


Technische Universität München
Department of Informatics

Bachelor's Thesis in Informatics

Efficient scans in a research network

Nils Mäurer

Technische Universität München

Department of Informatics

Bachelor's Thesis in Informatics

Efficient scans in a research network

Effiziente Scans in einem Forschungsnetzwerk

Author	Nils Mäurer
Supervisor	Prof. Dr.-Ing. Georg Carle
Advisor	Dr. Ralph Holz, Dipl.-Math. Felix von Eye, Oliver Gasser, M.Sc.
Date	15th February 2015

Informatik VIII
Chair for Network Architectures and Services

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

Garching b. München, 15th February 2015

Signature

Acknowledgements

First and foremost I want to thank my research advisors Ralph Holz, Felix von Eye and Oliver Gasser for their support and dedicated involvement. Without the many telephone conferences, even to Australia ;), meetings and the very good feedback regarding scientific research and writing a thesis, this work would not have been possible.

I would also like to express my gratitude to my supervisor Prof. Dr.-Ing. Georg Carle, especially for his trust in me as I could suggest and research the SSL vulnerability part (POODLE) in the thesis.

First of all persons helping me with this thesis not directly linked to the TUM or the LRZ, I want to mention my girlfriend Alina Meixl. Thank you so much for everything you are to me and for what you helped me with.

Of course without family support, this would not have turned out to be a complete thesis, but rather an essay about scientific facts. So first of all I want to thank my father for providing the necessary background about academic writing. Also I want to thank my mother and my sister for their encouragement.

Last but not least, I want two mention two wonderful Canadian persons, my granduncle John and my grandaunt Bärbel, who helped me improving my English writing.

Abstract

The Leibniz Supercomputing Centre (LRZ) is the provider of Munich's largest research network, the Munich Scientific Network (MWN). As the MWN is not a supervised company network, but a peripheral organized university network, port scans are required to get an overview about current activities in the MWN and the difference between actual and desired state of the network. Due to long scan times with the port scanner Nmap, used thus far for scanning the MWN, we looked for other solutions and found new port scanners like Masscan and ZMap. In this thesis, we compared and evaluated them and concluded, that Masscan is currently best suited for scanning the entire MWN.
Additionally, since October 2014, a new SSL/TLS security breach named Padding Oracle on Downgraded Legacy Encryption (POODLE) is known and the most secure way to prevent it, is to disable SSL 3.0 and older versions. This is the second task of this thesis: providing a fast solution regarding SSL 3.0 fallback detection. To fulfill this goal, we developed a new scanning evaluation tool, the Fast Port Scanning Tool, to scan the entire MWN. In the end, this tool allowed us to scan about 100 times faster than before. Furthermore interesting information, regarding the scanned hosts, can be saved in a file, containing IPs, open ports, services, operating systems (OSes) and vulnerabilities, written to single columns and sorted by IPs. We scaned 500,000 hosts in the MWN and more than 2000 hosts were detected with SSL 3.0 enabled. Furthermore we detected a part of the MWN to be unable to withstand a packet rate of more than 200,000 packets per second and identified three hosts, with more than 16,000 ports open. These results enable the responsible administrators of the MWN to improve its security.

Zusammenfassung

Das Leibniz Rechenzentrum (LRZ) ist Anbieter des größten Forschungsnetzwerks in München, dem Münchner Wissenschaftsnetz (MWN). Da es sich nicht um ein streng organisiertes Firmennetzwerk, sondern um ein dezentral organisiertes universitäres Netzwerk handelt, sind Informationen über aktuelle Aktivitäten und Unterschiede zwischen Soll- und Ist-Zustand schwer für die Verantwortlichen am LRZ zu erhalten. Dafür werden Portscans benötigt, welche bisher mit dem Portscanner Nmap durchgeführt wurden. Diese Scans dauern jedoch aufgrund der Größe des MWN mit Nmap bis zu einem halben Jahr, weshalb diese Arbeit eine Lösung zum schnelleren und effizienteren Portscannen sucht. Dafür wurden Scanner wie Masscan und ZMap mit Nmap verglichen und der am besten geeignete ermittelt: Masscan.
Außerdem ist seit Oktober 2014 die SSL/TLS Sicherheitslücke POODLE bekannt, welche vollständig nur durch das Deaktivieren der SSL 3.0 Verbindung zu vermeiden ist. Um die Hosts zu ermitteln, welche innerhalb des MWN für POODLE anfällig sind und um schneller als bisher scannen zu können, wurde das Fast Port Scanning Tool im Rahmen dieser Arbeit entwickelt. Es benutzt die Scanner Masscan und Nmap um Informationen über IPs, Ports, Services und Betriebssysteme zu ermitteln und abzuspeichern. Nach den Scans des MWN, konnten von 500.000 gescannten Hosts, mehr als 2000 Hosts als anfällig für POODLE ermittelt werden. Zusätzlich wurde ein Teil des MWN anfällig für hohe Paketraten mit mehr als 200.000 Paketen pro Sekunde ermittelt und drei Hosts mit über 16.000 offenen Ports gefunden. Auf Basis dieser Ergebnisse können die verantwortlichen Administratoren nun die Sicherheit des MWN erhöhen.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since the development of the well known port scanner Nmap, in the 1990s [4], many different scanners were invented. Among them are Superscan [5] or newer models like ZMap [6] and Masscan [7]. They all aimed to be even faster, more accurate and reliable, though Nmap remained one of the most commonly known and used scanners, due to its large community [4, 8, 9]. Even large Internet providers, such as the Leibniz Supercomputing Centre (LRZ) in Munich [10] are interested in fast and continuous port scans [10] to get an overview of activities in their network, the Munich Scientific Network (MWN), being an example of a large research network. Furthermore scanning for possible vulnerabilities and thus securing the MWN is also in the best interest of a peripheral organized university network. Being a research network the MWN has its advantages in providing open access, a large bandwidth, if required, for research purposes and not controlling their user's activities like a company network. On the downside, this makes it more vulnerable because the provider LRZ has no influence and widely no knowledge of activities ongoing in the MWN, but is responsible for the MWN. For gaining a quick overview about activities and the difference between actual and desired state, port scans are required. The port scanner Nmap was used for this task at first, providing a good summary about services and activities in the MWN, but due to the restricted speed of Nmap, these scans can take up to six months to complete [10]. As for the continuous growth of the MWN a new solution is required to complete entire scans faster than currently.

1.1 Goals of the thesis

In this thesis we address both issues- how to scan faster with the same accuracy as before and how to scan for vulnerabilities in the MWN, especially for the recent Secure Socket Layer (SSL) 3.0 fallback vulnerability Padding Oracle On Downgraded Legacy Encryption (POODLE) [11]. To find a good solution to these problems, the thesis has three common goals:

First of all, we evaluate suitable port scanners like ZMap and Masscan for their use regarding speed, plausibility and reliability in the MWN and concerning detecting POODLE. In the second step, making use of the respective scanners, a port scanning evaluation tool is developed in Python to enhance scanning speed significantly. We used this tool in the third step, scanning the entire MWN, finding hosts, open ports, services, additional banner information, operating system and vulnerable hosts to POODLE. This information is saved and statistically evaluated.
As network security is one of the main fields of research in the "Chair 8: Network Architectures and Services" [12] of the Technical University of Munich (TUM), located next to the LRZ, this chair is also interested in that topic and thus supervising this thesis. It is therefore a cooperation work of the LRZ and the department of informatics of the TUM, both interested in the development and use of the tool, the results and evaluation of the MWN scans based on the theoretical background about port scanners and POODLE.

1.2 Outline

In chapter 2 we introduce useful related work for this thesis regarding port scanning and Secure Socket Layer/Transport Layer Security (SSL/TLS) vulnerabilities.
Chapter 3 introduces the MWN and the port scanners Nmap, ZMap and Masscan in detail, gives an overview about the SSL/TLS encryption and closes with an explanation of the POODLE vulnerability.
In Chapter 4 the hands-on part of the thesis starts with gathering information about the chosen port scanners and setting the parameters for scanner comparison tests. Then we describe the approach of the tests and gather and evaluate the test results.
After the best suited scanner is evaluated, the development of the scanning tool, the "Fast Port Scanning Tool", is discussed in chapter 5. Starting with the planing phase, we continue with the implementation phase and end with hints how to work with the tool.
Chapter 6 describes the preparation phase and the approach of the scans of the MWN with the scanning tool, as well as the results and the evaluation of them, examining information about amounts of hosts up, ports open or services used and hosts vulnerable to POODLE.
With the conclusions in chapter 7 the thesis closes by summarizing the most important results and providing an outlook on future work.

Chapter 2

Related Work

In this chapter we provide an overview about related work regarding port scanners, port scanning techniques and SSL/Transport Layer Security (TLS) vulnerabilities.

2.1 Related work regarding port scanning

As a direct predecessor work of this thesis, the interdisciplinary project of Omar Tarabai "A Penetration Testing Framework for the Munich Scientific Network" [13] must be mentioned. Though differing in the goals, the work aimed to provide a framework for efficient port scanning with Nmap, detecting vulnerabilities with the Nmap Scripting Engine (NSE) and perform service and version detection on the MWN [13]. In contrast to [13], this thesis aims more to find one specific SSL/TLS vulnerability and provides a faster port scanning solution and evaluation than Nmap.
Another framework for fast port scanning the MWN is Dr. Portscan, developed by Felix von Eye, Wolgang Hommel and Stefan Metzger [14]. This tool was developed to get feedback concerning the difference between actual and desired state of a decentralized university network. However the emphasis of Dr. Portscan was more focused on automated processing of data than speed in showing the difference between the actual and desired state, given that Dr. Portscan uses Nmap as port scanner. Nevertheless this thesis builds upon information and lessons learned from these two earlier papers.
In the following we give a general overview of scientific papers regarding port scanning and SSL/TLS vulnerabilities. A well known port scanner is Nmap [4] because of its age, popularity and many functions. But with the development of Zmap [6] and Masscan [7], two possibilities for faster port scanning were developed. We describe the differences and functionalities of these three scanner in detail in chapter 3. Papers about Zmap like "ZMap: Fast Internet-Wide Scanning and its Security Applications" [2], "Zmap: The Internet Scanner" [6] and "Zippier ZMap: Internet-Wide Scanning at 10 Gbps" [15] and papers about Masscan such as "Finite State Machine Parsing for Internet Protocols: Faster Than You Think" [7] or "MASSCAN: Mass IP port scanner" [16] are very important for the thesis.

General work about port scanning are “A review of port scanning techniques” [17], “Framework for creating realistic port scanning benchmarks” [18] or “Investigating Study on Network Scanning Techniques” [19] all related to Special Interest Group on data COMMunications (ACM) (SIGCOMM). They especially helped designing the test environment for comparing the port scanner.
Lastly the importance of secure ports and regular updates concerning the services and tasks, using certain ports in a network is summarized in “Network attacks: Taxonomy, tools and systems” [20]. Here the authors emphasize the importance of broad background knowledge regarding publicly available tools and systems for preventing and defending against attacks [20]. Furthermore attacks are categorized and tools are introduced to help both, network attackers and defenders. Thus we used some content from their paper in this thesis for accessing background knowledge on the diversity of tools available to the public.

