV, sys, os, binascii, and scapy.

The *csv* library are used for writing to CSV files. *sys* is used for exiting the parser and capturing the given input when executing the parser. The os library is used for retrieving the directory and basename for the given packet capture file. Within the binascii library, the hexlify function is used for decoding TCP and UDP payloads to hex format. Scapy[1] are the key library used for the parser. Scapy is used to read the packet capture and extract all relevant content. The choice for using Scapy is based on personal preferences and familiarisation during earlier projects such as the FenrisBox, where Scapy was used to modify packets (Ø. Jacobsen et al. 2021).

Within listing 31 a header structure for CSV files are generated. The output after running this code is three CSV files, each file containing the given protocol. The write operation is defined within lines 1 to 6, and the operation itself is executed in lines 14, 17, and 22 in listing 31. Usage of the same writing operations is done later in the parser also.

The parser writes to three different files, each identified by the protocol in the name. This will return a file multiplication of three files for each packet capture file processed.

```
1  output_tcp = csv.writer(open(dir_name + '/output_tcp_' + base_name.replace(".pcap","") +
2  '.csv', 'w'), delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
3  output_udp = csv.writer(open(dir_name + '/output_udp_' + base_name.replace(".pcap","") +
4  '.csv', 'w'), delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
5  output_icmp = csv.writer(open(dir_name + '/output_icmp_' + base_name.replace(".pcap","") +
6  '.csv', 'w'), delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
7
8  ip_header = ['time', 'ip_version', 'ip_ihl', 'ip_tos', 'ip_len', 'ip_id', 'ip_flags',
9  'ip_frag', 'ip_ttl', 'ip_proto', 'ip_chksum', 'ip_src', 'ip_dst', 'ip_options']
10
11 ip_tcp_header = (ip_header + ['tcp_sport', 'tcp_dport', 'tcp_seq', 'tcp_ack', 'tcp_dataofs',
12 'tcp_reserved', 'tcp_flags', 'tcp_window', 'tcp_chksum', 'tcp_urgptr', 'tcp_options',
13 'payload'])
14 output_tcp.writerow(ip_tcp_header)
15
16 ip_udp_header = (ip_header + ['udp_sport', 'udp_dport', 'udp_len', 'udp_chksum', 'payload'])
17 output_udp.writerow(ip_udp_header)
18
19 ip_icmp_header = (ip_header + ['icmp_type', 'icmp_code', 'icmp_chksum', 'icmp_id',
20 'icmp_seq', 'icmp_ts_ori', 'icmp_ts_rx', 'icmp_gw', 'icmp_ptr', 'icmp_reserved',
21 'icmp_length', 'icmp_addr_mask', 'icmp_nexthopmtu', 'icmp_unused', 'payload'])
22 output_icmp.writerow(ip_icmp_header)
```

Listing 31: Generating CSV header in the Pcap Parser

The parser extracts IP header values as a bare minimum, shown in figure 4.6a.

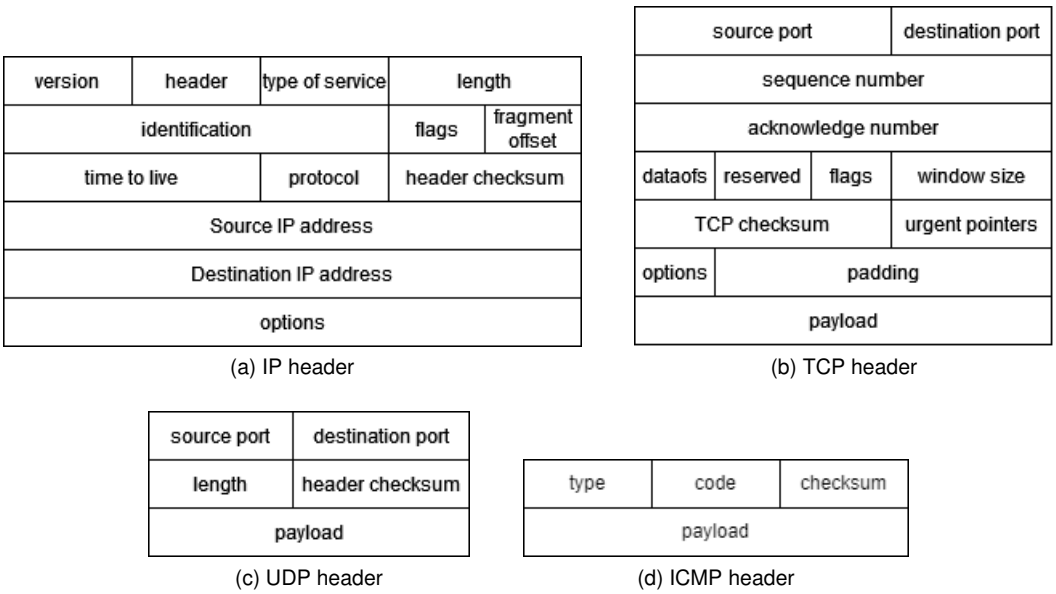
The values within the IP header in figure 4.6a is paired against the fields defined in lines 8 and 9 in listing 31 and the values used in Scapy. This pairing is documented in table 4.1.

Depending on which protocol each packet is identified with, the parser is able to extract values from the TCP header (shown in figure 4.6b), UDP header (shown in figure 4.6c), and ICMP header (shown in figure 4.6d).

[1] https://scapy.net/

Table 4.1: Mapping between CSV fields, IP/TCP/UDP/ICMP header and Scapy

IP header	CSV	Scapy		TCP header	CSV	Scapy
Version	*ip_version*	*version*		Source port	*tcp_sport*	*sport*
Header	*ip_ihl*	*ihl*		Destination port	*tcp_dport*	*dport*
Type of service	*ip_tos*	*tos*		Sequence number	*tcp_seq*	*seq*
Length	*ip_len*	*len*		Acknowledge number	*tcp_ack*	*ack*
Identification	*ip_id*	*id*		Dataofs	*tcp_dataofs*	*dataofs*
Flags	*ip_flags*	*flags*		Reserved	*tcp_reserved*	*reserved*
Fragment offset	*ip_frag*	*frag*		Flags	*tcp_flags*	*flags*
Time to live	*ip_ttl*	*ttl*		Window size	*tcp_window*	*window*
Protocol	*ip_proto*	*proto*		TCP checksum	*tcp_chksum*	*chksum*
Header checksum	*ip_chksum*	*chksum*		Urgent pointers	*tcp_urgptr*	*urgptr*
Source IP	*ip_src*	*src*		Options	*tcp_options*	*options*
Destination IP	*ip_dst*	*dst*		Payload	*payload*	*payload*
Options	*ip_options*	*options*				
UDP header	CSV	Scapy		ICMP header	CSV	Scapy
Source port	*udp_sport*	*sport*		Type	*icmp_type*	*type*
Destination port	*udp_dport*	*dport*		Code	*icmp_code*	*code*
Length	*udp_len*	*len*		Checksum	*icmp_chksum*	*chksum*
Checksum	*udp_chksum*	*chksum*		ID	*icmp_id*	*id*
Payload	*payload*	*payload*		Sequence number	*icmp_seq*	*seq*
				Originate timestamp	*icmp_ts_ori*	*ts_ori*
				Receive timestamp	*icmp_ts_rx*	*ts_rx*
				Gateway	*icmp_gw*	*gw*
				Pointer	*icmp_ptr*	*ptr*
				Reserved	*icmp_reserved*	*reserved*
				Length	*icmp_length*	*length*
				Addr mask	*icmp_addr_mask*	*addr_mask*
				Next-hop MTU	*icmp_nexthopmtu*	*nexthopmtu*
				Unused	*icmp_unused*	*unused*
				Payload	*payload*	*payload*



(a) IP header

(b) TCP header

(c) UDP header

(d) ICMP header

Figure 4.6: IP, TCP, UDP and ICMP header

The ICMP header is different from the TCP and UDP header, which have fixed fields. There are 3 fixed values for the ICMP header, which are type, code, and checksum. Remaining of the ICMP header depends on the *type* given according to RFC 792 (Postel 1981). This meaning, if the ICMP code is either 8 (echo message) or 0 (echo reply message) the ICMP header would include the *identifier* and *sequence number* fields as visualised in figure 4.7. As seen in these figures and table 4.1, Scapy is able to extract all the values for a packet.

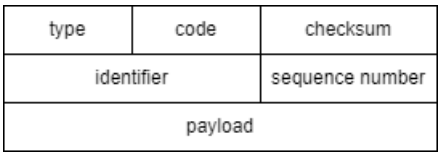


Figure 4.7: ICMP echo header

4.3.2 Bulk pcap parser

This script has the objective of iterating through a given directory containing packet capture files. Inputs to the script is the path to the directory where the PACKET CAPTURE PARSER is located and the location of the directory containing the packet capture files to be parsed. It identifies each packet capture file within a given directory and executes the PACKET CAPTURE PARSER.

```
#!/bin/bash
# Parse PCAP to CSV
PARSER_PATH=$1
PCAP_DIR=$2

if [ -z $PCAP_DIR ] && [ -z $PARSER_PATH ]; then
  echo "usage: bash $0 <path to pcap parser> <pcap directory>";
  exit
fi

for FILE in $(find $PCAP_DIR -name "*cap" -type f); do
  FILENAME=$(basename $FILE)
  DIRNAME=$(dirname $FILE)

  python3 $PARSER_PATH/pcap_argv_parser.py $FILE
done
```

Listing 32: Execute bulk pcap parsing using Python Pcap parser

4.4 Analysis

By generating a certain amount of packets, captures makes a valid number for conducting a comparative analysis on the synthetic generated data sets. During this research, the main analysis component chosen was the Jupyter notebook.

4.4.1 Jupyter notebook

Jupyter notebook is a non-cost web application used for editing code in the browser, which includes tab completion, highlighting, and indentation. Jupyter has the ability to display media formats such as LaTeX, images, and HTML. It also supports markdown language for adding text in an easily readable format which in the browser is compiled into HTML for the user. This enhances the readability for further commenting code by using for mating and is not only limited to code comments. The documents for Jupyter is called notebooks, which are JSON file containing computational session data recorded during a session (Sharma et al. 2021; Team 2015). This means that when data are inserted, such as in this packet capture research, it is recorded in the *.ipynb* file. When conducting packet capture analysis and reading amounts of data into the notebook, the processed data are stored, which streamlines the process of analysis of the data. The structure of a notebook is divided into cells, where either code, text, or media can be inserted and executed. Other useful features include popular used export formats such as LaTeX compiled PDF, Python, LaTeX, and HTML. A notebook can be easily opened when shared only by having Jupyter installed and running, without the need of re-running the notebook since all data already are saved as a checkpoint. Jupyter has the capability of running all code in one single push, which enables a semi-automatic analysis with the base in a notebook template.

Jupyter uses the underlying Python on the Jupyter host machine, which enables the use of all installed libraries on the machine. A capture of a part of the Jupyter notebook is shown in figure 4.8. The figure shows a code snippet that generates a plot of packet number correlated with destination ports and colorized with source ports, which represents scan number. At the top of the figure, a play button is shown, which enables the user of the notebook to execute the code within the notebook, and the results are represented either in a plot, raw text format, or table format. Another important feature in the Jupyter notebook is the checkpoint feature, which enables users to save the notebook in a given state for later use. This enables the possibility of sharing the notebook in the current state containing the data at a given stage for other users to review the results.

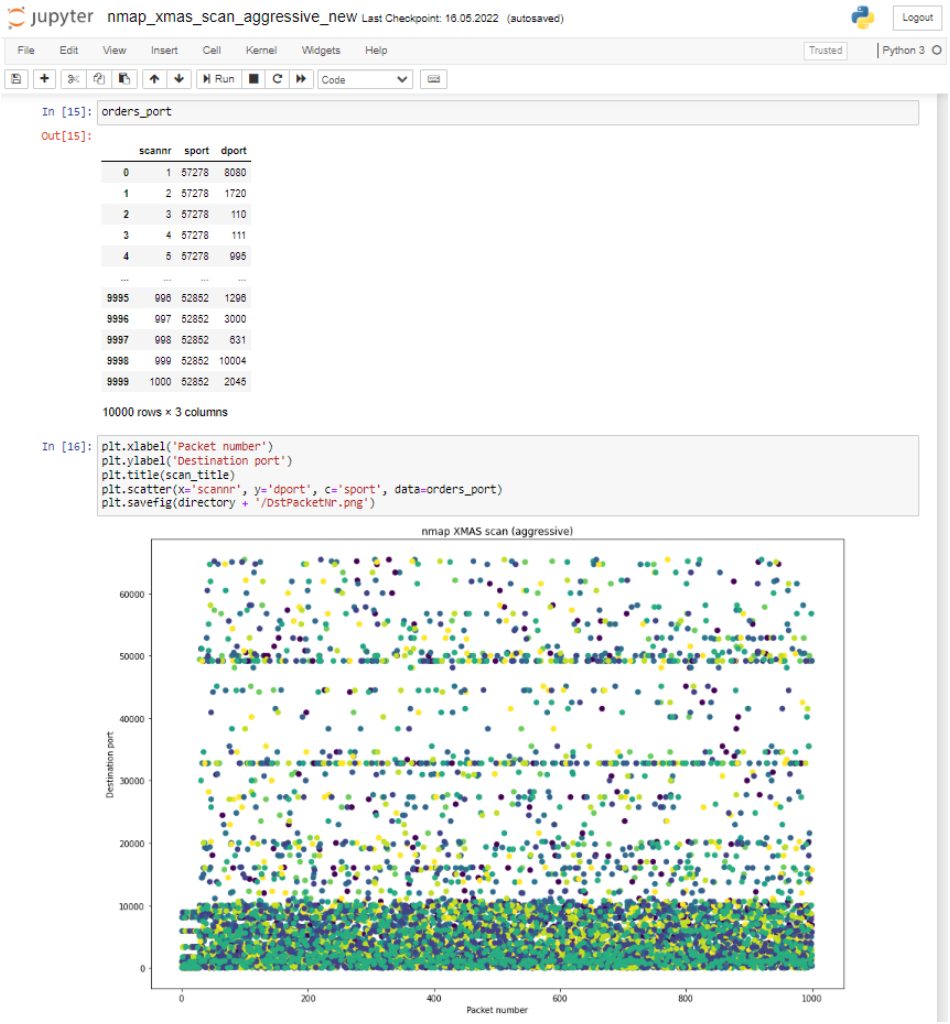


Figure 4.8: Capture of a part of the Jupyter notebook.

4.4.2 Python libraries for analysis

A popular Python library called MATPLOTLIB is often used for visualization, and this is supported in Jupyter. The visualizations in this research use matplotlib for plotting data for visualization, as shown in figure 4.8. The Pandas library[2] is a Python-built analysis tool that is used in this analysis. In listing 33 the data from given CSV files in a directory are imported into a Panda DataFrame in line 9. This is later used as an important data format for the whole analysis.