2.2 Related work regarding SSL/TLS vulnerabilities

A detailed overview about SSL/TLS is given in detail in chapter 3.3. Nevertheless the most mentionable papers regarding attacks on SSL/TLS should be introduced here.
The work of Gregory Bard in the year 2006 “A challenging but feasible block wise-adaptive chosen-plaintext attack on SSL” [21] provides the basis for many later following attacks on SSL/TLS such as Browser Exploit Against Secure Transfer (BEAST) [22], Compression Ratio Info-leak Made Easy (CRIME) [23] and Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext (BREACH) [24]. BEAST uses a weakness in the Chaining Block Cipher (CBC) mode of operation in SSL 3.0 and TLS 1.0. Every de- and encryption starts with an Initialization Vector (IV) and only the first IV of the first plaintext is chosen randomly, while the following IVs can be calculated from the last blocks of the previously encrypted plaintext. As an attacker can verify his guesses with a guessing oracle, it is possible to decrypt e.g. SSL/TLS secured cookies.
CRIME was made possible through BEAST [23] and uses a compression “side-channel” approach. Through Huffman-Coding and the “LZ77” algorithm, recurring strings are filtered and replaced with a reference to the last occurrence of that string. So the length of the request diminishes, which can be detected by an attacker, verifying his guesses about the content in the SSL/TLS encrypted message.
BREACH is based on CRIME, also using compression for a “side-channel” attack. The difference is, that each letter is weighted according to the length of the reply message. After enough tries, the letters and their position in the SSL/TLS encrypted message are known with a certain percentage, depending on the length of the message.
After fixing most of the vulnerabilities these recent attacks were using, the HEARTBLEED bug was made public [25] and the library OpenSSL was gravely affected by it. This weakness allows to steal the secret keys used for X.509 certificates, which makes stealing user names, passwords, messages and emails possible, without leaving a

trace [26].
The attack we deal in this thesis with, is POODLE [11] and thus many research papers such as “POODLE attacks on SSLv3” [27], “This POODLE Bites: Exploiting the SSL 3.0 Fallback” [11] and “The POODLE bites again” [28] are important for providing the necessary background information about the requirements for an attack on a POODLE affected system. The POODLE vulnerability and related attacks are described in detail in chapter 3.3.2. Furthermore as it is the latest yet known vulnerability, fixing it is a very important issue for any network, including the large research network MWN.

Chapter 3

Background

Beginning with an introduction of the respective research network MWN, we introduce the port scanners Nmap, ZMap and Masscan and provide reasons for the choice of these three scanner. We further introduce the SSL/TLS encryption and POODLE, providing background knowledge on the subjects discussed in the thesis.

3.1 An example of a large research network: The Munich Scientific Network

The MWN follows the principle to provide easy and unrestricted access for all employees and students of institutions connected to the network [29]. Its provider is the LRZ of the Bavarian Academy of Science (BAdW), which is a common data center for the BAdW, the Ludwig Maximilian University of Munich (LMU) and the TUM. Of course, far more institutions are using the MWN, all listed in a brochure about the MWN (see [29]).
Furthermore the entire network consists of 1,640,832 IPs (Internet Protocol (IP)) (dated 10th January 2015) divided into 1,179,648 private IPs for, e.g., student hostels or private households and 461,184 IPs used by institutions, e.g., TUM, LMU or BAdW. With more than one and a half million IPs, the MWN can be regarded as a large research network [10, 29]. In the following, we provide an overview about holder-of-rights-of-use, the services provided and, finally, advantages and drawbacks.
First, every member of connected universities as well as all employees of connected institutions can use the MWN. Even though, e.g., the faculties of medicine (LMU, TUM) and informatics (TUM) use their own network architecture because of special research and teaching character of medicine and informatics networks, the connection to the Internet (X-WiN) is used all together [29]. Via “LRZ-User maintenance” new user can be registered for using the MWN and additionally more than 65.000 installed workstation computers are ready for use. Concerning data volume on a monthly basis in 2012, more than 650 Terabyte (TByte) were received and 370 TByte were sent via the X-WiN interface. The scientific network X-WiN connects universities and research facilities in Germany and with scientific networks in Europe or on other continents. It

is the successor of Gigabit Wissenschaftsnetz (G-WiN), maintained by the Deutsches Forschungsnetzwerk (DFN) and currently one of the most performance-capable scientific network worldwide [30, 31]. Significantly, not only members of institutions or universities can gain access. Mobile terminal devices can also be registered via a Wifi connection. These devices pose a significant risk to the network in general, as it is exposed to unknown new software, brought through the mobile devices. This thread will be discussed below.
To get a better overview, where which services and bandwidth is provided, figure 3.1 unfortunately only available in a German version, sums this up [29].

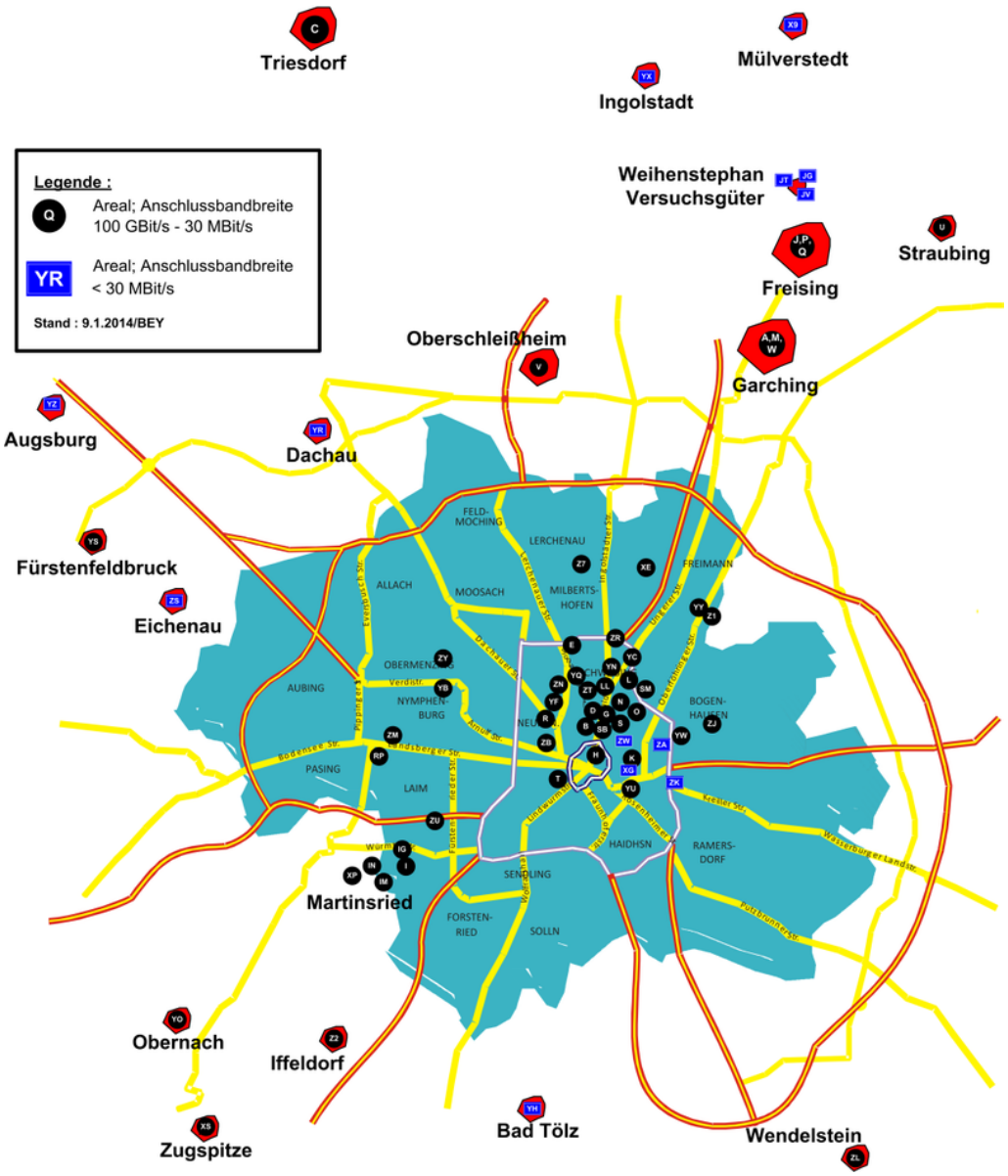

Figure 3.1: Overview of services and speed, provided by the MWN [1]

All services such as World Wide Web (WWW), E-Mail, Virtual Private Network (VPN), maintaining an outer firewall and access to online-media (e.g. universities libraries) are provided based on servers and services such as Domain Name System (DNS), Dynamic Host Configuration Protocol (DHCP) or Network Time Protocol (NTP) to every beneficiary. Also network wide Internet Protocol version 6 (IPv6) support has been established by the end of 2013 and the LRZ operates an own Autonomous System (AS). Concerning bandwidth, the MWN offers 1 to 100 Gigabit Ethernet connections, also Fast Ethernet with 100 Megabits per second (Mbps) and xDSL with 2 to 50 Mbps.
As already noted in the introduction (chapter 1), the MWN suffers from not having a well organized and maintained company network because each institution is responsible for maintaining its own part of the network. Consequently, the provider LRZ is not well informed about activities happening in the network. Lacking a central department responsible for security, the network is vulnerable to new attacks and new found vulnerabilities in network protocols.
Combined with the system Secomat for monitoring the private IPs and an Intrusion Detection System (IDS) "Suricata" a central evaluation of security messages based on the Open Source tool QRADAR is made possible, which helps maintaining a minimum level of security [29].
In some cases these measurements are not enough and port scanning must be performed, to get an actual summary about activities and the difference between desired and actual state.

3.2 Overview of port scanners

In this section we first give an overview about available port scanners. Afterwards the chosen port scanners Nmap, ZMap and Masscan for the Fast Port Scanning Tool are described in detail.
Due to its age and tested reliability by various researchers [4, 8, 9] and as Nmap was already used within the MWN with good results [14], Nmap serves as comparator for the accuracy of ZMap and Masscan. Those two scanner were chosen as their developers claim that their scanner can scan the entire Internet (Internet Protocol version 4 (IPv4) address space) in 45 (ZMap) or six minutes (Masscan) [2, 7] with the same accuracy as Nmap.

3.2.1 Nmap

Nmap was first released in 1997 by Gordon "Fyodor" Lyon and has become a well known network security scanner worldwide [4]. As this is one of the oldest and most used port scanners, it supports a variety of features including port scanning, service and version detection, Operating System (OS) detection and firewall or IDS evasion and spoofing, just to name a few of them.
Due to its age it is probably the slowest of the three mentioned port scanners but

according to many papers [2, 4, 7], also one of the most reliable.
Apart from all the possibilities already provided by Nmap, more feature can be added by writing customized scripts in the programming language Lua [4].
While Nmap supports a variety of different scan methods, such as the Transmission Control Protocol (TCP) connect scan, the User Datagram Protocol (UDP) scan, TCP NULL, FIN or Xmas scans [4], the most common used is the TCP SYN method:
TCP SYN scan[1] is probably the fastest scan technique. It works by sending a SYN packet to the host, acting as if a real connection should be established and waiting for a response. If a response is received (ACK), the port is very likely listening and therefore open, while a RST indicates a non listening (closed) port. In case, that after several retransmission no response is received, the port is listed as filtered [4].
All open or open|filtered ports are gathered via a regular port scan and passed to Nmap's service scanning module, which performs service/version detection in parallel. In case of an open TCP port, Nmap tries to connect to it and once the connection is established, listens. If a common service such as Secure Shell (SSH), Hypertext Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP) or Post Office Protocol 3 (POP3) is running on that port, it usually identifies itself by sending an initial welcome banner, which contains information about service and version number [4]. After comparing this information to the Nmap probe signature database with more than 3000 entries, the service can be fully identified, Nmap is done and prints the service name either to console or to a file. Otherwise Nmap continues to use other techniques as the "Nmap RPC grinder" and repeats sending other purpose-built packets to the host [4]. Regarding OS detection[2], Nmap sends up to 16 TCP or Internet Control Message Protocol (ICMP) probes to open or closed ports of the target host [4]. As these probes contain data, which exploits "various ambiguities in the standard protocol RFCs" [4], Nmap can generate a fingerprint of the OS by analyzing certain attributes and flags in the responses. With these values, Nmap can guess an OS with a certain percentage for correctness.

3.2.2 ZMap

ZMap, developed in 2013 by Zakir Durumeric, Eric Wustrow and J. Alex Halderman at the University of Michigan [6], uses a TCP SYN scan on specified ports. The main paradigm of the scanner is to be as fast as possible, which is achieved by using an asynchronous approach. Asynchronous means that its transmit and receive threads are independent from each other. Furthermore "skipping the TCP/IP stack and generating Ethernet frames directly" [2] makes sending packets as quickly as the source's Network Interface Controller (NIC) allows. In the standard configuration, ZMap only scans for one specified port in the given IP range, which requires to write wrappers or similar programs to scan for the entire port range spanning from 0 to 65,535.
As the tool is relying heavily on a high packet transmission rate, the developers recom-

[1] Nmap parameter: `-sS`
[2] Nmap OS detection command: `-O`

mend strongly to follow their “scanning best practices” guideline, consisting of seven points citedurumeric13. Furthermore they encourage to use a blacklist file (exclude file) to exclude every reserved, unallocated or otherwise undesired IP space.
Apart from supporting a TCP SYN scan, it also supports the ICMP echo request scan and also the UDP datagram scan. Figure 3.2 provides an overview about the ZMap architecture.

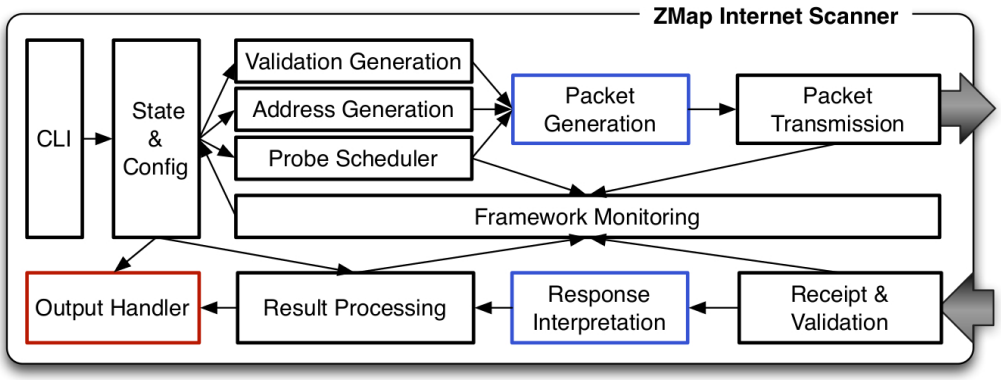

Figure 3.2: Architecture of ZMap [2]

Via the Command Line Interface (CLI) the ZMap commands can be entered by the user and parsed in the “State & Config” modules. The three methods “Validation Generation”, “Address Generation” are responsible for calculating the addresses and in “Probe Scheduler” the timing of probes is set. A probe is a special probe packet, filled with ,e.g., a TCP SYN request. “Packet Generation” builds these probe packets, supporting, e.g. TCP SYN and ICMP echo scans, and “Packet Transmission” sends these probes to the generated addresses of the scanned hosts. While the “Framework Monitoring” monitors every in- and outgoing packet, the “Receipt & Validation” part accepts responsive packets to the probes, the “Response Interpretation” interprets the answers, also supporting multiple kinds of probes [2]. Finally after “Result Processing” the processed information is either written to console via stdout or to a file in a application-specific way, making use of the “Output Handler” of ZMap [2].
The speed of ZMap is an issue regarding the addressing of probes, as sending them in numerical order would probably overload and block a network. So ZMap uses a random permutation of the address space, iterating over a multiplicative group of integers modulo p, with p being slightly larger than 2^{32} [2]. If p is prime, the group is cyclic and therefore reaches all addresses in IPv4 address space once in a cycle. After choosing the multiplicative group, selecting a new random permutation for each scan is the next problem. The authors solved this by choosing a new primitive root of the multiplicative group and a new random starting address [2]. Utilizing an isomorphism of the group, finding new random primitive roots is possible without too many extra operations. Once this primitive root is found, iterating “through the address space by

applying the group operation to the current address" [2] allows to end a scan once the initially scanned IP address is reached. Excluding IP addresses from an internet wide scan, resulting in scans, with only a very small part of all IPv4 addresses, is realized by using radix trees, often used by routing tables [32, 33]. These excluded ranges can be specified in a configuration/exclude file.
Sending packets is performed as fast as the source's CPU or network card allows or as fast as the user defines it to be. If an answer packet is captured, the source and destination port is checked and the packet is either discarded, if it was not intended for this scan or passed to the active probe module for further breakdown [2].
ZMap can also complete a TCP handshake and get additional information about the service running on a certain port. This is realized via "forge_socket", which bypasses the Linux kernel and allows communication over the ZMap-initiated TCP sessions via application-level handshakes [2].
Following Nmaps example, ZMap is also adaptable by the user, who can write his own probe and output modules and include them to ZMap, resembling the Lua scripting language support from Nmap.
Finally ZMap is a faster scanner than Nmap [2, 6, 15], though more difficult to use when scanning port ranges as only one port at a time can be specified for scanning. So for a fast, complete overview of all 65536 ports, ZMap needs to be executed, e.g., in parallel.

3.2.3 Masscan

Masscan was first presented on the 2014 IEEE Security and Privacy Workshop conference [7]. It provides a high scanning speed, the use of arbitrary IP and port ranges and is based on its own TCP/IP protocol stack.
Running with a maximum rate of seven million packets per second [15] is only achievable, when using a special Ethernet adapter and driver "PF_RING ZC (Zero Copy)" [34]. Though differing in speed Masscan uses similar commands as Nmap, also using the command line as default output.
Regarding its architecture, Masscan works more like ZMap [6], "scanrand" [35] or "unicornscan" [36] using asynchronous transmission [7]. Asynchronous, already explained in section 3.2.3, means that its transmit and receive threads are independent from each other.
Randomization of targets is necessary with the same reasons as for ZMap: not to overload and block the scanned network. This works by iterating through the range of IPs times ports ($IP\_Ports = IPs \cdot Ports$), encrypting the current value of i, the iteration variable, and thus creating a random 1-to-1 mapping between the "original index variable and the output" [7]. The next scanned host and port is then grabbed via the *pick* function, taking the address and port at the index of the encrypted index variable. After the receive threads receives a packet, its information is processed and passed of the output module.
Masscan is also designed for the "C10M problem", bypassing the kernel's custom net-

work driver, user-mode TCP stack and user-mode synchronization [7]. The "C10M problem" describes the concurrency problem of ten million operations per seconds, running independently from each other. Together with PF_RING, this allows for a maximum transmission rate limited only by the speed of the used network card and probably the CPU.
Apart from port scanning, Masscan can also grab banners, by completing the three-way handshake and "grab simple banner information" [7] from the application running on the port. This works basically the same way, as the first steps of service detection of Nmap, but again Nmap can perform more techniques, while Masscan is limited to a simple banner exchange.
In addition to its features, Masscan can perform scans designed for a specific purpose via appending scripts [3]. Especially the "poodle" script is important, as it allows to perform a port scan and a SSLv3 fallback check at the same time.

According to its developer Robert David Graham, Masscan is currently "the fastest Internet port scanner" [7] and should produce similar results to Nmap. Although this statement might have been true for some time, the development team of ZMap published a newer paper, stating: "Masscan [had its] peak [at] 7.4 Mpps using a single-adapter configuration ... [and] 50% of 10 GigE linespeed. ... On the same hardware, ZMap [had its] peak [at] 14.1 Mpps." [15]

3.2.4 Further scanning tools

Although tools such as "suscan" [5], "scanrand" [35] or "unicornscan" [36] also exist, they are either too old and with that too slowly for the task addressed here, lack functions such as service/OS detection or are not well documented. We also found another new scanner, "nscan" [37], but as it was released around the 30th of January 2015, it was released when this thesis was well underway and evaluating nscan was not possible anymore.

3.2.5 Comparison of Nmap, ZMap and Masscan

Regarding ZMap and Masscan, the major difference is that ZMap was especially designed to scan the entire IPv4 address space on one port, while Masscan is more adaptable, allowing arbitrary IP and port ranges.
When comparing Nmap to ZMap and Masscan there are some major differences: First of all Nmap uses a synchronous approach, meaning that transmit and receive thread are not independent, while the other two scanner are asynchronous. This means, that due to the architecture of ZMap and Masscan sending, receiving and creating packets runs independently [2,7]. Nmap on the other hand, while running synchronously, must

[3] `--script`

first create packets, each assigned to a certain host, then send and receive the responses. Another difference to Masscan is the possibility of host discovery to determine, if a host is up or not. If this option is not disabled [4], only hosts, marked as “up”, get scanned afterwards, which is usually found out by an ICMP echo request and reply.
Although Nmap seems like a reliable fast port scanner, it lacks the required speed for scanning a large research network with about 500,000 hosts.
A combination of Masscan and Nmap seems the most promising solution for scanning a large network for IPs, open ports, services and OSes.