[2]https://pandas.pydata.org/

Table 4.2: Python libraries used in analysis

Library	Usage
matplotlib.pyplot	Used for plot generation.
matplotlib.cm	Used for rainbow color generation.
pandas	Used for reading CSV files into Panda DataFrames for later use in data visualisation in plots with matplotlib.plot and table forms.
numpy	Used for numeric sequence creation for use in matplotlib to generate rainbow colors in plots.
os	Used for reading directory contents, checking file existence and and extracting file names.

```
directory = '/home/user/notebooks/pcaps/nmap_xmas_scan_aggressive'
scan_list = []

for filename in os.listdir(directory):
    f = os.path.join(directory, filename)
    if os.path.isfile(f):
        filename, ext = os.path.splitext(f)
        if ext == '.csv':
            read = pd.read_csv(f)
            if read.empty:
                pass
            else:
                scan_list.append(read)
```

Listing 33: Importing data from CSV files and generate Panda dataset in Jupyter

4.4.3 Analysis methodology

The analysis in Jupyter in this research is built up as a Jupyter template. This template is reused for the various types of scans and also for the various timing templates.

The notebook is structured with an import section where the chosen libraries are imported, the IP address to the scanning host is defined, and the title of the scan is defined. The IP address to the scanning host is defined for later in the process to filter out only incoming packets and not outgoing packets to the scanning host. Further the CSV files processed with the PACKET CAPTURE PARSER, described in section 4.3.1, are imported to a Panda data frame.

The analysis then goes into investigating the duration of each scan for the chosen timing template and scan type. A histogram is generated and visualizes the count of scans related to the duration of the scan. Figure 5.1 shows the histogram for various scanning types and timing templates, which is the first analysis visualization for each of the given scans in this research's Jupyter notebook. Measurements for establishing the minimal and maximum time used, the standard deviation, and the average are also visualized in a table, such as a table 5.2.

During the next two sections in the notebook, the packet count and packets per second are measured in order to identify any eventual anomalies in sent and received packets. Comparing the average packets per second rate to the used timing template can give an indicator of how fast or slow the scan is conducted.

Further, the orders of ports scanned are logged in order to spot similarities for which ports are scanned in which order. The similarities could easily be seen in a figure where each scan is represented in

its own color, such as in figure 5.4. Other statistical measurements were also conducted during this research, among visualizing source ports to destination ports on one single scan by visualizing the scan number and destination ports in a diagram results in a good visualization of locating where the main number of ports have been scanned. This would be described in the results in section 5.4. The importance of visualizing the scans gains a higher situational awareness of what is going on in each scan, and it simplifies the analysis of spotting anomalies and patterns.

A mathematical code computation was created to figure out each of the packet sizes for each received packet from the scanning host. The main reason behind this was to establish an eventual deviation from the "standard" packet size for each received packet. This will be further elaborated in the results chapter under section 3.6.

Other analysis parameters added to the analysis are the TCP window size and sequence number. Extraction of the TCP window size was added to easily spot abnormal window sizes. The sequence number is used for comparison against which number of scans is conducted, destination port, and source ports. These measurements are not further described within this research, but further analysis information is located within the Jupyter notebooks for the respective scans.

4.5 Summary

By using this lab environment described in chapter 3, the generation of synthetic packet captures can be conducted. This is a large step in streamlining the process of producing such data compared to conducting these scans in a manual method. A manual method increases the risk of errors and produces incomparable data if not conducted in the same procedure for all scans. The raw packet capture files are retrieved from each worker and parsed to comma-separated value (CSV) files by using the PACKET CAPTURE PARSER described in 4.3.1. These parsed CSV files are then read into the Jupyter notebook, where the semi-automatic analysis takes place. In the Jupyter notebook, it is possible to press *Run* to go through the whole analysis automatically. The manual required procedure if the notebook is to input the relevant variables in the head of the file, such as the destination for the given scan, the scan title, and the default figure size. The main reason for using the Jupyter notebook for analysis purposes is to streamline the process making the process semi-automatic. Only semi-automatic because the process is not fully automated due to the required given parameters in the head of the file must be changed to match the respective scan. The analysis notebook is segmented into sections mainly focusing on scan duration, packet counts, packets per second, and scanned port order. Plots for visualizing similarities and patterns were conducted for combining values for sequence numbers, IP ID, ports, and scan numbers. These plots have shown to significantly show interesting findings regarding traffic patterns, which will be further discussed in chapter 5.

5 Results

5.1 Introduction

Within this chapter, the results conducted through the analysis earlier discussed in section 4.4 be presented and discussed. By using the Jupyter notebook for conducting the analysis, visualizations are created in order to clearly see patterns that require further analysis. The most significant findings during the analysis are presented within this chapter.

Chosen Nmap scanning types are *xmas*, *connect* (TCP full scan), *null*, *fin*, *host discovery (ping)* and *service scan*, further described in section 2.16. All of Nmap's timing templates were used in the data generation, which is used for comparing the various scanning types in this chapter. To create a larger basis for comparison when visualizing diagrams in the results, one can type selected together mixed with one timing template. In this case, both the timing templates and the scan types can indicate what is normal for a scan by using visualizations.

Packet count metrics are further elaborated based on the selection process mentioned above. The metrics extracted are the packets per second (PPS) indicating standard deviation, minimum and maximum packets per second.

Port sequence analysis was applied in order to gain an understanding of the eventual existence of the same port sequence for each separate scan. This will be further described in the chapter, together with a number of unique source ports used for each scan. Each packet number is plotted in a diagram to visualize where the main emphasis of ports is targeted. Furthermore is the usage of both targeted (abbreviated as *used ports*) and untargeted (abbreviated as *unused ports*) ports presented.

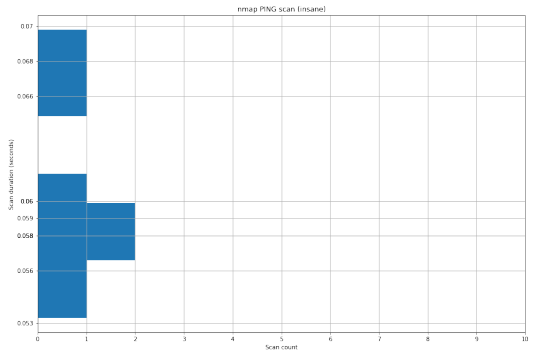
5.2 Packet counts

While examining the numbers, there are a few numbers that break out of the scope of each timing template. Those who really point out the numbers in the table are the service discovery scan used with an insane and aggressive timing template and the ping scan used with an aggressive and normal timing template. Nmap, by default, uses the normal timing template ($-T3$) to conduct scanning activities (Lyon 2009). In table 5.1 the packets pr. the second rate has many similarities to the given timing template compared to other types of scans. However, there are some numbers that stands out such as the *ping scan* using the *aggressive* and *normal* timing template, and the service scan using *insane* and *aggressive* timing template. These numbers are significantly higher compared to the other scans in the table with the corresponding timing template. In the *insane* column the packet per second rate is mainly between 1216 and 1559, *aggressive* between 1195 and 1732 and *normal* between 1196 and 1566 packets per second.

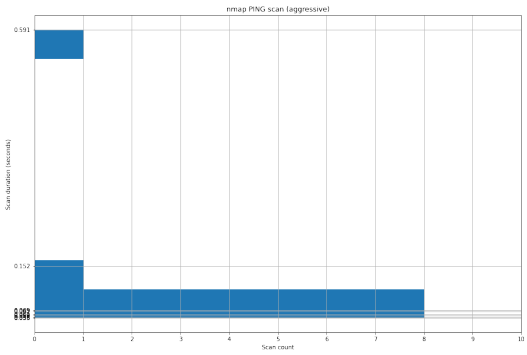
Table 5.1: Measurement of packets pr. second

Nmap scan	insane	aggressive	normal	polite	sneaky	paranoid
xmas scan	1310.072	1229.611	1196.482	49.980	1.333	0.067
fin scan	1328.788	1195.168	1296.515	49.978	1.333	0.067
null scan	1310.763	1288.222	1309.800	49.980	1.333	0.067
ping scan	1216.341	2477.556	2104.646	49.936	1.333	0.067
tcp full scan	1559.437	1732.328	1566.229	50.004	1.335	0.067
service scan	3348.940	4072.136	1253.609	49.979	1.333	0.067

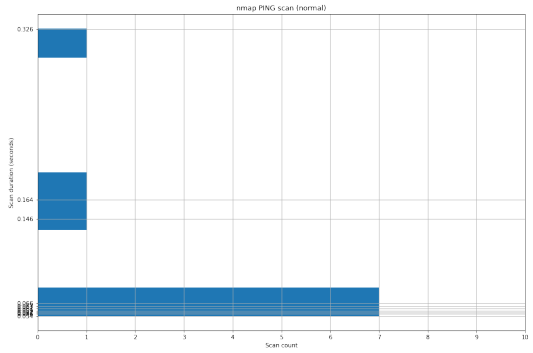
Another interesting packet per second finding is the *service discovery scan*, which used significantly more time in the *insane* and *aggressive* scan compared to other scan types. This clearly also points toward a commonly used type of scan where the user would want a fast reply, therefore using either the *insane* or *aggressive* timing template to quickly retrieve a scan result. Figure 5.1c confirms these findings, though 9 of the ten normal ping scans are shown to be in the range of 53ms (milliseconds) and 164ms. The 10^{th} scan in comparison used 326ms, which used significantly more time than the other scans, and this increases the average duration time significantly. Within figure 5.1b the aggressive ping scan shows similar results where 9 out of 10 scans used between 54ms and 152ms where the 10^{th} scan took 590ms. Within this scan, the average scan duration time is increased due to this 10^{th} scan. Similar to these findings is the insane service discovery scan shown in figure 5.1d and the aggressive service discovery scan shown in figure 5.1e. The insane service discovery scan had nine scans which took between 56ms and 216ms, where the 10^{th} took 915ms, as shown in figure 5.1d. 9 out of the ten aggressive service discovery scans used between 54ms and 107ms, shown in figure 5.1e, where the 10^{th} took over 13 times the time (1.44 seconds) that next on the list did. This can point to a high network load when conducting a certain number of scans from one scanner host.



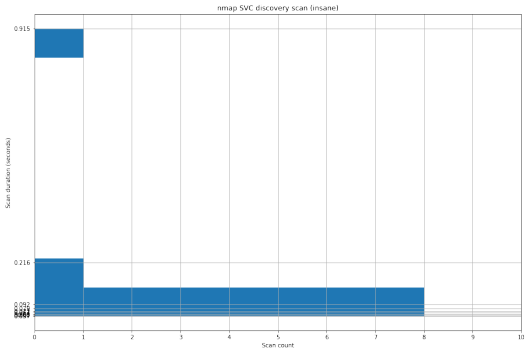
(a) ping scan insane



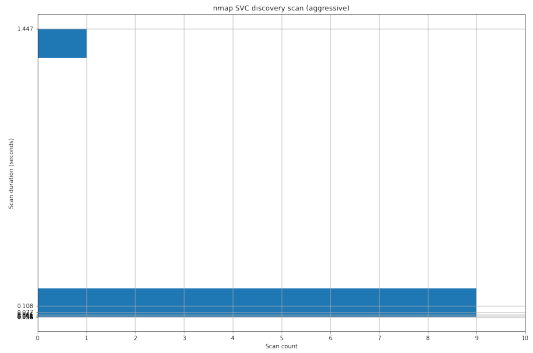
(b) ping scan aggressive



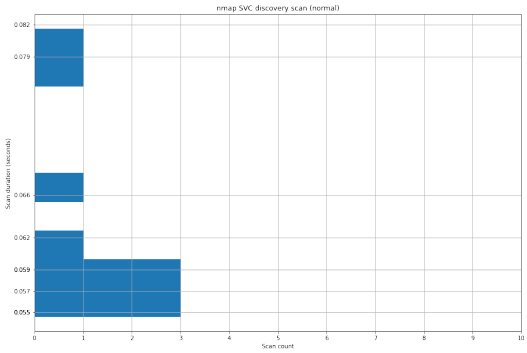
(c) ping scan normal



(d) service discovery insane



(e) service discovery aggressive



(f) service discovery normal

Figure 5.1: Identifying scans significantly increasing scan duration

The scan metrics for these scans are merged into table 5.2 where also a collection of two other scans, shown in figure 5.1a and figure 5.1f, are located. These two scans, the insane ping scan and the normal service discovery scan, show a very small standard deviation (std) as low as 0.005 seconds (5ms) and 0.010 seconds (10ms). This shows signs of the normal time a scan of the given type with the given timing template would take. In comparison, the other four scans have a significantly higher standard deviation, where the highest is the aggressive service discovery has a standard deviation

Table 5.2: Description of scan duration standing out statistics (seconds)

	svc insane	svc aggressive	ping normal	ping aggressive	ping insane	svc normal
count	10	10	10	10	10	10
mean	0.167	0.204	0.105	0.124	0.061	0.063
std	0.267	0.437	0.087	0.166	0.005	0.010
min	0.057	0.055	0.054	0.055	0.053	0.055
25%	0.060	0.056	0.057	0.059	0.058	0.055
50%	0.069	0.058	0.062	0.064	0.059	0.059
75%	0.089	0.074	0.126	0.069	0.064	0.065
max	0.915	1.447	0.326	0.591	0.070	0.082

of 437ms. The deviation results for the insane ping scan and normal service discovery scan prove a normal scan with a decent deviation.

As presented in table 5.2, only a chosen collection of scan types and timing templates are presented. These are in detail described in each respective Jupyter notebook in appendix C.

5.3 Port sequences

During the analysis of the port sequence, where each scanning type was compared between the respective timing templates, there were some interesting findings.

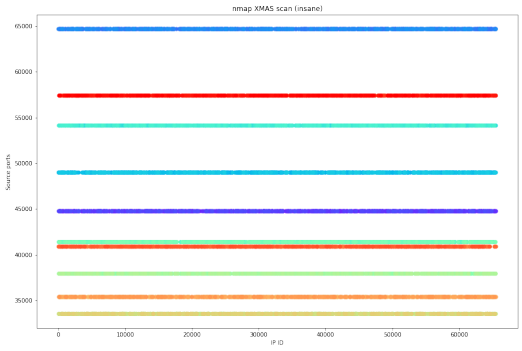
First, the IP ID field was plotted together with the source port for each conducted scan. Figure 5.2 shows diagrams of various types of scans with various timing templates. Common for 5 of these scanning types is that the scan uses the same source port for all sent packets, as the figure shows, though the sneaky and paranoid scans must be elaborated further. The TCP SYN scan could be viewed as artwork, illustrated in figure 5.2b, which shows colors all over the specter which means it uses a different source port for each sent packet. Figure 5.2a, 5.2c and 5.2d shows each 10 lines in each of the figures, which means the same source port are applied to each scan conducted. The same applies to the sneaky and paranoid scan shown in figure 5.2e and 5.2f, though in a limited way since these two-timing templates bring a different pattern into the diagram. The diagram still shows ten solid lines, but it also shows source ports that are close to the solid line, which makes the given source ports not only 10 but more.

Within the Jupyter notebook a simple line of code could be applied to further investigate why it appears “feathers” on the sneaky and paranoid scan, shown in listing 34.