3.3 SSL/TLS vulnerability POODLE

In this section we give a short overview of the SSL/TLS encryption, possible attack patterns and the current vulnerability POODLE.

3.3.1 Introduction to the SSL/TLS-encryption

SSL version 1.0 has been developed at the beginning of 1990s by the company Netscape with the goal to create a secure connection between applications, typically between web server and Browser. Connections established via SSL/TLS were built to provide mutual authentication of participants, encryption and integrity [38]. As fundamental design flaws were corrected, to withstand newer attacks [39], in 1999 SSL was renamed to TLS. In 2013, the most spread implementation of TLS was 1.0 [38] although versions 1.1 and 1.2 were long available, providing even more security.
In the depiction 3.3 connection establishment is summarized.
Concerning its functionality, the establishment of a TLS connection begins with a handshake phase: During that phase, all cryptographically required data is exchanged between client and server. First the client connects to the server, which authenticates itself, usually with a X.509 certificate towards the client. Optionally the client can authenticate itself too, towards the server also normally using that kind of certificate. Afterwards the client performs a key exchange with the server. Also the client sends the desired Cipher suite and a mutual secret is calculated either from a Diffie-Hellman key exchange or from the exchange of the encrypted random number. To provide encryption, a symmetric cipher is used to encrypt every packet with the cryptographic key, derived from the mutual secret. For authentication and integrity a Message Authentication Code (MAC) is used. These two measurements are combined in the following order: MAC then encrypt. So first a message is authenticated and then encrypted. With the calculation of the session key from the mutual secret, the handshake phase is over and information exchange begins encrypted and authenticated [38].
Following the handshake-phase, the record layer is responsible for the exchange of application data over a secure channel.

[4] `-Pn`

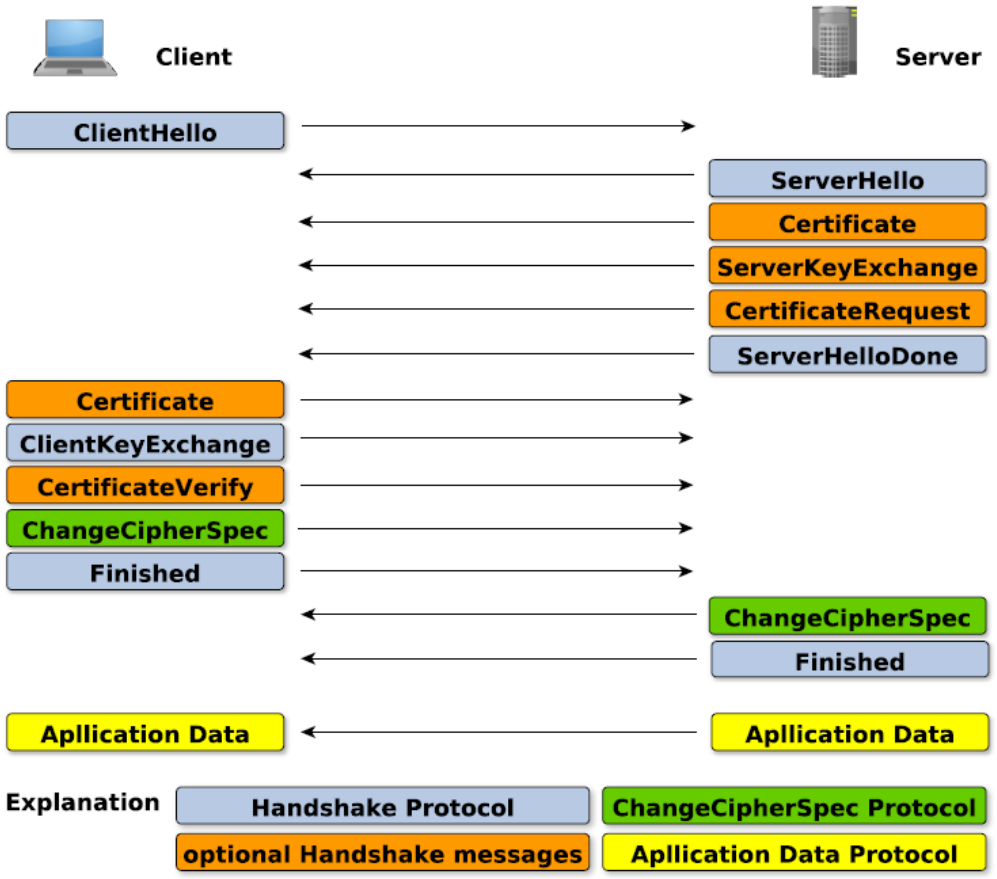

Figure 3.3: SSL/TLS connection establishment [3]

3.3.2 POODLE

More recently, there were many possibilities to break the TLS encryption, such as CRIME [23], BREACH [24] and BEAST [38] and finally POODLE [11].
Every Browser and server, still supporting SSL 3.0 is vulnerable to POODLE and since October 2014 it is known, that even TLS versions can be affected. This is due to the direct takeover of the padding mechanisms in SSL 3.0 to TLS 1.0 in some implementations, e.g., when using load-balancer of companies such as A10 or F5 [28].
Regarding the use of load-balancer in the MWN, there are almost none [10]. With this information, we decided that scanning for SSL 3.0 fallback in the MWN suffices entirely to eliminate the POODLE vulnerability within the MWN.
In September 2014, Bodo Möller, Thai Duong and Krzysztof Kotowicz, all employed as cryptographers by Google, published a security advisory describing the POODLE thread [11].
It was also mentioned by Adam Langley, also cryptographer and employee of Google,

in his blog “ImperialViolet” [27]. In the following an attack is described, making use of the SSL 3.0 fallback:
To make connection establishment with legacy server easier for clients, a downgrade of SSL/TLS versions is usually implemented on client side [11]. If the client has only SSL 3.0 at his disposal or is forced, e.g., by an attacker to pretend that SSL 3.0 is the latest available version, the client establishes a connection with the server via SSL 3.0. An attacker can do that, if he controls the network between the client and the server and interferes with any attempted handshake offering TLS 1.0 or later, with the result, that such clients will readily confine themselves to SSL 3.0 [27]. Figure 3.4 shows this downgrade-dance, forced by a Man in the Middle (MitM). If an attacker knows which

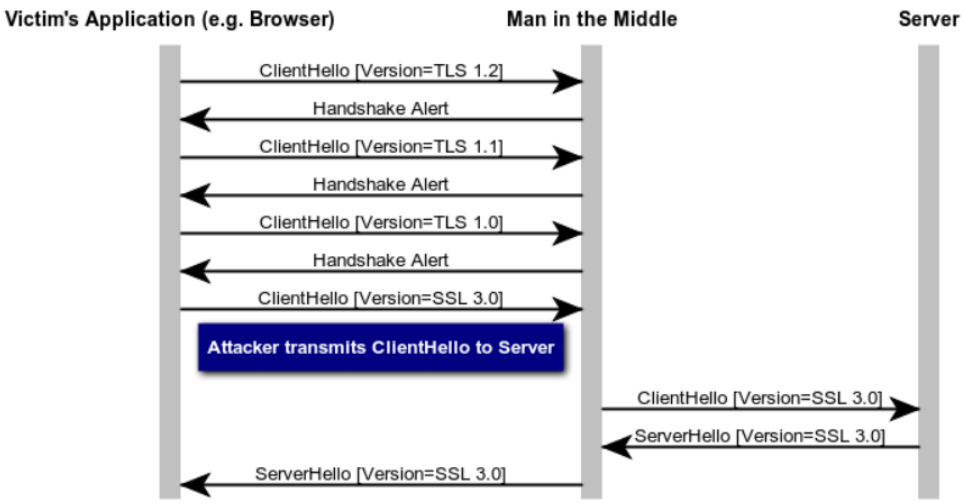

Figure 3.4: SSL 3.0 downgrading, forced by a MitM attacker

load-balancer the client uses, even TLS 1.0 downgrading might suffice to succeed with this attack.
Now a client server connection is established using SSL 3.0. If SSL 3.0 uses the RC4 stream cipher for encryption, receiving information by an attacker, e.g., an encrypted cookie for login information of the victim, is relatively easy as it is known, that “if the same secret (such as a password or HTTP cookie) is sent over many connections and [thus] encrypted with many RC4 streams, more and more information about it will leak” [11]. Receiving a secret from the victim, if the CBC block cipher is used by SSL 3.0, is a bit trickier:
The groundwork of the following attack pattern is based on the work of Gregory Bard [21]. In this work, he describes a structural weakness of CBC in SSL 3.0 and TLS 1.0. CBC begins every encryption and decryption with an IV. The problem with TLS 1.0 and older versions is, that only the IV of the first plaintext is chosen randomly and all further IVs are calculated from the last block of the previously encrypted plaintext. Furthermore an attacker can verify his guesses about the first IV.

The reason behind this is, that the block cipher padding of CBC encryption in SSL 3.0 is not deterministic and not covered by the MAC [11] as SSL/TLS calculate the MAC first and then encrypts the message. Being not deterministic, when decrypting, it can not be verified, e.g. by the victim, if the padding has not been changed, so if integrity is compromised.
Even though this attack also works in other ways on other protocols, one of the most common attack is against Hypertext Transfer Protocol Secure (HTTPS) secured cookies, which is the reason why this scenario was picked for detailed description:

With S being the block size in byte, padding *1* to S bytes is used to get a certain " integral number of blocks before performing block wise CBC [...] encryption" [11]. To exploit this weakness, it is easiest if there is an entire block of padding of *S-1* arbitrary bytes, followed by a value *S*. With the initialization vector C_0 and blocks C_i, processing incoming cipher text record $C_1...C_n$ on recipient's side works by determining $P_1...P_n$ with $P_i = D_k(D_i) \oplus C_{i-1}$. D_k is the block-cipher decryption per-connection key K [11]. Now padding is checked and removed from the end of the message and the same happens with the MAC. If there is a full block of padding and the last block C_n is replaced by the attacker with any earlier cipher text block C_i from the same encrypted message, this cipher text will be accepted if $D_k(C_i) \oplus C_{n-1}$ has S as its final byte [11]. Otherwise it will we rejected, making a padding oracle attack possible.
In order for this attack to succeed several conditions must be matched:

1. The attacker can read the network traffic of the victim, e.g., by being a Man in the Middle.

2. The attacker can force the victim to send HTTPS request to the server.

3. The attacker has full control of the path to the target of the HTTPS request of the victim.

Fulfilling these prerequisites, launching an attack is covered in the following steps. Figure 3.5. shows an attack on a HTTPS secured cookie of the online payment company "paypal".

The attacker forces the Browser or the application communicating with the Internet to send HTTPS requests to the target server (e.g. https://www.example.com/ample). "ample" is chosen in a way by the attacker, that the first Byte *b* of the unknown cookie header is written to the last position of the plaintext block. With *P* being the plaintext block this looks like this:*P|b*. The Browser of the victim negotiates for the SSL/TLS connection a shared secret key with the server and generates via CBC mode an encrypted request, which is sent to the server.
The attacker intercepts the encrypted request. Now the SSL record is modified with a chance that the server will accept the modified record, which allows the attacker to

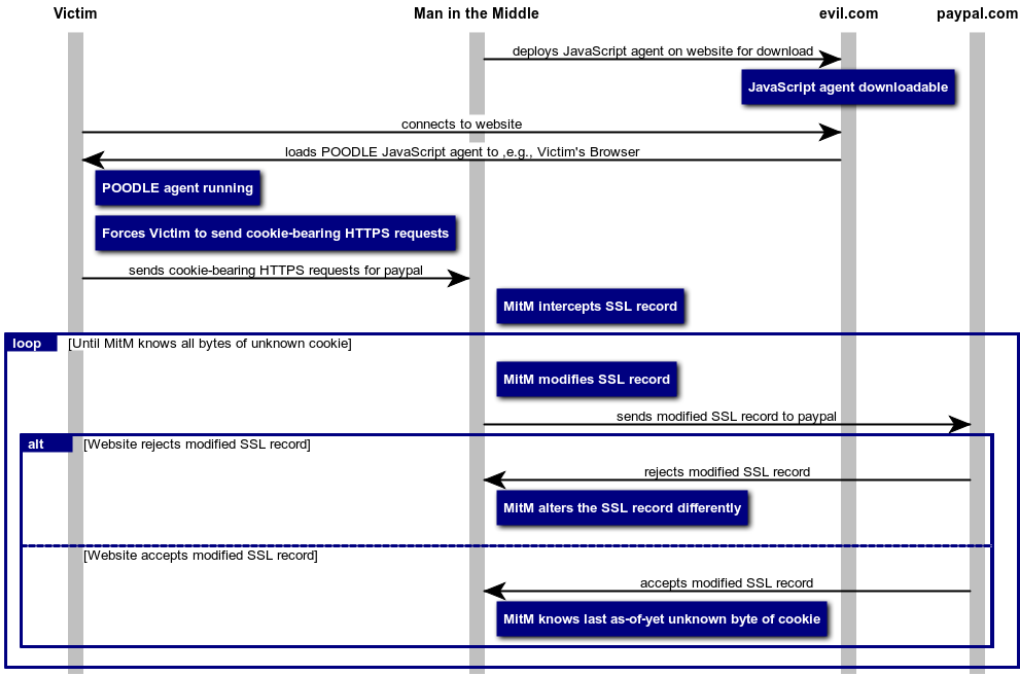

Figure 3.5: Overview about POODLE attack

decrypt one byte of the cookie: b. To calculate b some assumptions must now be made: Each block of C has 16 bytes ($C[0]$,...,$C[15]$), which is used, e.g., for Advanced Encryption Standard (AES). If the size of the cookie is unknown, we can use the padding size to determine the size of the cookie. By inducing byte-growing requests (e.g., by appending a letter in ASCII) allows to observe the point where the block boundary is crossed [11]. When the padding size is revealed, also the size of the cookie is known. Typically the MAC size in SSL 3.0 CBC cipher suites is 20 bytes and without CBC an encrypted HTTPS request looks as follows:
`POST /path Cookie: name= value... body || 20byte MAC || padding` As the attacker controls the request path and body, inducing requests with the following conditions is possible [27]:

1. Padding fills an entire block (encrypted in C_n)
2. The cookie's first not yet known byte is in the final byte in an earlier block (encrypted in C_i).

After the attacker has replaced C_n by C_i, this modified SSL record is send to the server. In an average of 255 of 256 cases, the server will reject the record and the attacker has to try again with a another, new request. If the server accepts the modified record, the attacker can conclude that $D_k(C_i)[15] \oplus C_{n-1}[15]$ =,e.g., 15 and that $P_i[15] = 15 \oplus C_{n-1}[15] \oplus C_{i-1}[15]$ [11]. With this the cookie's next, before unknown byte was calculated by the attacker.

By changing the sizes of request path and body at the same time, so that the request size stays the same, but the position of the headers is shifted, the attacker can find out the next byte [11]. This procedure is repeated until all bytes are known. The expected overall effort is 256 SSL 3.0 requests per byte.
As we already mentioned above, POODLE poses a serious thread and the easiest way to avoid it, is to disable the SSL 3.0 protocol on both, server and client, side. The problem is to identify the servers and hosts still supporting SSL 3.0. So the MWN requires an entire network scan, to identify the servers still supporting SSL 3.0.

Chapter 4

Scanner Evaluation

Finding the best suited scanner for implementing a solution for fast and reliable port scanning is the main goal of this chapter. To evaluate the accuracy of Masscan and ZMap compared to Nmap, we planned and conducted test scans, described in chapter 4.1 and 4.2. The most important scanning results were gathered in chapter 4.3. In chapter 4.4 we give a summary of arguments in favor or against ZMap or Masscan and conclude in chapter 4.5 that Masscan is the most suitable scanner for the further approach.

4.1 Setting the test environment

After considering potential test environments, it was agreed to define the ensuing parameters, described in detail in the further subsections:

1. Testing network
2. Location
3. Rate limiting
4. Acceptance by the administrators
5. Output
6. Speed evaluation
7. Plausibility
8. Files as parameter input source

4.1.1 Testing network

As testing network, we chose the real network of “Chair 8: Network architectures and services” as many different services run on the hosts, a usual workday takes place, a firewall is installed and all hosts are differently configured , with different OSes and

executing different tasks. As to its size and exact IP range, the network exists of 2048 possible hosts with the IP ranges 188.95.232.0/22, 192.48.107.0/24, 131.159.14.0/23 and 131.159.20.0/24.In essence, as this is a real subnet, it simulates the authentic circumstances of the far larger, but also diversely configured, MWN.

4.1.2 Location

First, the question from where to scan must be clarified. For this purpose we set up a Linux based virtual machine, using the infrastructure of the LRZ. The Virtual Machine (VM) is accessible via a SSH remote connection, established from the home residence of the student, and reachable via the address 129.187.251.11, which is the VM "lxdps04.srv.lrz.de". To prevent disconnects to the remote VM, which would result in ending the running programs on the VM, the terminal multiplexer "tmux" [40] was used.
In addition to the VM, we also set up a hardware machine located in the "Chair 8: Network architectures and network services" [12], "bozen.net.in.tum.de", for testing purposes.

4.1.3 Rate Limiting

Another parameter is rate limiting, because user should be disturbed as little as possible by a scan, while working in the MWN. On scanner side, the bandwidth was limited to 20 Mbps for smaller networks, like the network of "Chair 8: Network architectures and network services" with 2048 possible hosts and up to 100 Mbps for larger networks such as the entire MWN.

4.1.4 Acceptance by administrators

While working with this limitation, another issue must be settled: administrator complaints. It is essential during the time of this work, that users or administrators, working in the respective network should be disturbed in his or her normal daily affairs as little as possible. If any complaints occur, the parameters must be adapted or the problem solved in a different way so that acceptance of the scans is guaranteed.

4.1.5 Output

We decided, that every test using one of the scanners, should be documented by an output file in the xml or csv format, so the test results are saved and the test itself stays repeatable. Optional, if necessary, a "pcap" file can be created by the network analyzer "Wireshark" [41]. Furthermore, date, time and the scanning commands were saved separately in another file.

4.1.6 Speed Evaluation

For speed evaluation, some guidelines are mandatory:

1. We scan the same IP address range with the comparable scanners.
2. We scan for the same ports in this IP range with the comparable scanners.
3. We use the same scanning method such as TCP SYN Scan (see chapter 3) for comparison.
4. We use built in randomization of Nmap, ZMap and Masscan for IPs and ports concerning every scanner (see chapter 3.2).

4.1.7 Plausibility

Testing the data for plausibility is very important, as it is not possible to determine if the output of a port scanner is indeed "correct". This shows if a port is really open or closed or if, for example, a firewall blocks the access. For ensuring that the tests are valid, each test scan will be repeated at least two times within a maximum time span of two days. Due to changes in the network itself it is never guaranteed that each test run returns the exact same result. So the comparison scans are simply used to evaluate if the result of the first scan are similar.
The standard approach is to scan with Nmap first and compare the results of ZMap and Masscan to each Nmap scan, as Nmap was already used in the MWN with good results and can thereby regarded as accurate [14].
We regard a scan as plausible, when the results, concerning ports, do not differ more than ten percent from each other in one of the three comparison scans.

4.1.8 Files as parameter input source

Another step to produce comparable test scenarios is to write every command to a file, which is then executed by using the `--config` option using ZMap and the `-c` option using Masscan. Working with Nmap requires to enter all the desired commands in a script file, which is then used as parameter input source. This ensures that apart from being saved in the output file and in a separate text file, the scans stay reproducible.

4.2 Conduction of test scans

After having specified the parameters for scanner comparison, we started comparing the tools' accuracy in test scans. Main testing was performed within a month to get enough different scan results. During that time many different configurations were tried, which are all summed up in the following.
Figure 4.1 shows, where the VM and the hardware machine were located in the different

AS of LRZ and TUM. The external firewall or the IDS should of course be located before entering the TUM [I8] AS.

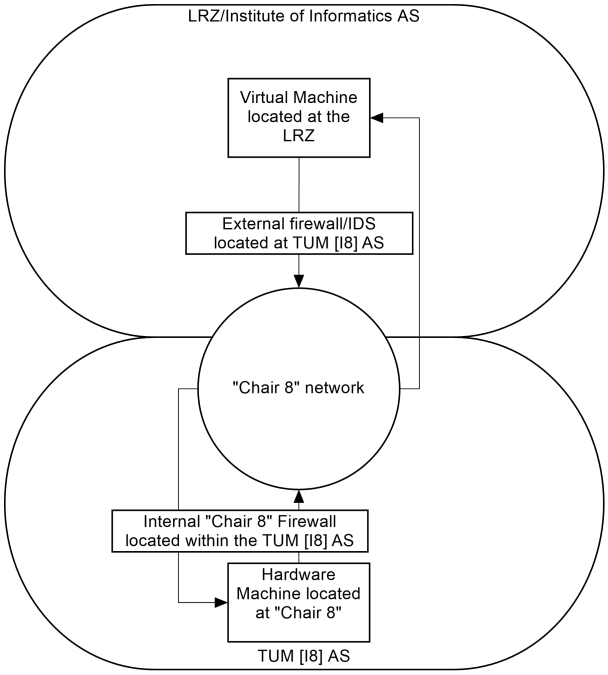

Figure 4.1: Differences between Nmap and Massacan scans, performed from hardware

We started with Nmap, with a full scan of all 2048 hosts in the “Chair 8 ” network. As the scan was interrupted after 20 hours with no output, the need of Linux tools such as “GNU screen” [42] or “tmux” [40], to ensure the stability of the scans, became evident (see subsection 4.1.2).
For this scenario we added an additional “tee” [43] output to collect data until the interrupt.
Furthermore, we detected the different speed limiting possibilities by ZMap, Nmap and Masscan, which led to several purpose-built “pcap”-outputs from “Wireshark” to set an upper limit for the respective packet size of Nmap and Masscan. This was necessary as Nmap and Masscan only support packet rate limitation. We calculated these packet rate limitation to the desired bandwidths of 20 Mbps and 100 Mbps accordingly.
The outputs from “Wireshark” of the captured Nmap and Masscan scans showed both a packet size of outgoing TCP packets of 54 Bytes including headers in 99.95% of all cases and 60 Bytes in the remaining 0.05%, which resulted in the following packets per second rates:

$$\frac{20,000,000}{0.9995 \cdot 54 \cdot 8 + 0.0005 \cdot 60 \cdot 8} = 46,293$$

$$\frac{100,000,000}{0.9995 \cdot 54 \cdot 8 + 0.0005 \cdot 60 \cdot 8} = 231,481$$