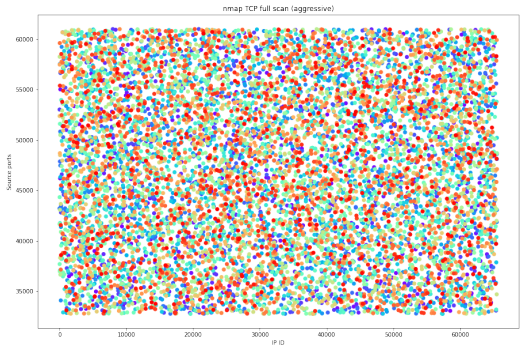
```
set(orders_port['sport'])
len(set(orders_port['sport']))
```

Listing 34: Print unique source ports and length

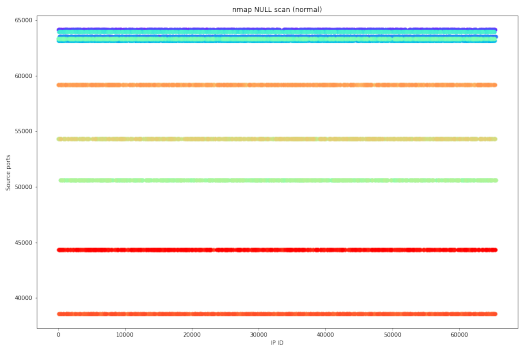
This line of code extracts each unique source port during the ten scans of the chosen scanning type with the corresponding timing template. Further investigation on the paranoid ping scan shown in figure 5.2f shows 1000 unique source ports for this scan, while the sneaky fin scan in figure 5.2e uses 915 unique source ports. Worth investigating is the aggressive TCP SYN scan as well, as the figure 5.2b shows dots all over the diagram. By applying the same code to this scan in Jupyter, the number of unique source ports used for this scan is 7122 ports. Since these scans in figure 5.2 each



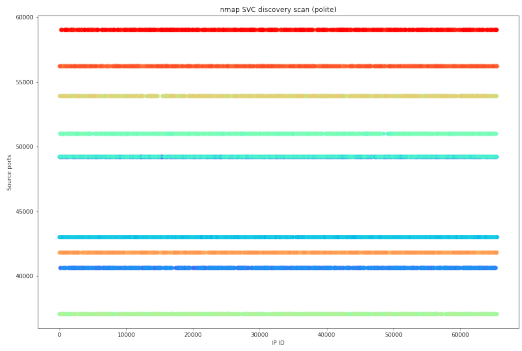
(a) xmas insane



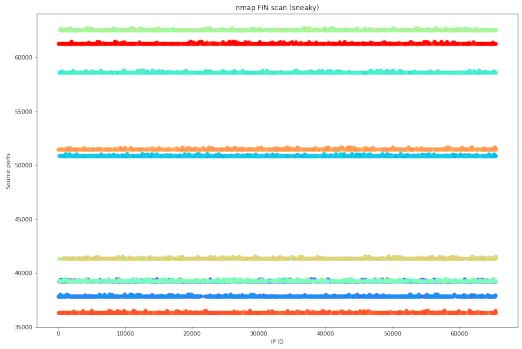
(b) TCP SYN aggressive



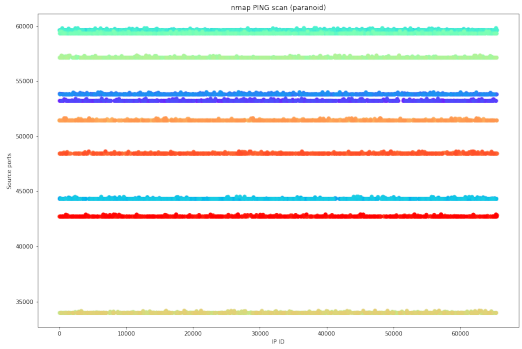
(c) null scan normal



(d) service discovery polite



(e) fin scan sneaky



(f) ping scan paranoid

Figure 5.2: Diagram of IP ID and source ports with various scan types and timing templates

are a different type and different timing templates, investigating the usage of unique source ports is conducted and shown in table 5.3. This proves that all of the chosen scanning types except the TCP SYN scan have one dedicated unique source port for each conducted scan for the insane, aggressive, normal, and polite timing template.

To further examine the retrieved source ports, the command in listing 35 can be applied to the Jupyter notebook to list up the top 30 source ports with a count of how many times they were used.

Table 5.3: Number of unique source ports

scan type	insane	aggressive	normal	polite	sneaky	paranoid
xmas scan	10	10	10	10	1000	999
fin scan	10	10	10	10	915	910
null scan	10	10	10	10	1000	975
ping scan	10	10	10	10	1000	1000
tcp full scan	7108	7122	7116	7144	7743	7664
svc	10	10	10	10	967	998

```
orders_port['sport'].value_counts()[:30]
```

Listing 35: Investigating top 30 unique source ports

On all of the conducted paranoid and sneaky scans, except for the TCP SYN scan, the top 10 source ports are counted either 1000 or 1001 times. The 1001 count appears four times among all the mentioned scans, while 1000 is the regular count seen. This could be shown in the figure 5.3 where we see the top 10 has a count of 1000 or 1001.

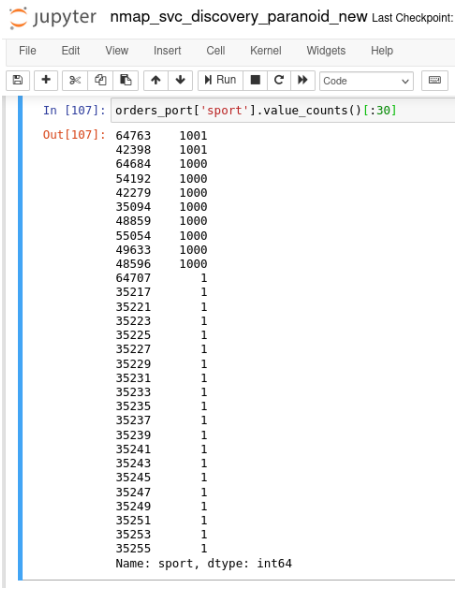
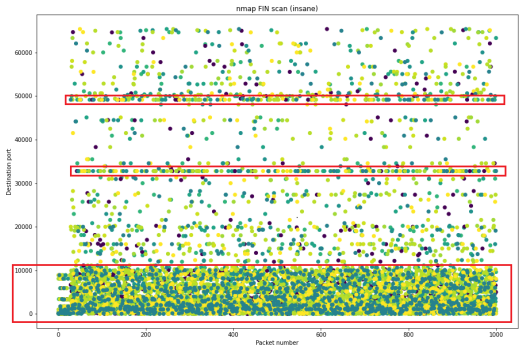
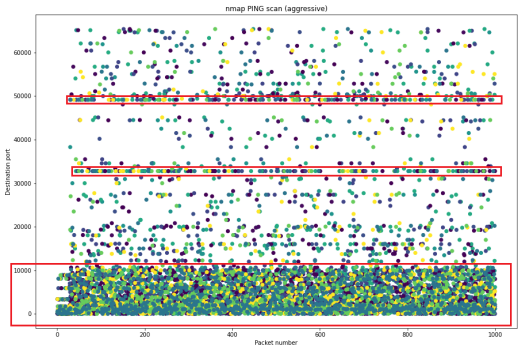


Figure 5.3: Jupyter notebook top 30 source ports used in Service Discovery scan

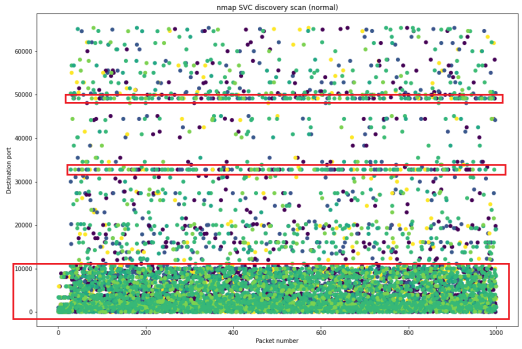
By comparing the various types of scans, each represented with its own timing template, there are definitely similarities between the conducted scans. The port sequence scatter plot in figure 5.4 clearly outlines this. Within the figure, each color symbolizes a unique source port number, The highest amount of destination ports reached is located close to and below port 1000. This similarity is clear evidence that Nmap focuses on ports around this port range. Other interesting characteristics of this figure are the line right above port 30000 and the line right below port 50000.



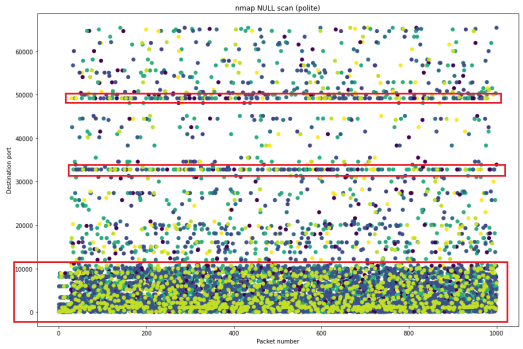
(a) fin scan insane



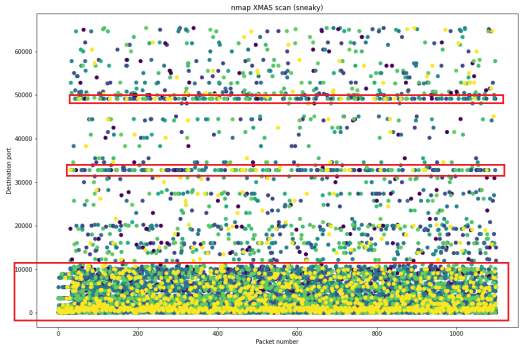
(b) ping scan aggressive



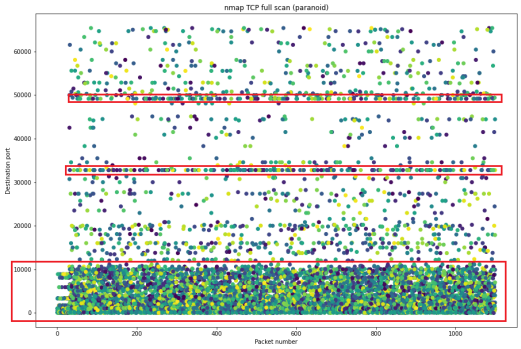
(c) service discovery normal



(d) null scan polite



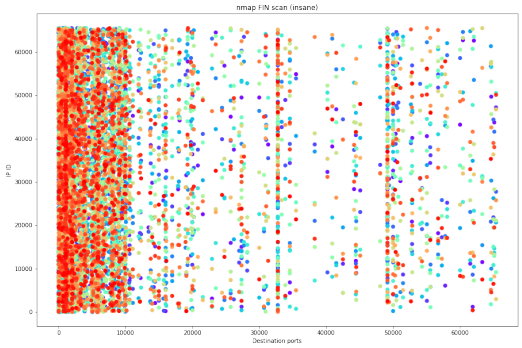
(e) xmas scan sneaky



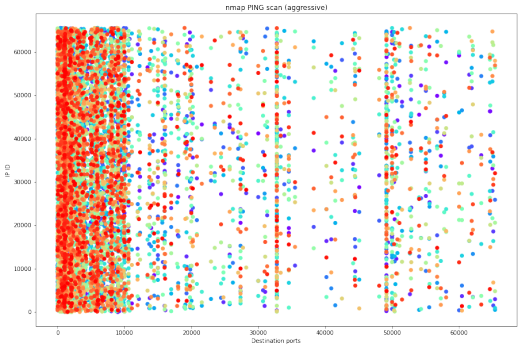
(f) TCP SYN paranoid

Figure 5.4: Destination port sequence with various scan types and timing templates

In figure 5.4 similarities are highlighted. By further looking at these patterns, the same patterns can be seen in figure 5.5 only with a flipped x and y-axis. Though within figure 5.5 the values of the IP ID field is used instead of the packet number as seen in figure 5.4. It is clear from the figure that the values of the IP ID field go through the whole specter close to 0 up to over 60000. Adding one line of code at the end of the Jupyter notebook for printing the ip_ids variable, which is a list, confirms this. This line was added to the $nmap\_null\_scan\_polite$ notebook, and the result from this shows the lowest IP ID field to be 13, while the highest was 65533.

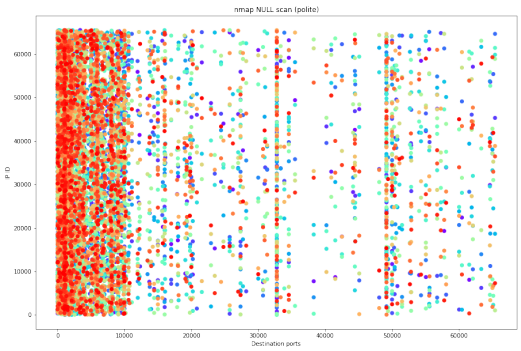


(a) fin scan insane

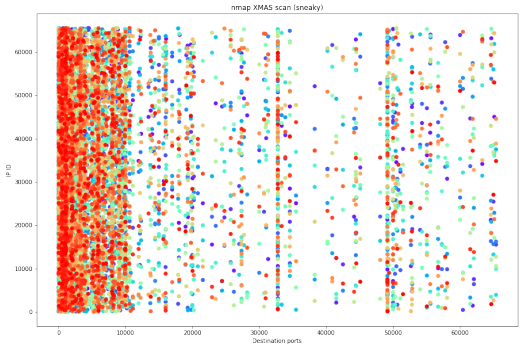


(b) ping scan aggressive

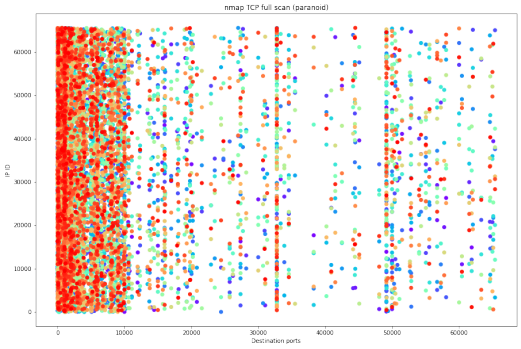
(c) service discovery normal



(d) null scan polite



(e) xmas scan sneaky



(f) TCP SYN paranoid

Figure 5.5: Destination port and IP ID with various scan types and timing templates

Further in the analysis, the IP ID correlated with the destination port, shown in figure 5.5, in a diagram shows the similar plots as in the port sequence diagram in figure 5.4. The main part of ports targeted are under or close to 10000, and there's a line above 30000 and a new line just below 50000. By rotating the diagram in figure 5.5 90 degrees to the left, these plots could be compared to figure 5.6 which clearly shows the same patterns as the destination port is a common denominator between the two mentioned figures.