So we limited Nmap's and Masscan's packet rates to 46,000 for 20 Mbps and to 230,000 for 100 Mbps to ensure, that we don't cross the bandwidth limit.
With the output of the first test trials, reference scanning started. Every scan was conducted in this and the following subsections (4.3.2, 4.3.3, 4.3.4 and 4.3.5) with the parameters shown in table 4.1. So Nmap scanned all ports, did not perform host detection and DNS resolution, did a TCP SYN scan, had a verbosity level of one, wrote all information to an xml file, used more than 1000 packets per second, was limited to 46,000 packets per second maximum, randomized hosts and had one try per host. For comparison reason we gave Masscan the same scanning commands for scanning behaviour. ZMap was limited to 20 Mbps instead of 46,000 packets per second, wrote the result to a csv file and had a cooldown of 10 seconds per scan.
All parameters are also described in detail in "Appendix B - Scanner parameters". One

Scanner	Parameters
Nmap	-sS, -v, -p-, -oX, -n, -PN, --min-rate, --max-rate, --randomize-hosts, --max-retries, IP range
Masscan	-p, -oX, --max-rate, IP range
ZMap	-B, -p, -o, -c, IP range

Table 4.1: Scan parameters for scanner comparison tests

problem was the speed of Nmap, as it still took Nmap up to 20 hours to finish a scan of all 2048 hosts with all 65536 ports, so due to different time slots, it was difficult to equally compare these test results as ZMap and Masscan were far faster to finish their scans.
For easier evaluation we wrote a comparison program in Python, so that output of Nmap and Masscan only showed hosts, which were up, with their respective open ports. Table 4.2 shows a part of a Masscan scan, evaluated with the self written evaluation tool. As both Nmap and Masscan results could be changed to that form, comparing the

IP	Ports
'131.159.14.36'	'80' , '443'
'131.159.14.111'	'21', '80', '6969'
'131.159.14.197'	'53'
'131.159.14.204'	'80', '443'
'131.159.14.216'	'53'
'131.159.14.221'	'53', '80', '389', '443', '636'
'131.159.14.245'	'443'

Table 4.2: Example output from comparison tool

scans, e.g., via the linux command "diff" [42] returning the differences in the files, was possible.
After the scanner produced several test scans, from hardware and virtual machine, we stopped comparison scanning and gathered the results.

4.3 Results of comparing tests

In this section we gather the results from the scanning phase, beginning by showing an enormous difference concerning host detection when scanning from the hardware or from the VM. Afterwards the results from the different scanners Nmap, ZMap and Masscan are compared.

4.3.1 Particular characteristics of ZMap and Masscan

As mentioned in chapter 3.2.2, we could verify a scanning hindrance with ZMap, as ZMap does not accept arbitrary port ranges but only one port as parameter for each scan. Thus one ZMap call returned only information about one port, while Nmap and Masscan scanned all ports per call.
Regarding Masscan, we detected some duplicate entries in the output files, which needs to be eliminated to allows equal scanner comparison.

4.3.2 Scan results from scanning within the “Chair 8” network from a hardware machine

First, Nmap scans from the hardware were more accurate than the scans of ZMap and Masscan as shown in figure 4.2. This was mostly due to a firewall policy allowing only

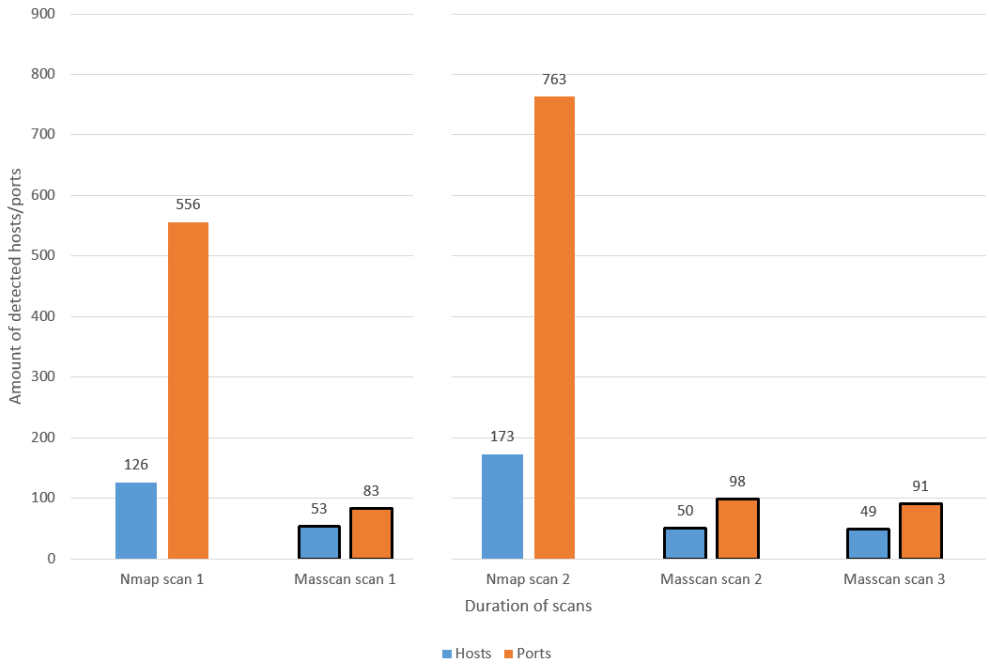

Figure 4.2: Differences between Nmap and Masscan scans, performed from hardware

a certain amount of packets to get through and limiting ICMP) packets to a certain

amount per minute. Furthermore the scans from the hardware machine use another interface than from the VM:
Scanning from the "bozen" server in "Chair 8" is only possible via the interface "eth0.220", which can be specified in Nmap with "-i"[1] and in Masscan via the gateway Media Access Control address (MAC address) [2]. With the high packet transmission of ZMap and Masscan they could not detect as many hosts as Nmap, as Nmap had far more time to wait for a response [12].
However it was not possible, against sincere efforts, to get results from ZMap, scanning from the hardware machine[3], which is the reason, why ZMap is excluded from comparison here.
Even though Masscan detected the same hosts and ports as Nmap has, the difference in numbers is too huge. For more reasons of this significant difference, further tests would be required, which is beyond the scope of this thesis. To summarize this section:
As scans on the Munich Scientific Network will be run from a virtual machine within the LRZ network, these test results have no significance for the further approach and were only done for testing purposes.

4.3.3 Comparing Masscan and Nmap

Following the guidelines listed at the beginning of the chapter it was possible to get good test results comparing Nmap and Masscan, when scanning from the VM. However, the speed difference between Nmap and Masscan made it impossible to scan at the exact same time. Thus the results are not exactly the same, but differ slightly. As figure 4.3 shows only absolute numbers, the following description lists differently detected hosts and ports by Nmap and Masscan.

1st Nmap vs. 1st Masscan scan
Masscan detected one host with one port more.

1st Nmap vs. 2nd Masscan scan
Nmap detected seven other hosts with 18 other ports than Masscan.
Masscan detected one other host with two other port than Nmap.

2nd Nmap vs. 3rd Masscan scan
Masscan detected one other hosts with two other ports than Nmap.

2nd Nmap vs. 4th Masscan scan
Nmap detected five other hosts with nine other ports than Masscan.

We performed two Nmap scans and four Masscan scans to get different values from different points in time. Regarding duration of the scans, Masscan required 40 to 50

[1] -i eth0.220
[2] --gateway-mac 00:1b:ed:e6:7b:00
[3] sudo zmap -B 20M -c 5 -G 00:1b:ed:e6:7b:00 -p 80 -o 19_11_ZMap_80_HW_first.csv 188.95.232.0/22 192.48.107.0/24 131.159.14.0/23 131.159.20.0/24

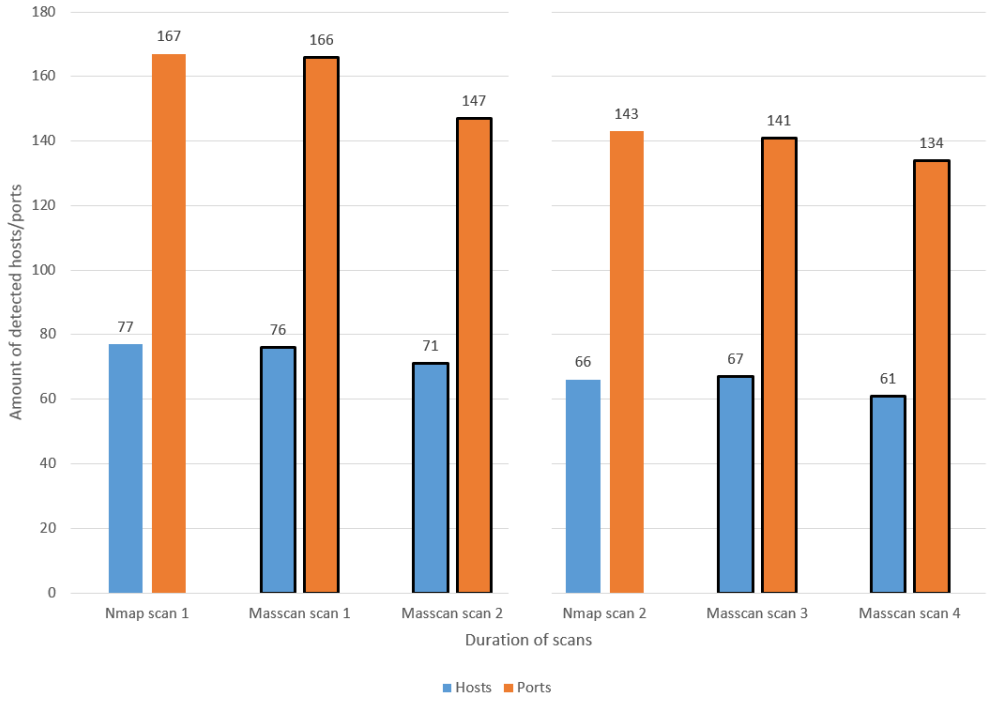

Figure 4.3: Differences between Nmap and Masscan scan results on testing network, performed from VM

minutes for all 2048 hosts on all ports, while Nmap required eight to ten hours on the same range. Furthermore, we performed all tests of this chapter within a day. As these tests were carried out to evaluate the accuracy of Masscan compared to Nmap, the accuracy of Masscan can be calculated with the values shown in figure 4.3.
If every port is counted as one, the port difference between Nmap and Masscan from scans, performed on the VM is 167 vs. 166 and 147 ports in the first run and 143 vs. 141 and 134 ports in the second run. This results in the following port cover percentage:

$$\frac{(166 + 147) : 2}{167} = 93.71\%$$

and

$$\frac{(141 + 134) : 2}{143} = 96.15\%$$

Combining these two values results in $\frac{96.15\%+93.71\%}{2} = 94.93\%$. With these results we concluded that Masscan possesses an accuracy of 94.93% if compared to Nmap.

4.3.4 Comparing ZMap and Nmap

As already mentioned in section 4.2, ZMap can only scan one port per scan run. So a ZMap scan was performed on the first 1024 ports, as they are “named ports” [44], and

on several other ports detected by Nmap and Masscan. In the end we chose 15 different ports for comparison testing, as ZMap discovered on all of them more than one host up. Figure

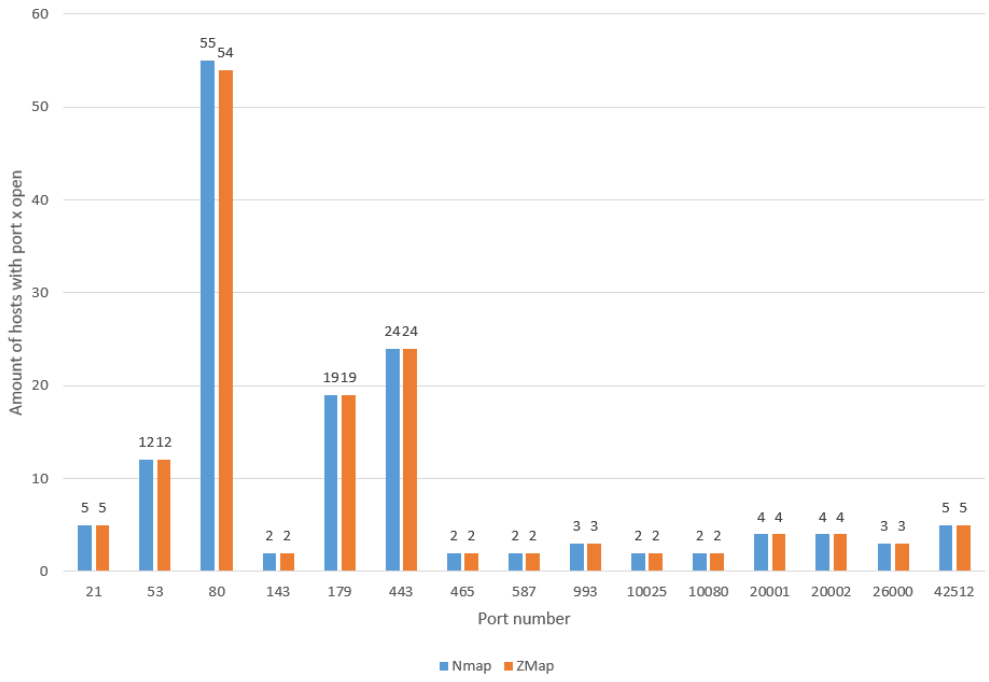

Figure 4.4: Comparing Nmap and ZMap by selected ports, performed from VM on the same day

As above in section 4.3.3, we performed two tests to get different values from other points in time. ZMap required ten seconds per scanned port on all 2048 hosts, because of its internal cooldown, while Masscan stayed at 40 to 50 minutes for all ports on all hosts, as in section 4.3.3. Furthermore, we performed all tests of this chapter within a day. Figure 4.4 and 4.5 show the amount of detected hosts on that port and compare the results of Nmap and ZMap

Regarding different detected hosts, Nmap detected one host more, while ZMap detected three more hosts in the second test compared to the respective other scanner. The rest was identical.

So ZMap and Nmap had a port difference of four, but it must be reminded that not every port was scanned but only a selection of the ten most mentioned ports. Hence Nmap discovered 139 hosts on ports 21, 53, 80, 143, 179, 443, 465, 587, 993, 10025, 10080, 20001, 20002, 26000 and 42512 and ZMap discovered 138 hosts on the same ports. In the second turn Nmap discovered 123 hosts on these ports, while ZMap discovered 122 on the same ports. The difference based on all ports is: one port in the first run and five ports in the second run. This results in a port cover percentage of:

$$1 - (\frac{1}{139} + \frac{5}{123}) = 95.21\%$$

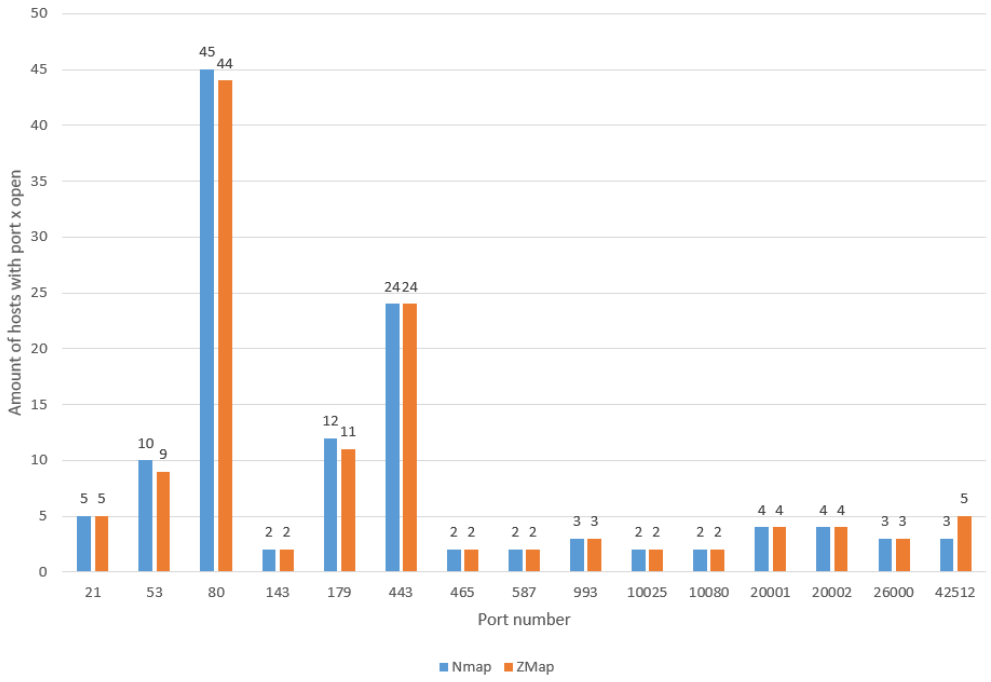

Figure 4.5: Comparing Nmap and ZMap by selected ports, performed from VM on the same day

So ZMap matches Nmap's port coverage in 95.21% of all cases, measured on the fifteen ports, where more than one host was detected.

4.3.5 Comparing Masscan and ZMap

We compared Masscan and ZMap to see if they produce similar results. Therefore the same ports as in section 4.3.4 were scanned with ZMap and Masscan and the results can be seen in figure 4.6 and 4.7.
As these figures show, the differences between ZMap and Masscan are very small. ZMap detected one host with an open 53 port more than Masscan, while Masscan detected one host more with an open 80 port and no hosts on port 10080 in the second run. Apart from that the detected ports per host are identical.
Scanning duration of Masscan and ZMap stayed the same as in chapter 4.3.3 and 4.3.4 with Masscan taking 40 to 50 minutes on the complete range and ZMap taking ten second per port on all hosts. Furthermore we performed all tests of this chapter within a day. In the first run a total of 138 hosts were detected on the 15 ports by ZMap and in the second run 122. Differing in two and eight hosts in the first and second run, the host per port coverage of Masscan compared to ZMap was:

$$1 - (\frac{2}{138} + \frac{8}{122}) = 91.99\%$$

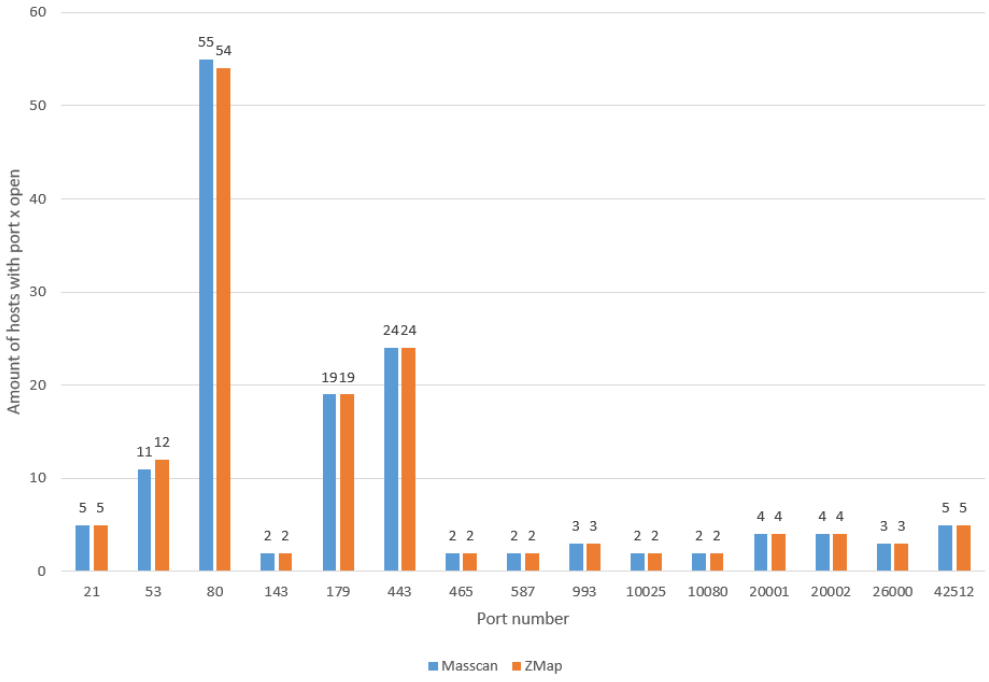

Figure 4.6: Comparing Masscan and ZMap by selected ports, performed from VM.