5.4 Destination port patterns

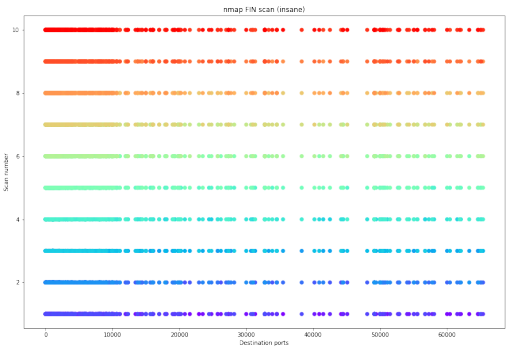
By comparing a collection of scans by type and timing template, a clear pattern could be shown as in figure 5.6. Out of these sub figures, the patterns for each respective scan are similar. The same ports are targeted within the time span of one scan.

The figure does not clearly point out the sequence of which ports are targeted in sequence. This is elaborated in section 5.3. It neither tells anything about how many destination ports is targeted. A code snippet could be implemented to go through which ports are not targeted, as seen in listing 36.

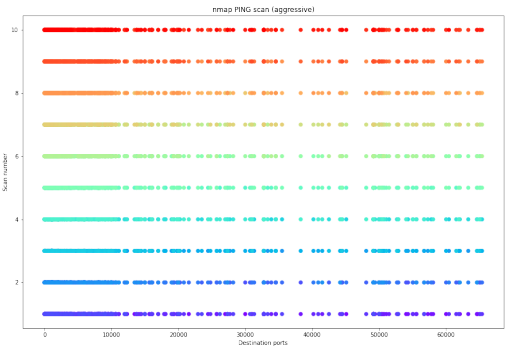
```
unused_ports = []
used_ports = []
for x in range(1, 65536):
    if x not in tcp_dports:
        unused_ports.append(x)
    elif x in tcp_dports:
        used_ports.append(x)

print(len(unused_ports)) # print the count of unused ports
print(unused_ports) # print all the unused ports
print(len(used_ports)) # print the count of used ports
print(used_ports) # print all used ports
```

Listing 36: Print values and counts of unused ports

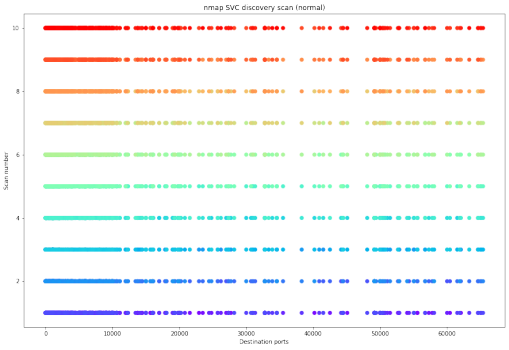
The code in listing 36 were implemented into the Jupyter notebooks for the scans seen in figure 5.6. Each of these notebooks reported an unused port count of 64535 ports for each scan where the scanning type used and the timing template used were irrelevant, and the count stayed the same.



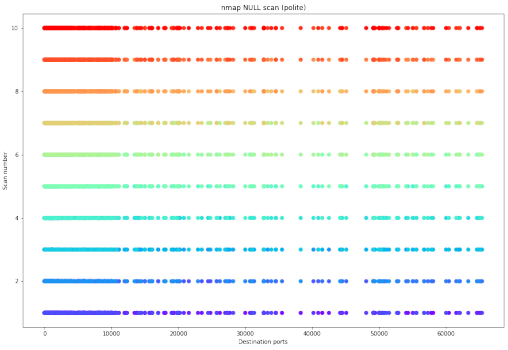
(a) fin scan insane



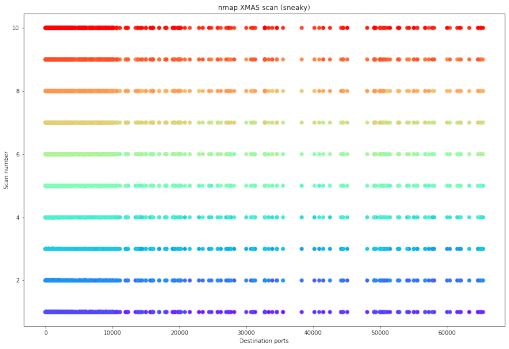
(b) ping scan aggressive



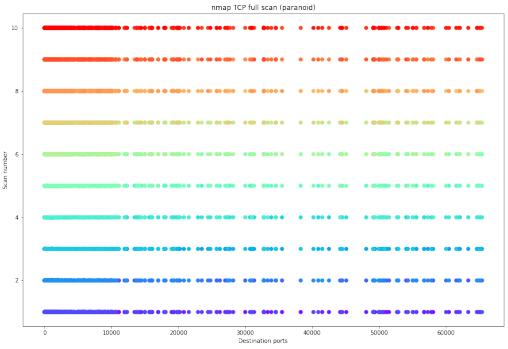
(c) service discovery normal



(d) null scan polite



(e) xmas scan sneaky



(f) TCP SYN paranoid

Figure 5.6: Scan number and destination port diagram

5.5 Summary

Using the Jupyter notebook for analysis of all parsed packet captures combined with the use of visualizations increased the quality outcome of the analysis. The most interesting findings of the analysis were presented deeper in each respective subsection within chapter 5.

The packet count was compared against the various types of scanning types and the timing templates in order to find similarities and to stand out statistics. Standing out was the *ping scan* using the *aggressive* and *normal* timing template, which had a significantly higher packet per second (PPS) compared with the next scans on the list. Other standouts were the *service scan* using *insane* and *aggressive* timing templates. Using the *insane* timing template, it had over two times higher PPS score compared to the TCP full scan. Using the *aggressive* timing template, it had a 1.64 times faster PPS compared with the ping scan. These results point to 1 out of 10 scans clearly using a significant amount of more time compared to the others. This can point to a high network load. These are further investigated in section 5.2.

Port sequences were investigated and concluded with not following the same sequence for each scan conducted. Out of the six scan types investigated, five of these clearly show the same patterns regarding source ports targeted and the IP ID number used. The*TCP full scan* deviates from this logical pattern, shown in figure 5.2. Five of these patterns consists of the same source port for one scan with various IP IDs. The *TCP full scan* clearly differs from these results with a different source port for each packet sent, resulting in an artwork diagram. The *TCP full scan* has a number of unique source ports in a range between 7108 and 7664, shown in table 5.3. Other interesting patterns is figure 5.2 is the *sneaky* and *paranoid* scan, which shows “feathers” symbolizing using not only one source port, but a port within close range. Further, the targeted destination ports show similar patterns among the various scanning types and timing templates. The main range of destination ports is ranges below 10000, while other patterns symbolizing a clear line are shown both in the range below 30000 and below 50000. This is shown in figure 5.4. The same destination port patterns are also shown when plotting the IP ID and the destination ports in a diagram shown in figure 5.5.

Regarding destination port patterns, a clear similarity between all various scan types and timing templates is seen. The same targeted destination ports are targeted for each scan type and for each timing template, resulting in similar diagrams shown in figure 5.6. Interesting is the existence of unused destination port blocks, shown in the same figure. By executing a code snippet for mapping out untargeted and targeted ports, the amount of untargeted ports is 64535 resulting in 1000 targeted ports which can be compared to the packet count of 1000 in one scan.

6

Conclusion

6.1 Introduction

In this thesis, the development of a framework for synthetic packet generation and characterization of Nmap scans are discussed. During this thesis, a series of tests have been conducted while continuously developing both the scripts needed to conduct the automated scanning procedures together with the other necessary components for task management. After the lab environment was completed, the packet generation was started. Issues with the developed tools were immediately fixed when encountering errors making this environment more reliable. The given scan numbers were 10 for each scan type and 10 for each timing template, resulting in 60 scans for one scan type. In total, 360 packet captures were created for empirical analysis purposes. When all tasks were completed and the number of given scans was reached, the development of the analysis template in Jupyter was continued. This was a work in parallel while waiting for all scans to be completed. To be able to analyze the captured datasets, the raw packet captures had to be parsed to a comma-separated value format. The development of the PACKET CAPTURE PARSER was done. The parser converted the raw packet captures to CSV files, and the analysis could start. Giving the required parameters to the analysis template, a run through the given Jupyter notebook was completed resulting invaluable data and visualizations later used for analysis, described both in chapter 4 and chapter 5.

6.2 Summary of Research

During the establishment of the research project, a Scrum methodology[1] was adopted with the use of the Kanban board. Primarily this was established to keep a structured view of progression, maintain

[1] https://www.scrum.org/resources/what-is-scrum

issues and tasks, and streamline the development process to accomplish the research objectives. A step-to-step approach to achieving each milestone was crucial to streamline the process, given the specific timeframe set for the research. In the Scrum methodology, sprints and sprint goals are an important part of the methodology to achieve the given goals. Within this research, Github milestones were used as an alternative to Scrum's sprint goals since these milestones integrate with Github repositories and Github projects. Other usages within Github were the project feature, containing Kanban board for issue tracking and task management[2].

In the early phase of the research, a lab environment fully segmented from the local network was needed to conduct this type of research. This was created as a first step where the chosen virtualization platform was VMware Workstation due to previous knowledge of the use and setup of the platform. By choosing this platform, the risk of non-familiar setup and creation issues was mitigated. The chosen operating system for each worker was Ubuntu Linux, also as a personal preference and experience within the system. The same applies to the choice of the scanner operating system, which was Kali Linux. Kali Linux already had all the required tools installed for the conduction of this research. Regarding the choice of the Nmap, scanners were also an argument of familiarisation and earlier use cases.

The choice of automating this was taken early as a step toward streamlining the data generation process. During the project initiation, the chosen number of scans was 100 scans. This was later reduced and restricted to 10. The main reason behind this was the consummation of time for the slowest timing templates, especially the *paranoid* timing template. A *paranoid* scan kept a worker busy over a certain amount of time, resulting in not being able to conduct other faster tasks simultaneously. Another important argument for this was the number of scans for each timing template needed to be the same amount of scans.

The main programming language chosen for the development of the tools used for automating the scans was Bash. The choice was based on personal development experience in the language. Another important argument for choosing Bash is that its default is installed on most Linux distributions, making installation of other tools and software which require the internet not relevant. The scripts developed within this research are further described in section 3.4.

For the generated packet captures, parsing of the data from packet capture format to a comma-separated value (CSV) was needed in order to feed this to the Jupyter notebook during the analysis phase. This parser was written in Python, which extracts all relevant fields from the packet captures and translates them into CSV for analysis purposes. The PACKET CAPTURE PARSER is further described in section 4.3.1.

The analysis part of this research was developed in the Jupyter notebook, further described in section 4.4.1. Jupyter has capabilities of writing Python code in the web browser, which enables the use of visualizations and empirical data analysis. This must be seen as yet another step in process streamlining, making this analysis a semi-automatic procedure. Most importantly with Jupyter is the *Run* function, which makes the code in Jupyter executed step by step without user interactions.

The main risk factor for this research was time management and using products with earlier gained experience reduced this risk factor to a minimum. Other risk factors were hardware crashes and

[2]https://github.com/users/orjanj/projects/3

backups. These were mitigated by using *rsync*[3] for backing up code and generated production data to other hard drives on the same computer. The component PACKET CAPTURE RETRIEVAL was developed for the purpose of syncing generated packet captures on worker hosts to the scanner host. This component is used on the scanning host to retrieve packet captures from each worker and is further described in 3.4.4. From the host running VMware Workstation, *rsync* were used to back up the data in cases of either hardware crash or virtual machine breakdown.

6.3 Research Objectives

The main objective is the creation of a framework for synthetic packet generation and characterization of Nmap scans. Together with the main objectives for the research is the problem statement and scope and limitations described within chapter 1. Required for these objectives are the design and implementation of a local lab environment, further described in chapter 3, including the necessary tools for conducting data generation and task management. To achieve the goals of conducting empirical analysis, the development of an analysis template is further described within chapter 4, together with the generation, processing, and analysis of data. The actual results of the analysis conducted are presented in chapter 5.

6.4 Research Contribution

This thesis presents a framework for synthetic packet generation using Nmap and an analysis of the captured data. The produced artifacts and expected outputs of this research are the developed tools for conducting data generation, processing, and analysis found in C. These tools enable other researchers to conduct synthetic packet generation together with analysis of the data outcome. By automating this process conducted within this thesis, researchers are able to scope up the generation of data, not only limiting these scans to a number of 10 scans as in this thesis. Useful areas for this framework and other use cases are presented in section 6.5.

6.5 Future work

A valuable future of work based on the outcome of this thesis is the improvement of the framework. This includes further developing the analysis tool to include more visualizations for characterizations of scans. More work should be put into streamlining the code for the tools developed in the research making this more efficient and increasing reliability. The developed code is in a state of working while not focusing on either the efficiency or the aesthetic aspect. While conducting task management for each worker, this is done through SSH. The amount of SSH traffic is as well in a state of working and in an as-is or proof of concept state.

By evaluating the design of the framework, this could be greatly improved. Other improvements to this framework regarding design evaluation are to further develop this framework to fit within a Kubernetes orchestrated cluster[4] combined with the use of Docker containers[5].

[3] https://linux.die.net/man/1/rsync
[4] https://kubernetes.io/
[5] https://www.docker.com/

The framework can be extended from only focusing and using the Nmap scanner to use other scanners. Combining this framework with Metasploit, described in section 2.11, for both service discovery and applying exploits is an interesting future work that can be done. This means an eventual future work for the framework, depending on the scope, could lead to an offensive tool for automatically applying exploits during scans, though this requires additional research. The Zmap scanner described in 2.10 is a fast scanner that could be integrated for use within this framework for generating synthetic packet captures as well for analysis purposes. Nessus, described in section 2.7, and Unicornscan, described in section 2.8, can also be integrated into this framework for also generating synthetic packet captures. Finally, Shodan, described in section 2.9, can be integrated through CLI into this framework. Primarily Shodan is a cloud tool but can run locally as well using an API. Though, some technicalities must be applied here to be able to generate synthetic packet captures. Additional research for this scanner is needed in order to elaborate in detail on the potential of Shodan within this framework.

Finally, development of intrusion detection system signatures for Suricata[6] or [7] based on the packets characteristics from this research is also a alternative future of work. Classification of scanning are further elaborated within section 2.12.

[6] https://suricata.io
[7] https://www.snort.org

References

Antonakakis, Manos et al. (2017). “Understanding the Mirai Botnet”. In: *Proceedings of the 26th USENIX Conference on Security Symposium*. SEC’17. Vancouver, BC, Canada: USENIX Association, pp. 1093–1110. ISBN: 9781931971409.

Arkin, Ofir (1999). “Network scanning techniques”. In: *Publicom Communications Solutions*.

Arkko, Jari, Michelle Cotton, and Leo Vegoda (Jan. 2010). *IPv4 Address Blocks Reserved for Documentation*. RFC 5737. DOI: 10.17487/RFC5737.

Assange, Julian (1995). *strobe HISTORY*. URL: http://ftp.cerias.purdue.edu/pub/tools/unix/scanners/strobe/strobe/HISTORY (visited on 10/18/2021).

– (1999). *strobe(1): Super optimised TCP port surveyor - Linux man page*. URL: https://linux.die.net/man/1/strobe.

Barnett, Richard J and Barry Irwin (2008). “Towards a Taxonomy of Network Scanning Techniques”. In: *Proceedings of the 2008 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries: Riding the Wave of Technology*. SAICSIT ’08. Wilderness, South Africa: Association for Computing Machinery, pp. 1–7. ISBN: 9781605582863. DOI: 10.1145/1456659.1456660.

Blyth, Andrew (1999). “Footprinting for intrusion detection and threat assessment”. In: *Information Security Technical Report* 4.3, pp. 43–53. ISSN: 1363-4127. DOI: https://doi.org/10.1016/S1363-4127(99)80078-8.

Chumachenko, K and Dmytro Chumachenko (2017). “Study of Snort performance in counteracting port scanning techniques”. In: *Ukrainian Scientific Journal of Information Security* 23, no. 1, pp. 15–18.

Dabbagh, Mehiar et al. (2011). “Slow port scanning detection”. In: *2011 7th International Conference on Information Assurance and Security (IAS)*. IEEE, pp. 228–233.

Day, David and Benjamin Burns (2011). “A performance analysis of snort and suricata network intrusion detection and prevention engines”. In: *Fifth international conference on digital society, Gosier, Guadeloupe*, pp. 187–192.

Daya, Bhavya (2013). “Network security: History, importance, and future”. In: *University of Florida Department of Electrical and Computer Engineering* 4.

Durumeric, Zakir, Eric Wustrow, and J Alex Halderman (2013). “ZMap: Fast Internet-wide scanning and its security applications”. In: *22nd {USENIX} Security Symposium ({USENIX} Security 13)*, pp. 605–620.

FBI (2019). *Morris Worm — Federal Bureau of Investigation*. (Visited on 10/19/2021).

Fekolkin, Roman (2015). “Intrusion detection & prevention system: overview of snort & suricata”. In: *Internet Security, A7011N, Lulea University of Technology*, pp. 1–4.

Gallopeni, Getoar et al. (2020). “A Practical Analysis on Mirai Botnet Traffic”. In: *2020 IFIP Networking Conference (Networking)*. IEEE, pp. 667–668.

Gerich, Elise P. (May 1993). *Guidelines for Management of IP Address Space*. RFC 1466. DOI: 10.17487/RFC1466.

Higgins, Melissa (2011). *Julian Assange: WikiLeaks Founder: Wiki Leaks Founder*. ABDO.

Holm, Hannes et al. (2011). “A quantitative evaluation of vulnerability scanning”. In: *Information Management & Computer Security*.

Hoque, Nazrul et al. (2014). “Network attacks: Taxonomy, tools and systems”. In: *Journal of Network and Computer Applications* 40, pp. 307–324.

Houmz, Abdellah et al. (2021). “Detecting the impact of software vulnerability on attacks: A case study of network telescope scans”. In: *Journal of Network and Computer Applications* 195, p. 103230. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2021.103230.

Hutchins, Eric M, Michael J Cloppert, Rohan M Amin, et al. (2011). “Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains”. In: *Leading Issues in Information Warfare & Security Research* 1.1, p. 90.

Inc., Dyad Security (2004). *DYAD Security*. URL: https://web.archive.org/web/20041207034549/http://www.dyadsecurity.com/s_unicornscan.html (visited on 10/21/2021).

Jacobsen, Ørjan et al. (Apr. 2021). *FenrisBox*. URL: https://github.com/Team-Fenris/tfcctrl (visited on 05/20/2022).

Jacobsen, Ørjan Alexander (May 2022). *Framework for synthetic packet generation and characterization of Nmap scans*. URL: https://github.com/orjanj/nmap-pkg-generation-analysis (visited on 05/23/2022).

Jammes, Z and M Papadaki (2013). “Snort ids ability to detect nmap and metasploit framework evasion techniques”. In: *Adv. Commun. Comput. Netw. Secur* 10, p. 104.

Leonard, Derek et al. (2012). “Stochastic analysis of horizontal ip scanning”. In: *2012 Proceedings IEEE INFOCOM*. IEEE, pp. 2077–2085.

Liu, Jun and K. Fukuda (2014). “Towards a taxonomy of darknet traffic”. In: *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pp. 37–43.

Liu, Jun and Kensuke Fukuda (2018). “An Evaluation of Darknet Traffic Taxonomy”. In: *Journal of Information Processing* 26, pp. 148–157. DOI: 10.2197/ipsjjip.26.148.

Lockheed Martin (2015). *Gaining the advantage*. Tech. rep. URL: https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/Gaining_the_Advantage_Cyber_Kill_Chain.pdf (visited on 10/11/2021).

Lyon, Gordon Fyodor (2009). *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Sunnyvale, CA, USA: Insecure. ISBN: 0979958717.

Matherly, John (2015). “Complete guide to shodan”. In: *Shodan, LLC (2016-02-25)* 1.

Mazel, Johan, Romain Fontugne, and Kensuke Fukuda (2014). “A taxonomy of anomalies in backbone network traffic”. In: *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pp. 30–36. DOI: 10.1109/IWCMC.2014.6906328.

Microsoft (2021). *System Error Codes (0-499) (WinError.h) - Win 32 apps — Microsoft Docs*. URL: https://docs.microsoft.com/en-us/windows/win32/debug/system-error-codes--0-499- (visited on 05/15/2022).

Mitnick, Kevin D and William L Simon (2009). *The art of intrusion: the real stories behind the exploits of hackers, intruders and deceivers*. John Wiley & Sons.

O'Harrow Jr, Robert (June 5, 2012). "Search engine exposes industrial-sized dangers". In: *The Sydney Morning Herald*. URL: https://www.smh.com.au/technology/search-engine-exposes-industrialsized-dangers-20120604-1zrnw.html (visited on 10/24/2021).

Patel, Satyendra Kumar and Abhilash Sonker (2016). "Internet Protocol Identification Number Based Ideal Stealth Port Scan Detection Using Snort". In: *2016 8th International Conference on Computational Intelligence and Communication Networks (CICN)*, pp. 422–427. DOI: 10.1109/CICN.2016.89.

Petersen, R. (2020). *Ubuntu 20.04 LTS Desktop: Applications and Administration*. Surfing Turtle Press. ISBN: 9781949857115.

Pinkard, Becky and Angela Orebaugh (2008). *Nmap in the enterprise: your guide to network scanning*.

Postel, J. (Sept. 1981). *Internet Control Message Protocol*. RFC 792. DOI: 10.17487/RFC0792. URL: https://www.rfc-editor.org/info/rfc792.

Rogers, Russ (2011). *Nessus network auditing*. Elsevier.

Sharma, Gaurav et al. (2021). "Analysis and Implementation of Threat Agents Profiles in Semi-Automated Manner for a Network Traffic in Real-Time Information Environment". In: *Electronics* 10.15. ISSN: 2079-9292. DOI: 10.3390/electronics10151849. URL: https://www.mdpi.com/2079-9292/10/15/1849.

Singh, Abhinav (Feb. 2018). *Metasploit Penetration Testing Cookbook - Third Edition*. Packt Publishing Ltd. ISBN: 9781788623179.

Team, Jupyter (2015). *The Jupyter Notebook - Jupyter Notebook 6.4.11 documentation*. URL: https://jupyter-notebook.readthedocs.io/en/stable/notebook.html (visited on 05/16/2022).

Thapa, Suman and Akalanka Mailewa (2020). "The Role of Intrusion Detection/Prevention Systems in Modern Computer Networks: A Review". In: *Conference: Midwest Instruction and Computing Symposium (MICS)*. Vol. 53, pp. 1–14.

Treurniet, Joanne (2011). "A Network Activity Classification Schema and Its Application to Scan Detection". Undetermined. In: *IEEE/ACM Transactions on Networking* 19.5, pp. 1396–1404. DOI: 10.1109/TNET.2011.2109009.

A

Short Paper

This appendix contains the short paper published as a part of the research process.

Automating synthetic generation and capture of network scanning packet captures

Ørjan Alexander Jacobsen
Cyber Security
Noroff University
Bodø, Norway
oaj@oaj.guru

***Abstract*—Today the lack of early warning scanning detection is a problem needed to be mitigated. This article will explain the evolution and historical background of network scanning methods from the early internet age until today. People today use network scanning for multiple purposes; conducting audits towards their own manageable infrastructure, researching purposes, conducting cyber attacks, or autonomous malicious activity. The importance of the knowledge of how a network scanner works and various types of scans will be elaborated on within this research.**

Related to the pending security action being conducted, a base knowledge of which types of tools to conduct various tasks such as audit is crucial to mitigate risks of network scanning. Classifying network activity is an important step in mitigating future risks of cyberattacks. In order to detect such activity, synthetic data sets are needed to compare synthetic generated network scanning packet captures to real-world scanning data. This paper will focus on the setup of a virtual lab environment, where synthetic scanning data would be generated automatically through a tasking system and prepared for comparative analysis.

***Index Terms*—Classification, detection, nmap, network scanning**

I. Introduction

Scanning attacks are often applied to collect information about a network, mainly in the first step of the cyber kill chain known as the reconnaissance phase. The Lockheed Martin Cyber Kill Chain [1] is a framework used for identification and prevention of cyber intrusions activity, which is a part of the Intelligence Driven Defense model. The Cyber Kill Chain framework proves seven phases of a kill chain to accomplish a successful attack. Threats should ideally be detected within the first steps of the kill chain; reconnaissance, weaponization, and delivery [2]. The report distinguishes between two-phase detections; early and late phase detection. In the late phase, the detection is within the step of command and control, which is the 6th step in the kill chain. This means the threat actor already has delivered and installed malicious components on the target side. Meanwhile, in the early phase, the detection is noticed within the delivery phase, where threat actors attempt to deliver the malicious component towards a target.

Scanning is also a central part of penetration testing, where vulnerabilities are identified. During the phase of scanning and information gathering, activities such as port and vulnerability scanning are conducted as a normal thing for gathering intelligence.

Understanding patterns of attacks and threats are crucial to allow early detection of attacks [3]. Primarily detecting known scanners is an important step of both ongoing detecting scans and preventing cyber attacks. Analyzing amounts of data allows for gaining an understanding of patterns used in attacks [3]. The main objective of this research is to automate network scans in order to generate synthetic data sets, which in the future would optimize the method of collecting quality data for analysis purposes. The research outcome is expected to generate data sets containing synthetic data, which would be used as a step in the preparation process of signatures as a final research output.

This paper elaborates on the automatic generation of synthetic network scanning packet captures using various types of scan modes and templates using the Nmap scanner. By generating synthetic data sets, where the control of the tasks conducted against each worker host running packet capturing, the analysis of these data sets could be used in a comparative analysis to classify and detect various types of network scans. Following this method elaborated in the paper would allow synthetic data generation for use in a comparative analysis.

This paper is structured into sections elaborating related work in section II, the methodology of generating and collection in section III-B, and processing of the synthetic packet captures in a virtual lab environment. The needed software, operating systems, and packages are further elaborated in section III-A.

II. Related work

Network scanning is a reconnaissance activity conducted within the field of network intrusion [4]. During an intelligence-gathering step, network scanning is one of the first steps before an attack is about to happen [5]. Conducting a network scan would reveal information about hosts on a network, which includes running services and applications, open ports, and operating system details. The main techniques applied during network scanning are version and service detection, scanning ports, operating system detection, and network mapping [6]. During a network scan, devices and services vulnerable to attacks could be unveiled [5].

By conducting port scanning, information about running services and open ports could be retrieved. This could further be used to determine misconfiguration on a system or version

of the software, which could be mapped against known exploitable vulnerabilities. Multiple types of port scanning methods exist, such as TCP SYN and XMAS. Another interesting scan is an ICMP echo scan [6].

Port scanning is a reconnaissance technique used to discover services running on open ports which could be exploited to gain access to a system. Port scanning is primarily divided into vertically and horizontally scans [7].

In vertical attacks, multiple ports are scanned on one host. This is a common process when a threat actor is interested in one specific host to attack. Vertical scanning sends probes on various ports against one specific host on the network [4]. A vertical scan could be used to map out known vulnerabilities using a scanner to detect various operating system and service information, including versioning information.

Nmap is free software released in September 1997, mainly created for users to scan networks under security audits by system administrators and other IT professionals. It has a wide range of capabilities, such as port scanning a single host or a subnet, detecting operating system and service versioning information, and echo ping scans. The software has various usage areas, such as asset management, system, and network inventory, compliance testing, and security auditing. During audits, it could be useful to conduct OS versioning to detect the need for patching and updates. Since Nmap could usually be used for this purpose, it could also be used during harmful intrusions to map the infrastructure of a network. Furthermore, this could be used to exploit vulnerabilities within the network. Regarding types of scans that could be conducted, Nmap could use various timing templates to avoid detection, such as paranoid and sneaky mode. On the other side, Nmap can speed up a scan by applying aggressive or insane scan since these templates only are based on timings [6].

In cyberspace today, threats have become a major concern for businesses. There exist other security mechanisms than firewall and anti-virus that need to be implemented to increase security within a company's network [8]. Network scanning is a common reconnaissance activity [4], and the classification of a network scanner could be described. Furthermore, illustrating the complexity of the activity shows that over 96% of all anomalous events can be detected and labeled. These results would make the unlabelled source rate very low [9].

For the mission of detecting abnormal activity within a network, a taxonomy of anomalies could be created to detect such behavior. There exist multiple techniques for detection, such as creating signatures for pattern recognition statistical methods, among other things. These methods provide information that further could be used for anomaly detection. Traffic metrics could be used for detecting anomalies, such as scanning activity and DDoS. Labeling each occurring event could be applied to categorize and differentiate between abnormal traffic and normal traffic. Categorization of traffic could be separated into subcategories such as unknown, anomalies, or normal [10].

The observation of cyber attacks and anomalies has among security researchers enabled Liu and Fukuda to develop cyberspace monitoring approaches to detect anomalies. The study also shows that darknet provides effective passive monitoring approaches, which often refer to globally routable and unused IP addresses that are used to monitor unexpected incoming network traffic. This is an effective approach for viewing network security activities remotely. The main goal of the study was to create and propose a simple taxonomy of darknet traffic [11].

Mazel, Fontugne, and Fukuda documented anomalous events classified using data collected through six years [10]. The research points out that previously unknown events are now accountable for a substantial number of UDP network scans, scan responses, and port scans. With their proposed taxonomy, the number of unknown events has decreased from 20% to 10% of all events compared to the heuristic approach [10]. Such scans can be handled by the Nmap scanner, popularly used for administration, security auditing, and network discovery [6]. Subverting firewalls and IDS, optimization of Nmap performance, and automating common network tasks could also be done with the Nmap.

Scans are one type of event among others, which could be represented through characteristics and patterns. It might be several or single scanned hosts. Scans are conducted to gain knowledge about a target. This can be opened ports, which operating system is running, or versions of software that are using the open ports. In a network scan, hosts alive could be detected among services and ports open. During a host scan, the same results could be retrieved through checks towards the allowance of packets to a specific port. Network scans can be used to gather intelligence of how the network is structured, how many clients are connected, and which services at which version are located on the network. After gathering intelligence about clients in a network, the intelligence itself can tell something about which exploitation could be used against the various services at which specific version [10] during the delivery step in the cyber kill chain.

Various traffic characteristics during scanning would be caught, such as the following. During TCP scanning, SYN is used to initiate connection while ACK is mapping ports by filtering port answers through ICMP [10]. A signature is created with rules for different attributes of the traffic. Abnormal characteristics such as scanning attempts can be detected and generate a signature out of the behavior of the traffic recently conducted [10].

While identifying scanning techniques, a quantitative amount of data is required to conduct identification and evaluation of scans conducted. During the research of [4] the port scanning tool, Nmap was used to construct the scanning, and tcpdump was used to capture packets on the network interface of a FreeBSD host.

During research by Jammes and Papadaki [12], Snort was able to detect five out of six of the different timing templates used for Nmap, which is default delivered by Nmap. The last template, the paranoid template, was Snort not able to detect, according to the study. These templates are named insane, aggressive, normal, polite, sneaky, and paranoid. The difference between these is the minutes waited between sending a probe

to the target host. During the paranoid mode, Nmap waits 5 minutes, while at the sneaky mode, it waits for 15 seconds [6], [12]. During a scan execution, the parameter $-T$ could be used to set the template. The higher the number is, the faster the scan are [6].

During a Nmap scan, other parameters could also be applied, such as the Time-to-Live parameter. Another method for avoiding detection is to randomize the order hosts are scanned. Nmap has capabilities for sending UDP and TCP packets with invalid checksums, which often firewalls and other security controls drop due to the fact the checksum isn't checked. If a reply is given, these are often from firewalls or other security controls.

III. Generating synthetic data

To conduct this research, a virtual lab environment was needed to segment captured scanning traffic from real internal network traffic to ensure the validity of the synthetic generated packet capture data, meanwhile having the capability of managing each worker host. The importance of generating synthetic data is to have a clean data set, with minimal noise, for comparison against real-world data containing a substantial amount of noise traffic.

A. Lab environment

VMware workstation 16.1 was used to set up 20 virtual hosts (referred to as workers) using Ubuntu 20.04 (Linux kernel 5.4.0-81-generic #91) with the only target of capturing packet captures from the scanner host. One virtual host (referred to as scanner host) were installed with Kali Linux 2021.4 (Linux kernel 5.14.0-kali4-amd64 #1) for conducting various type of scans, issuing tasks to the workers, and monitoring ongoing scans.

Two closed internal virtual networks are set up for the purpose of data generation and management. This is for the production of raw packet captures in accordance with generating comparative data sets, which could be used for comparison between the generated synthetic data and data in the wild. Through such a comparison, these synthetic data sets could be used to classify various types of scanners and scanning types. The chosen IPv4 unicast address block for virtual network 1 (referred to as "scanning network") is $192.168.2.0/24$. This address block is reserved for testing and is enlisted as TEST-NET-1 in RFC 5737 [13], which describes the usage of the network during this research. Each worker host has its increment asset name where the hostname is in accordance with the last two digits in the last IP octet (e.g., $bsc01$ resolving to $192.168.2.101$). The scanning host has a higher last IP octet to ensure no confusion and IP collision regarding the static allocation for worker hosts. Within this research, the scanning host has the address $192.168.2.230$. This address allocation logic only shows flaws when the number of workers is above 99. As this is a proof of concept and the low number of workers, this is not a problem.

Secondly, another virtual network is set up for the usage of managing worker hosts in the network. The main reason is to reduce noise traffic on the scanning network and to segment management of each worker, such as managing tasks, monitoring ongoing tasks, and retrieving raw packet capture data. Chosen IPv4 unicast address block for the virtual network 2 (referred to as "management network") is $194.100.10.0/24$. Within RFC 1466 [14] a class C network in the address space $194.0.0.0$ - $195.255.255.255$ is allocated for management use in Europe. The same logic for address allocation used in the scanning network is also applied to the management network, where the last IP octet represents each worker host. Both these networks are visualized in 1.

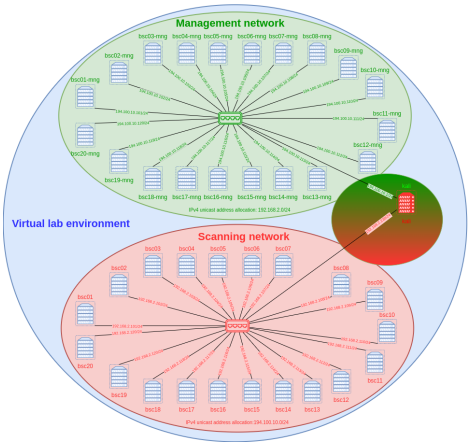


Fig. 1. Capture of network diagram for the virtual environment.

Another network characteristic is DHCP, which in both networks are disabled to ensure the logical structural setup of the network, such as static address allocation to each virtual host in accordance with a hostname. Basing the network setup on the RFC standards reduces the possibility for collision and confusion regarding usage of the networks [13], [14].

Through an initiation worker preparation script, there are implemented commands to reduce traffic noise affecting the scan results. This script disables certain services known for querying servers on the internet for the purpose of updating time through NTP, automatic operating system updates, and operating system package updates. This script must be run during the setup of the environment to ensure noise reduction. Another very important step during this initial configuration is change moving the tcpdump command to pecial ($chmod + s$), meaning a regular user is able to run tcpdump as a regular user. The command must, in this case, be executed by root. During the initial configuration of each worker, the SSH public key for the scanner host is added into the worker's authorized keys in $/.ssh/authorized\_keys$. It allows the scanner host to remotely access each worker with the use of its SSH private key to managing tasks for the worker through SSH. Each

worker has its own *bscadm* regular local user for this purpose. SSH listens only on the management network interface on each worker.

B. Data generation and collection

Management of each worker is conducted from the scanner host, as this is the host initiating the scanning tasks. The task issuing is conducted through a Bash script, running as a cron job on the scanner, iterating through given workers, and determining the first available worker. The task list is enlisted in a comma-separated file for enhanced readability for the user populating the task list as seen in figure 2.

```
~# cat comparative_data.csv | sed 's/,[0-9]*,/,/g' | sort -r | uniq | head -n 8
priority,task_name,task_staus,scanner,extra_args
3,nmap_xmas_scan_polite,completed,nmap,"-T2 -sX"
3,nmap_tcp_full_scan_polite,completed,nmap,"-T2 -sT"
3,nmap_svc_discovery_polite,completed,nmap,"-T2 -sV"
3,nmap_ping_scan_polite,new,nmap,"-T2 -PE"
3,nmap_ping_scan_polite,completed,nmap,"-T2 -PE"
3,nmap_os_svc_discovery_polite,completed,nmap,"-T2 -A"
3,nmap_null_scan_polite,completed,nmap,"-T2 -sN"
```

Fig. 2. Capture of task list.

Given task files are iterated until a task with the status new is found. From this point, the scanner host deploys a task to the worker through the management network, commanding it to dump traffic on the incoming interface pointed to the scanning network. When this task is successfully deployed, the scanner hosts start the scan on the scanning network towards the worker with the given scanning characteristics. As a default, using the Nmap scanner, the result of the scan from the scanner host is outputted to an XML result file represented with the given task name. The status for the representative task in the task file is changed from *new* to *ongoing*. This operation is iterated through all workers until all workers are busy when the scanning host cannot deploy more tasks until one or more of the workers are available for work again. A capture of this is shown in 3.

```
bsc07-mng is busy (error code: 1).
[ Success ] bsc08-mng is available.
[ Success ] Capturing traffic on bsc08-mng on task nmap_fin_scan_insane
tcpdump: listening on ens33, link-type EN10MB (Ethernet), capture size 262144 bytes
[ Success ] Scanning bsc08-mng on task nmap_fin_scan_insane
[ Success ] bsc09-mng is available.
[ Success ] Capturing traffic on bsc09-mng on task nmap_null_scan_insane
tcpdump: listening on ens33, link-type EN10MB (Ethernet), capture size 262144 bytes
[ Success ] Scanning bsc09-mng on task nmap_null_scan_insane
[ Success ] bsc10-mng is available.
```

Fig. 3. Capture of task allocation.

Aside from this, a task processing script is running to maintain the task list with the correct given status of tasks. This meaning is only comparing running scan processes, identified by task name on the scanning host, against running tcpdump tasks on the given worker. When this differs, and only the tcpdump process together with the task name are found in the process list, it terminates the tcpdump on the worker and updates the task status in the task list.

A monitor script is also created to monitor the ongoing tasks. This monitor updates every 2 seconds, capturing scanning specific tasks and tcpdump tasks returning data for worker corresponding with task name, shown in figure 4.

Worker_Host	Process_Name	Task
bsc01	nmap	nmap_fin_scan_paranoid
bsc01-mng	tcpdump	nmap_fin_scan_paranoid
bsc02	nmap	nmap_null_scan_paranoid
bsc02-mng	tcpdump	nmap_null_scan_paranoid
bsc03	nmap	nmap_os_svc_discovery_paranoid
bsc03-mng	tcpdump	nmap_os_svc_discovery_paranoid
bsc04	nmap	nmap_ping_echo_scan_paranoid_1
bsc04-mng	tcpdump	nmap_ping_echo_scan_paranoid_1
bsc05	nmap	nmap_svc_discovery_sneaky_7
bsc05-mng	tcpdump	nmap_svc_discovery_sneaky_7
bsc06	nmap	nmap_svc_discovery_paranoid
bsc06-mng	tcpdump	nmap_svc_discovery_paranoid
bsc07	nmap	nmap_tcp_full_scan_paranoid
bsc07-mng	tcpdump	nmap_tcp_full_scan_paranoid

Fig. 4. Capture of scan monitoring tool

Other useful tools are developed, such as the task populator. The populator iterates through a given task file containing tasks that wanted to be run a number of times in order to generate a comparable synthetic data set in the end. An incrementing number is added to the task name (e.g., $nmap\_xmas\_scan\_1$) to uniquely identify a task when iterated through by the scan deployment stage.

A simplistic collection of captured packet captures run through a retrieve script on the scanning host, which iterates through each worker host matching a specific filter (e.g., task name) and remotely synchronizes packet captures to the local output directory on the scanning host.

IV. Data processing

One of the components developed in this research is the packet capture parser. Inputs to this parser are the result directory containing packet captures captured by each worker during each separate scan task, and also an IP ignore list to filter out unwanted IP addresses. The retrieval component synchronizes packet captures to an output directory given by an input parameter. To keep a structure to separate each task from each other for use in the analysis, the directory tree structure for the retrieved packet capture is shown in figure 5.

```
$ tree -L 1 .
.
├── nmap_fin_scan_aggressive
├── nmap_fin_scan_insane
├── nmap_fin_scan_normal
├── nmap_fin_scan_polite
├── nmap_null_scan_aggressive
├── nmap_null_scan_insane
├── nmap_null_scan_normal
├── nmap_null_scan_polite
├── nmap_os_svc_discovery_aggressive
├── nmap_os_svc_discovery_insane
├── nmap_os_svc_discovery_normal
├── nmap_os_svc_discovery_polite
├── nmap_ping_scan_aggressive
├── nmap_ping_scan_insane
├── nmap_ping_scan_normal
├── nmap_ping_scan_polite
├── nmap_svc_discovery_aggressive
├── nmap_svc_discovery_insane
├── nmap_svc_discovery_normal
├── nmap_tcp_full_scan_aggressive
├── nmap_tcp_full_scan_insane
├── nmap_tcp_full_scan_normal
├── nmap_xmas_scan_aggressive
├── nmap_xmas_scan_insane
└── nmap_xmas_scan_normal
```

Fig. 5. Capture of tree directory structure for retrieved packet captures.

The parser then iterates through the given result directory identifying each respective task by name and the respective files related to the task. Scapy is used to read each representative packet capture into an object. Furthermore, header fields are defined for IP, TCP, UDP, and ICMP header, which are directly written to a file, each resulting in 3 files (TCP, UDP, and ICMP), with the pattern enlisted as *protocol_taskname.csv*. Each packet is iterated through, where only IP packets are processed. The parser creates a list of each packet during the

iteration and detects the protocol before writing the packet details into its representative CSV file. Another Python library used during iteration is the binascii library, which decodes the payload to hex. A comparison of the raw packet capture and the parsed CSV file is shown in figure 6.

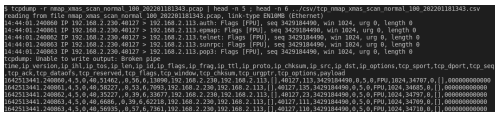


Fig. 6. Capture of comparison of raw packet capture and parsed CSV output.

V. Preliminary results

Central components from this research are the automated scanner, task population, task monitor tool, task managing tool, packet capture retrieval tool, and packet capture parser. Each of these components has its own feature.

The automated scanner reads through the task list, determines available workers, and issues tasks to the representative worker for dumping packet captures meanwhile running a scan against the worker. In detail descriptions is found in section III-B.

Developed code using Python to iterate through given directories to identify parsable packet capture files. This parser outputs the relevant data into three separate files, each segmented into the protocol, details described in section IV.

Other smaller components were created during this research for the purpose of monitoring, maintenance, task management, worker preparations, and debugging. The task processor monitors ongoing tasks using the status in the task list and process checking. Monitoring processes were important during debugging, and ongoing scans and is one useful component for checking status for users using the component. During the population of tasks, a useful component to use is the task populator which iterates through the task list and adds an incrementing number to clone a given task in the task list to uniquely identify a task. Lastly, a retrieval component was developed to easily retrieve packet captures from each worker automatically. All these components are further described in section III-B.

The preliminary results are the development of code, and the creation of a virtual environment for generating amounts of synthetic packet captures in a cost-effective way. This means more tasks could be applied to the lab for either other tasks using Nmap or introducing other scanners. During the development, scalability was a priority to be able to increase the number of workers and introduction of other network scanners. During the conducted scans, packet captures for the modes aggressive, insane, polite, and normal were set to a limit of 100. These tasks have been issued two times, the first time with too much background noise, which needed to be eliminated fully. From here, a platform walkthrough eliminating such noise was conducted.

The packet captures were generated in the following timeframes seen in table I.

TABLE I
Packet capture time frames and counts

Amount	Started	Finished	Timing template
100	Jan 18 02:57	Jan 18 10:38	aggressive
100	Jan 18 02:24	Jan 18 11:05	insane
100	Jan 18 11:05	Jan 18 14:44	normal
100	Jan 18 14:51	Jan 18 23:22	polite
100	Jan 18 04:02	Jan 18 08:45	sneaky
10	Jan 18 04:01	Ongoing Jan 20 22:08	paranoid

VI. Conclusion and further work

This research shows the importance of generating quality synthetic data for the use of a comparative analysis where the content is familiar when it comes to the analysis. If there exist patterns able to grow doubt of what the contents really are, the data are not of good quality for conducting a comparative analysis to identify various types of scans in a later stage. The importance of filtering away noise traffic not relevant for the analysis to generate a high quality is experienced. Having high-quality data early in the stage of developing a signature for given traffic results in a high-quality signature.

Other key aspects are the availability of a lab environment to generate quality packet captures and a worker setup that could be used for other tasks than only scanning conducted in this research. A fully automated setup enables, in certain scenarios, further tasking and an increased amount of packet generation.

Further work within the research is to conduct an analysis of the generated and processed packet captures. For the analysis, various Python modules have been researched to determine the way ahead and how to be able to identify patterns for each various scan. Through an analysis, patterns and other relevant packet characteristics could be extracted. In this research, the parser has converted packet capture files to comma-separated value files, which with the use of the Python library, pandas could structure a specific scan type with a given timing template into a larger data frame for analysis purposes. Matplotlib could be used to plot data into visualized compiled data to extract the most interesting findings.

After an analysis, the creation of IDS signatures is a planned step further in the process to output a signature as a final goal of the research.

References

[1] Lockheed Martin, “Gaining the advantage,” Lockheed Martin, Tech. Rep., 2015. [Online]. Available: https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/Gaining_the_Advantage_Cyber_Kill_Chain.pdf

[2] E. M. Hutchins, M. J. Cloppert, R. M. Amin *et al.*, “Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains,” *Leading Issues in Information Warfare & Security Research*, vol. 1, no. 1, p. 90, 2011.

[3] W. Acosta, “Large-scale analysis of continuous data in cyber-warfare threat detection,” in *The Proceedings of the 6th International Conference on Information Warfare and Security*, 2011, p. 317.

[4] R. J. Barnett and B. Irwin, "Towards a taxonomy of network scanning techniques," in *Proceedings of the 2008 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries: Riding the Wave of Technology*, ser. SAICSIT '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 1–7. [Online]. Available: https://doi.org/10.1145/1456659.1456660

[5] A. Houmz, G. Mezzour, K. Zkik, M. Ghogho, and H. Benbrahim, "Detecting the impact of software vulnerability on attacks: A case study of network telescope scans," *Journal of Network and Computer Applications*, vol. 195, p. 103230, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804521002290

[6] B. Pinkard and A. Orebaugh, "Nmap in the enterprise: your guide to network scanning," 2008.

[7] M. Dabbagh, A. J. Ghandour, K. Fawaz, W. El Hajj, and H. Hajj, "Slow port scanning detection," in *2011 7th International Conference on Information Assurance and Security (IAS)*. IEEE, 2011, pp. 228–233.

[8] R. Fekolkin, "Intrusion detection & prevention system: overview of snort & suricata," *Internet Security, A7011N, Lulea University of Technology*, pp. 1–4, 2015.

[9] J. Liu and K. Fukuda, "Towards a taxonomy of darknet traffic," *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pp. 37–43, 2014.

[10] J. Mazel, R. Fontugne, and K. Fukuda, "A taxonomy of anomalies in backbone network traffic," in *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2014, pp. 30–36.

[11] J. Liu and K. Fukuda, "An evaluation of darknet traffic taxonomy," *Journal of Information Processing*, vol. 26, pp. 148–157, 2018.

[12] Z. Jammes and M. Papadaki, "Snort ids ability to detect nmap and metasploit framework evasion techniques," *Adv. Commun. Comput. Netw. Secur*, vol. 10, p. 104, 2013.

[13] J. Arkko, M. Cotton, and L. Vegoda, "IPv4 Address Blocks Reserved for Documentation," RFC 5737, Jan. 2010. [Online]. Available: https://rfc-editor.org/rfc/rfc5737.txt

[14] E. P. Gerich, "Guidelines for Management of IP Address Space," RFC 1466, May 1993. [Online]. Available: https://rfc-editor.org/rfc/rfc1466.txt

B

Virtual machine creation and OS installation

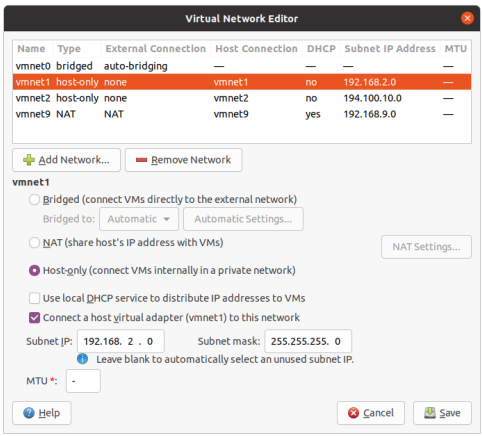
This appendix describes the installation guide of each worker including virtual machine setup and operating system installation.

Ubuntu 20.04 Installation guide

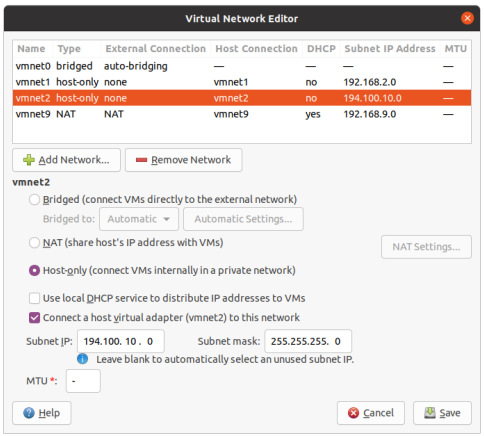
Jacobsen, Ørjan Alexander

Content: Guide for installing Ubuntu 20.04 for this research

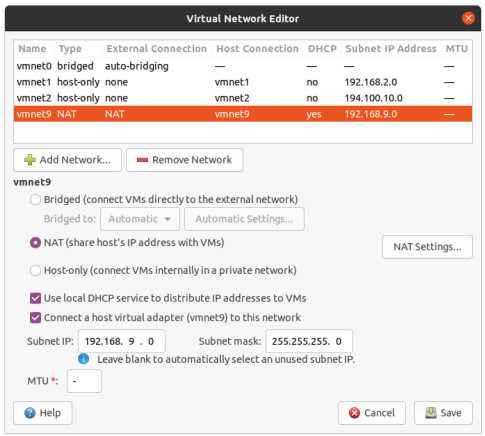
1 SETTING UP VIRTUAL NETWORK IN VMWARE WORKSTATION



(a) vmnet1 - Scanning network



(b) vmnet2 - Management network



(c) vmnet9 - Internet connectivity

Figure 1.1: Virtual network settings

To set up virtual networks in VMware Workstation, open the VIRTUAL NETWORK EDITOR which is found in under **Edit** in the Menu bar. As seen in figure 1.1, there exists four virtual networks for this setup. **vmnet0** is not to be configured for this research and would stay as default configured.

vmnet1 must be configured for the SCANNING NETWORK with the following settings, as seen in figure 1.1a:

- **Host-only**
- **Do not check the box for** *Use local DHCP service to distribute IP addresses to VMs*
- **Connect to a host virtual adapter (vmnet1) to this network**:
  - **Subnet IP:** 192.168.2.0
  - **Subnet mask:** 255.255.255.0

vmnet2 must be configured for the MANAGEMENT NETWORK with the following settings, as seen in figure 1.1b:

- **Host-only**
- **Do not check the box for** *Use local DHCP service to distribute IP addresses to VMs*
- **Connect to a host virtual adapter (vmnet2) to this network**:
  - **Subnet IP:** 194.100.10.0
  - **Subnet mask:** 255.255.255.0

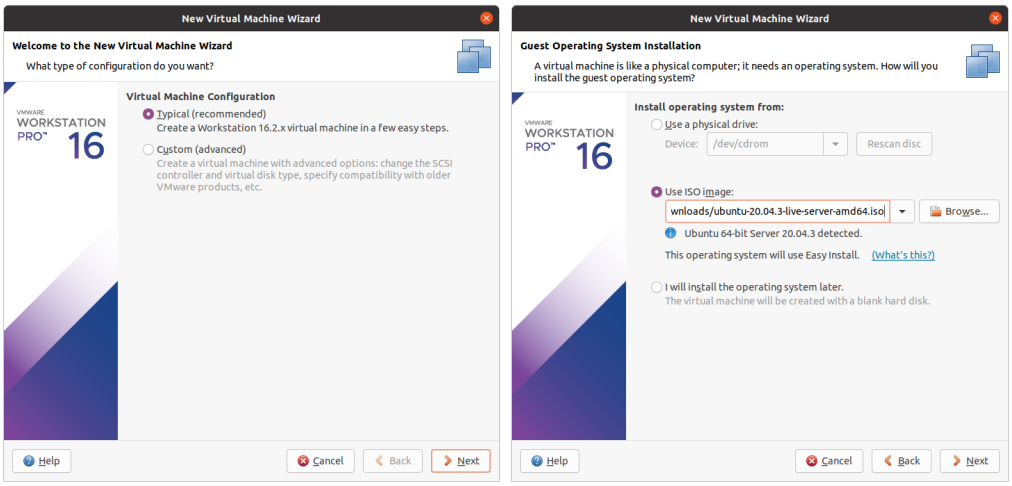
vmnet9 must be configured for the INTERNET CONNECTIVITY with the following settings, as seen in figure 1.1c:

- **Host-only**
- **Check the box for** *Use local DHCP service to distribute IP addresses to VMs*
- **Connect to a host virtual adapter (vmnet9) to this network**:
  - **Subnet IP:** 192.168.9.0
  - **Subnet mask:** 255.255.255.0

The **vmnet9** interface are to be used in rare use-cases in the Kali Linux installation, which could be activated **only** if needed to install packages and updates.

2 CREATING VIRTUAL MACHINE IN VMWARE WORKSTATION

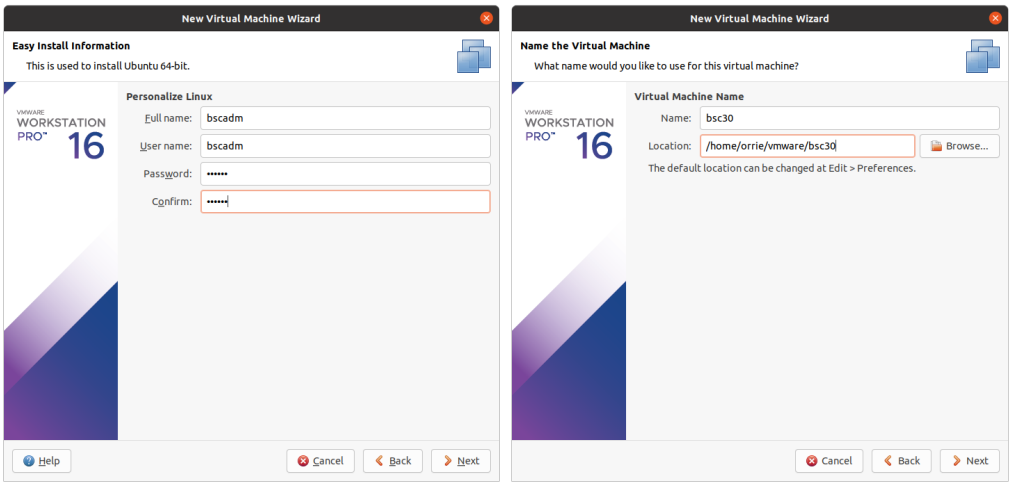
In the main window of VMware Workstation, press **File** and **New Virtual Machine** in the Menu Bar. Alternatively the hotkey $CTRL + N$ could be used. Go through the guide as this steplist describes.



(a) VM configuration (b) ISO image

Figure 2.1: VM configuration and ISO image

Choose **Typical** in figure 2.1a and press **Next**.
Choose the ISO image for Ubuntu 20.04 as in figure 2.1b and press **Next**.

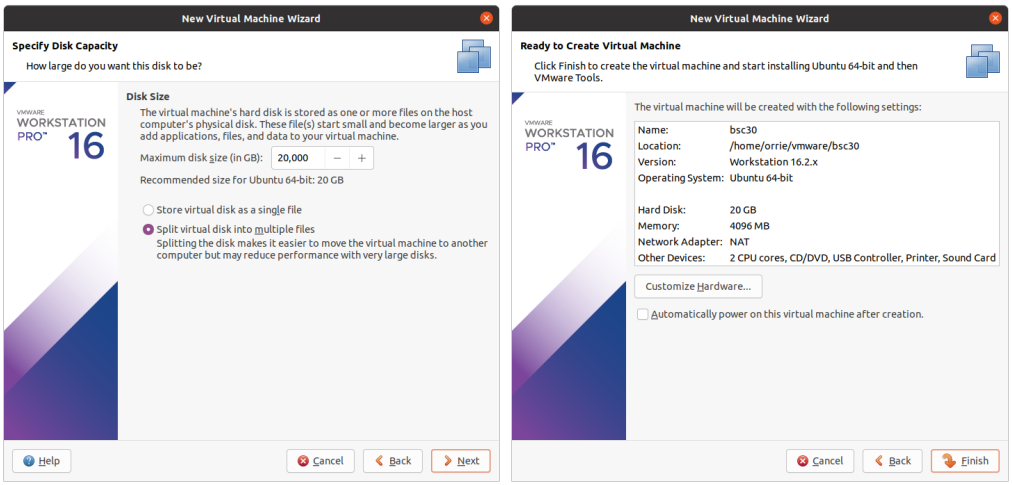


(a) Username and password

(b) Hostname

Figure 2.2: Username, password and hostname

Insert the username and password for the user like in figure 2.2a. Within this research the default user **bscadm** are used together with the password the same as the username, for simplistic reasons. Press **Next**.
Name the virtual machine (naming convention for the research is $bscXY$ where XY symbolises increment number from 01 to 20) as shown in figure 2.2b and choose location for the machine to be stored. Press **Next**.

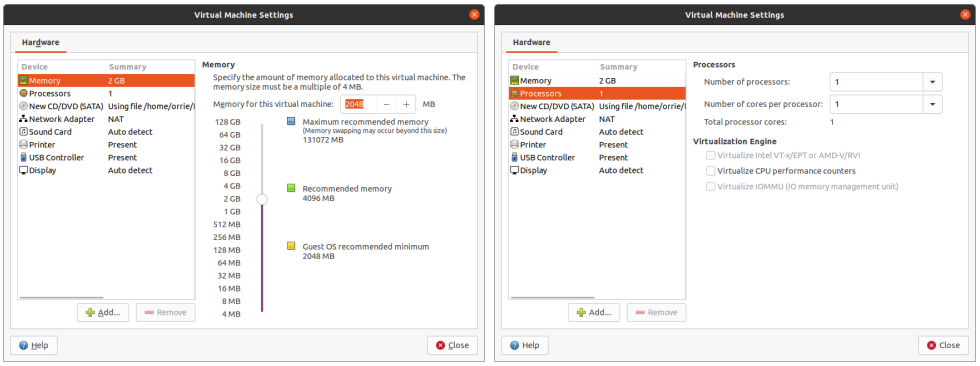


(a) Disk size

(b) VM summary

Figure 2.3: Disk size and VM summary

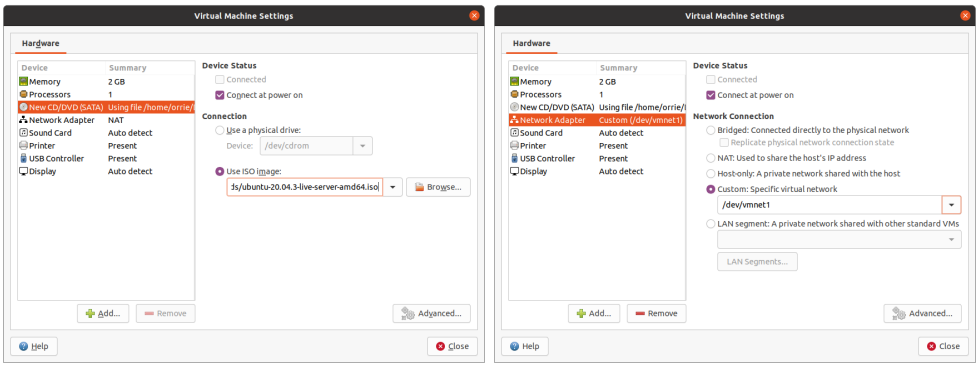
Choose disk size (default **20GB**) and press **Next**.
Look at the summary and press **Next** if as shown in figure 2.3b.



(a) Memory settings (b) Processor settings

Figure 2.4: Memory and processor settings

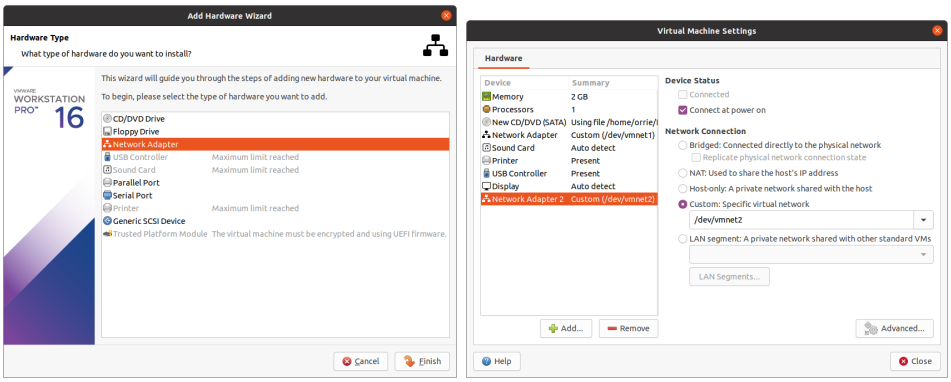
Choose the virtual machine and press **Edit Virtual Machine Settings**. Could be done using the Menu Bar by pressing **VM** and **Settings**.
Configure memory and processors as shown in figure 2.4.



(a) ISO image (b) Network adapter 1 settings

Figure 2.5: ISO image and network adapter 1

Check the settings against figure 2.5. The ISO image for the Ubuntu installation must be chosen and the **Network adapter 1** must be connected and set to use **Custom /dev/vmnet1**.

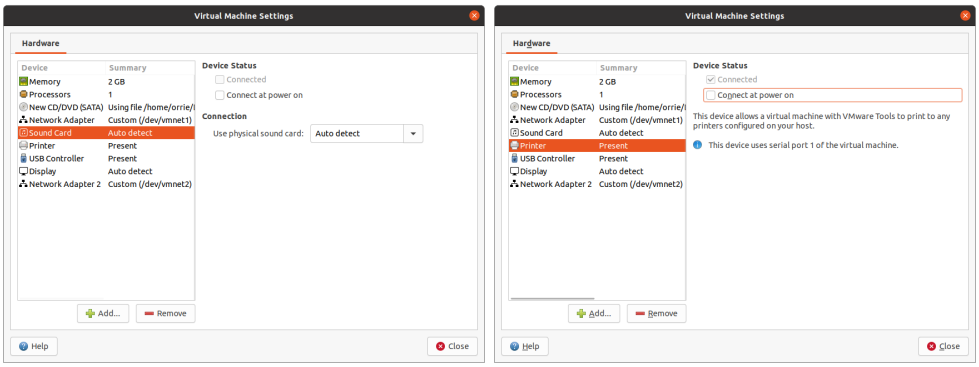


(a) Add new hardware (b) Network adapter 2 settings

Figure 2.6: Add new network adapter and configure

Press **Add** and chose **Network adapter** and press **Finish** as shown in figure 2.6.

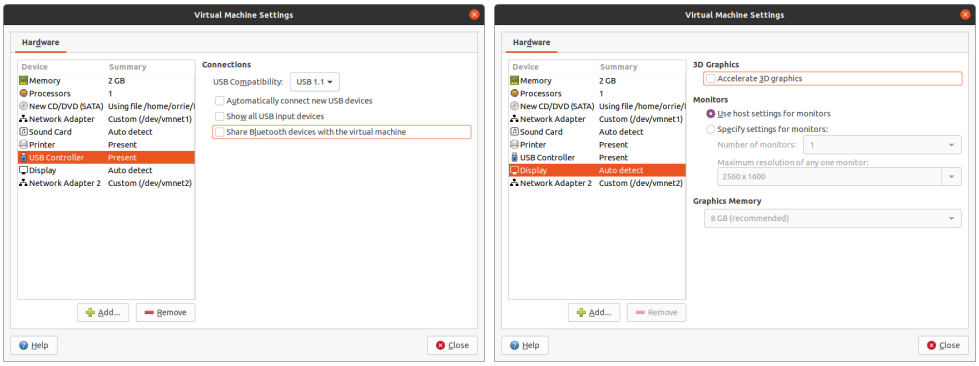
The **Network adapter 2** must be configured to **Connected at power on** and set to **Custom /dev/vmnet2**.



(a) Sound card settings (b) Printer settings

Figure 2.7: Sound card and printer settings

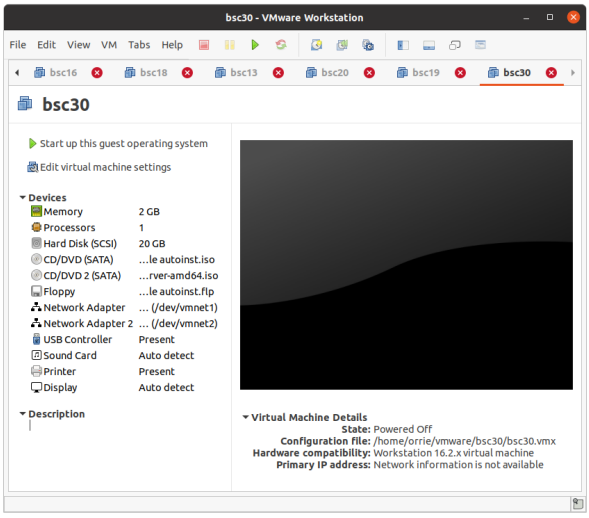
The sound card and printer do not need to be connected as shown in 2.7.



(a) USB settings (b) Display settings

Figure 2.8: USB controller and display settings

Disable all settings on the USB controller and Display as shown in figure 2.8.



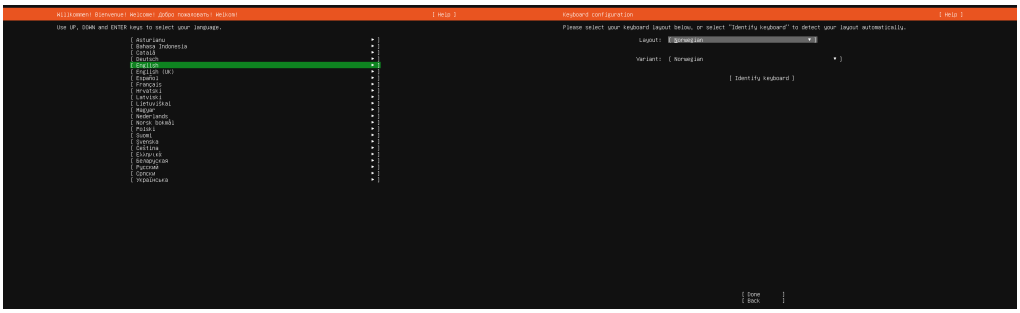
(a) New VM settings summary

Figure 2.9: New VM settings summary

Check the settings for the virtual machine, it should be similar as in figure 2.9.

3 INSTALLING UBUNTU 20.04

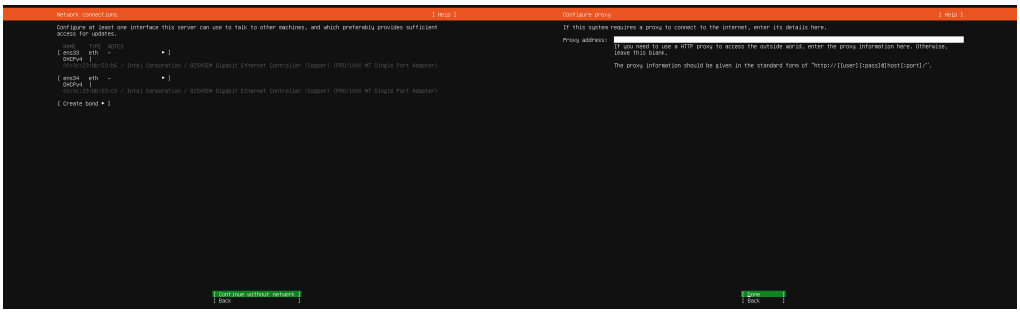
Now, **Power on** the virtual machine and a installation is prompted. Continue with the steps as follows.



(a) Language settings (b) Keyboard settings

Figure 3.1: Language and keyboard settings

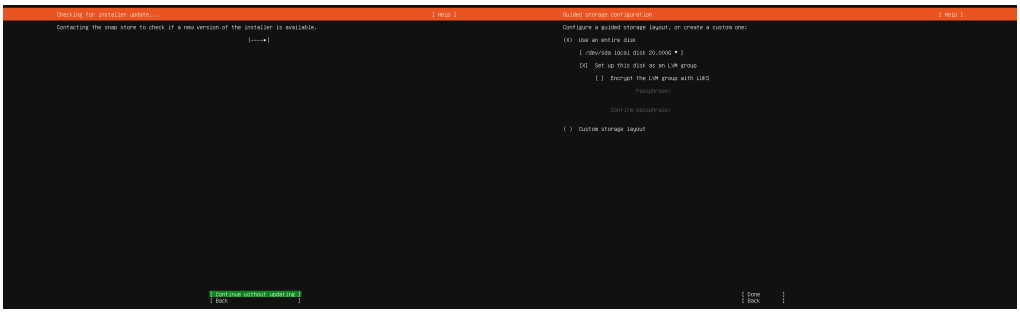
Choose **English** language and preferred keyboard layout.



(a) Network settings (b) Proxy settings

Figure 3.2: Network and proxy settings

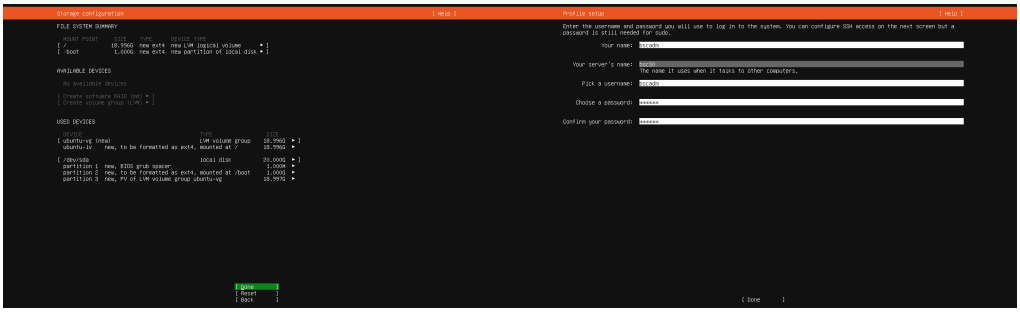
No need for configurating network now. Proceed, same with proxy settings.



(a) Snap settings (b) Storage settings

Figure 3.3: Snap and storage settings

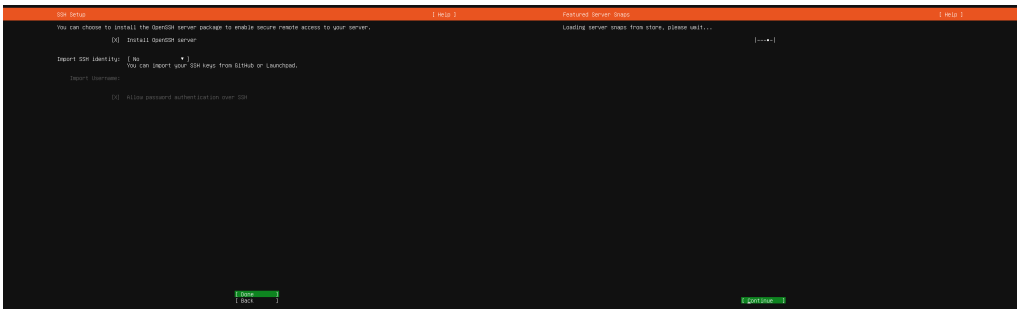
Default settings for Snap seems fine, continue. Same with the disk settings, will use the whole disk.



(a) Disk settings (b) Users settings

Figure 3.4: Disk and users settings

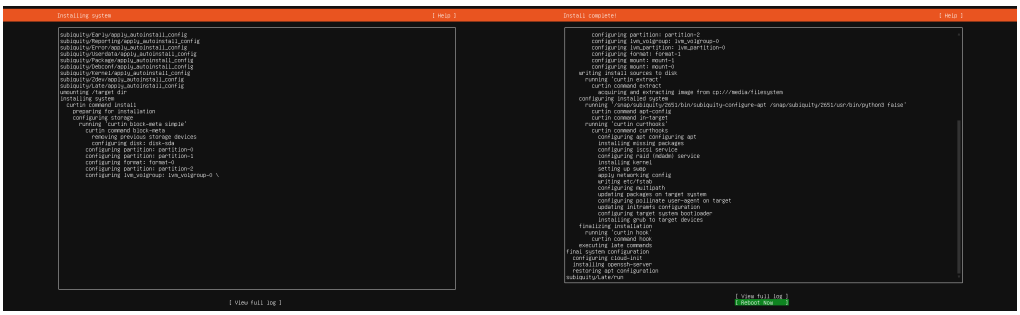
The default partitioning settings seems fine, continue.
Within the user fields preferred username and password are entered.
For this research is the **bscadm** used, and must be used. The password are set to the same as username for simplistic reasons.



(a) SSH settings (b) Snap

Figure 3.5: SSH and snap

Check the box for installing OpenSSH server since the worker are managed through SSH. Continue from Snap.



(a) Installing (b) Install done

Figure 3.6: Installation

An installation prompt will now be shown, and when this completes reboot the OS.

C

Electronic resources

This appendix links to the Github repository[1] where the developed code for this research project is located (Ø. A. Jacobsen 2022). The Jupyter notebook, earlier discussed in section 4.4.1, are published in this repository. Located within the same repository is the scripts described in section 3.4 and the packet capture parser described in 4.3.1.

[1] https://github.com/orjanj/nmap-pkg-generation-analysis