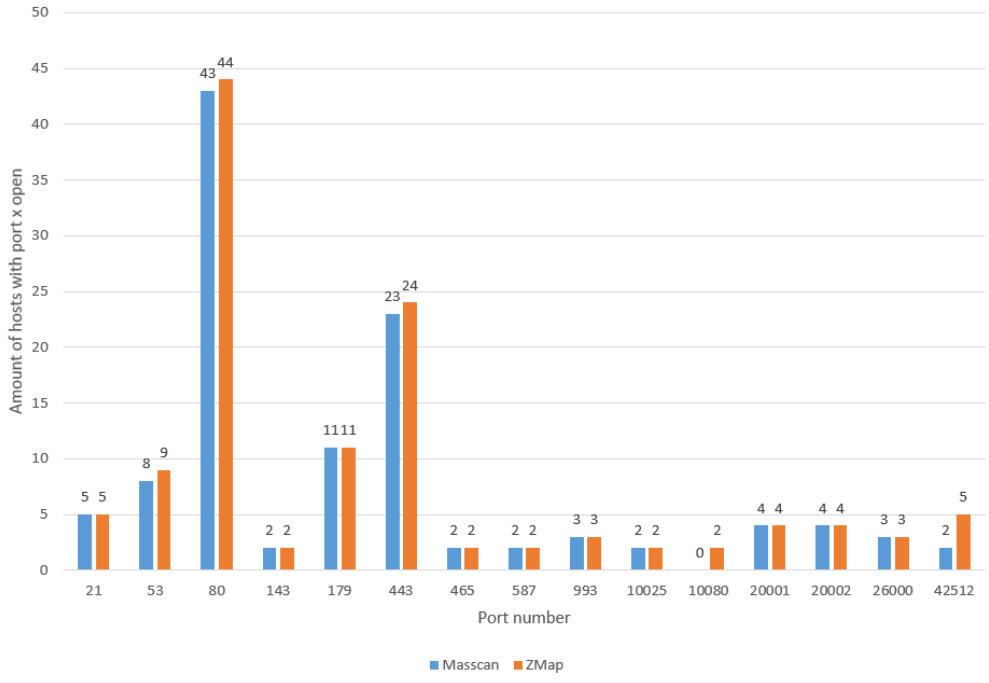

Figure 4.7: Comparing Masscan and ZMap by selected ports, performed from VM.

So Masscan equals ZMap in 91.99%, regarding host per port detection on the fifteen selected ports.

4.4 Evaluation of Nmap, ZMap and Masscan test scans

In this section, we collect all arguments for choosing a certain scanner and close with reasons why Masscan was chosen for this thesis.

4.4.1 Duplicates

As mentioned in section 4.2 and 4.3.1 the major difference, regarding defining scanning arguments between ZMap and Masscan is, that ZMap can only scan one port in the network at a time, while Masscan as well as Nmap, can already scan every port from 0 to 65535 simultaneously. According to one of the developers of ZMap, this design was chosen because the problem of duplicates in the scan results couldn't be solved satisfactorily, when ZMap was adapted to scan port ranges [6]. When scanning with Masscan, some scan results occurred several times (see section 4.3.1), so by adapting ZMap to all ports at once, the output of ZMap and Masscan would equally be filled with a few duplicates entries. From that point of view, it does not make sense to alter ZMap into a tool with the same output as Masscan.

4.4.2 Accuracy

The most likely scenario where the developed tool will be used, is when scanning the MWN from a VM from the LRZ, so the results from the hardware can not be taken into account here.
More interesting were the two Nmap scans from the VM compared with the scans from the other tools, which has shown that Nmap and Masscan were the most comparable tools - despite their immense difference in speed.
With the calculations from section 4.3.3 and section 4.3.4, we showed that Masscan compared to Nmap has a port cover percentage of 94.93%, while ZMap reaches 95.21%. Therefore ZMap is slightly more accurate compared to Masscan and both scanner compared to Nmap, keeping in mind that comparing ZMap to the other two is rather difficult as it can only scan one port per run. These results are also supported by the research of David Adrian et al. published in their paper "Zippier ZMap: Internet-Wide Scanning at 10 Gbps" [15].

4.4.3 Speed

With respect to speed, both scanners can reach beyond the 100 Mbps limit, which is required for fulfilling the tasks of this thesis. As mentioned in [15] it must be stated, that ZMap can speed up to 14 million packets per second, while Masscan reaches only 7 million packets per second. But these packet rates would require far too much

bandwidth, so they could be rated equally, as both easily fulfill the 100 Mbps limit. But ZMap needs to initialize each time when addressing another port. So we concluded, that when scanning port ranges, Masscan is currently faster than ZMap.

4.4.4 Documentation

Both tools were developed as open source tools, published on github. But ZMap was developed at the University of Michigan, resulting in many research paper about ZMap like [2, 6, 15] and a larger documentation about it, than Masscan. Masscan has only a README file along with the provided C code, so ZMap is better documented in general.

4.4.5 Features

We found, that both scanners provide blacklisting for excluding certain IP ranges, as well as running the scan using parameters defined in configuration files. Although both support banner grabbing, "ZMap and banner-grab can have significant performance and accuracy impact on one another if run simultaneously." [6] There is a significant difference between the scanners, as Masscan can perform a SSL 3.0 detection, required to fulfill the scanning tasks in one step. As service and version detection are very important requirements for fulfilling the tasks addressed in the thesis, Masscan is better suited for the further approach regarding its features.

4.5 Reasons for choosing Masscan

We chose Masscan for further development as it can already scan port ranges, needs less time to finish the addressed tasks than ZMap and has the features needed to fulfill the scanning in one step, like SSL 3.0 detection. Even though ZMap is slightly more accurate and better documented in general, altering ZMap for scanning port ranges would result in a tool very similar to Masscan, also not having solved the problems of duplicate entries in the output files.

Chapter 5

The Fast Port Scanning Tool

This chapter provides an overview about the planing and implementation phase of the scanning tool. Furthermore we explain certain design decisions and the modes the tool supports. According to its purpose we named the tool the “Fast Port Scanning Tool”.

5.1 Planning Phase

Here we explain the steps before implementation.

5.1.1 First drafts

While running the comparison scans with Nmap, Zmap and Masscan, a simple help function was written in Python 2.7 (see Chapter 4.1) to eliminate duplicate entries and enable easy comparison between scanning results.
Starting with this simple tool, which reads IP addresses and ports from an xml file and writes these, sorted by IP address and port to a txt file, the requirements for the port scanning tool were gathered.
We agreed the tool should also be written in Python 2.7 and that the port scanners should be called from the tool, executed and the result written to .csv files for further processing. Deciding which information is necessary was the next step, summarized in table 5.1. With the existing tools and programs, port scanning itself could be performed but for a fast detection of SSLv3 fallback, there was no satisfying solution yet.
OpenSSL [45] seemed to be a good choice for checking if a host could still accept SSLv3 or only newer TLS 1.X requests. But checking every host with OpenSSL for the oldest SSL version available on host side took too long, which is five seconds per host.
As OpenSSL was not a good choice, the NSE [4] seemed promising, as there are already existing scripts designed for checking for POODLE [4]. But even with the fastest options, this took too long, once again with about 5 seconds per host.
In the end, we chose Masscan for this task as it also provides an additional script for checking for SSLv3 fallback [7].

Entry	Description
IP address	This entry is the IP address of the respective scanned host.
Port	This entry holds an open port detected on the host.
Service	This entry shows which applications run on this port
Banner	This entry provides additional information regarding running services such as, e.g., version numbers.
OS	This entry exists only if the host was scanned by Nmap as Masscan doesn't provide OS detection.
Vulnerability	As this thesis only takes a closer look on POODLE, the standard vulnerability is POODLE.
SSL 3.0 vulnerable Y/N	This entry indicates whether the host is vulnerable to SSLv3 fallback or not.

Table 5.1: Overview about important information from the scans

5.1.2 Requirements for the Fast Port Scanning Tool

The cases the tool must be able to deal with, are the following:

1. IP range: Hosts for scanning can be set in a separate configuration file.
2. Excluded IPs: IPs, which are private [44] or should not be scanned can be set in a separate exclude file.
3. Scan Parameters: Together with the IP range, scan parameters can be set in the same configuration file for Masscan. Nmap's parameter are hard coded as the tool requires Nmap's OS and service detection to work correctly.
4. Output: The tool always produces the specified .csv file as defined above and optionally a xml or a txt file containing the output from the scanner itself. The name of the output from Masscan or Nmap can be defined by the user or taken from the default value from the tool.
5. Fast overview of the network: If only one scan is required, either a fast Masscan scan is executed once, or a Nmap scan is appended for more service and especially OS detection.
6. Continuous scans: In a predefined period of time, scanning repeats in the same user-set time intervals until the user interrupts.
7. Scheduler for scanning: The user can input any time and date, when scanning should start.

5.1.3 Decision for multiple scanning modes

Because of the time differences regarding the scanners, the tool would greatly benefit from two different modes for different occasions.

We decided for a fast mode, performing one or multiple Masscan scans, depending on the user's choice, providing a fast overview about activities in the network. On the other hand we designed a complete mode, scanning with Masscan and Nmap, taking approximately two to three times longer as the fast mode, but also providing detailed service and OS detection.
The reason behind dividing the tool into two modes, is time-efficiency as it is estimated that a complete scan of 1800 /24 subnets at 100 Mbps on scanner side, takes about 40 hours with Masscan, while appending Nmap takes approximately 40 hours additionally. The calculus for the duration of one Masscan scan is shown here, with the assumption that one packet per port suffices:

$$1800/24 \text{ subnets} \cdot 256 \text{ hosts per subnet} = 460,800 \text{ hosts}$$
$$460,800 \text{ hosts} \cdot 65,535 \text{ ports} = 30,198,528,000 \text{ total ports}$$
$$100,000,000 \text{ Bits} : (54 \cdot 8) \text{ bits} = 231,481 \text{ packets per second}$$
$$30,198,528,000 \text{ ports} : 231,481 \text{ packets per second} = 130,457 \text{ seconds}$$

Thus 130,457 seconds are required for one fast scan, which is about 36 hours. So if only a fast overview of the network is required, the fast scan mode suffices.
These two modes should fulfill all properties addressed above and with the gathered requirements (see section 5.1.2) implementation could start.

5.2 Implementation phase

This part describes the different methods of the Fast Port Scanning Tool, why they were designed that way and how they work together. First we will explain the defined classes and then the functions used in the Fast Port Scanning Tool.

5.2.1 Class MasscanTarget

The class *MasscanTarget* defines all properties for the *MasscanTarget* objects, containing four variables for IP (String), Port (Integer), Service (String) and Banner (String).

5.2.2 Class NmapTarget

This class is similar to *MasscanTarget* but applies to Nmap targets. The only difference between those two classes is an additional attribute *OS* for an Nmap target, as Nmap also performs OS detection.

5.2.3 Main function

As usual by calling the main function the program is executed. It starts with this prompt: "Welcome to the Fast Port Scanning Tool! :)"

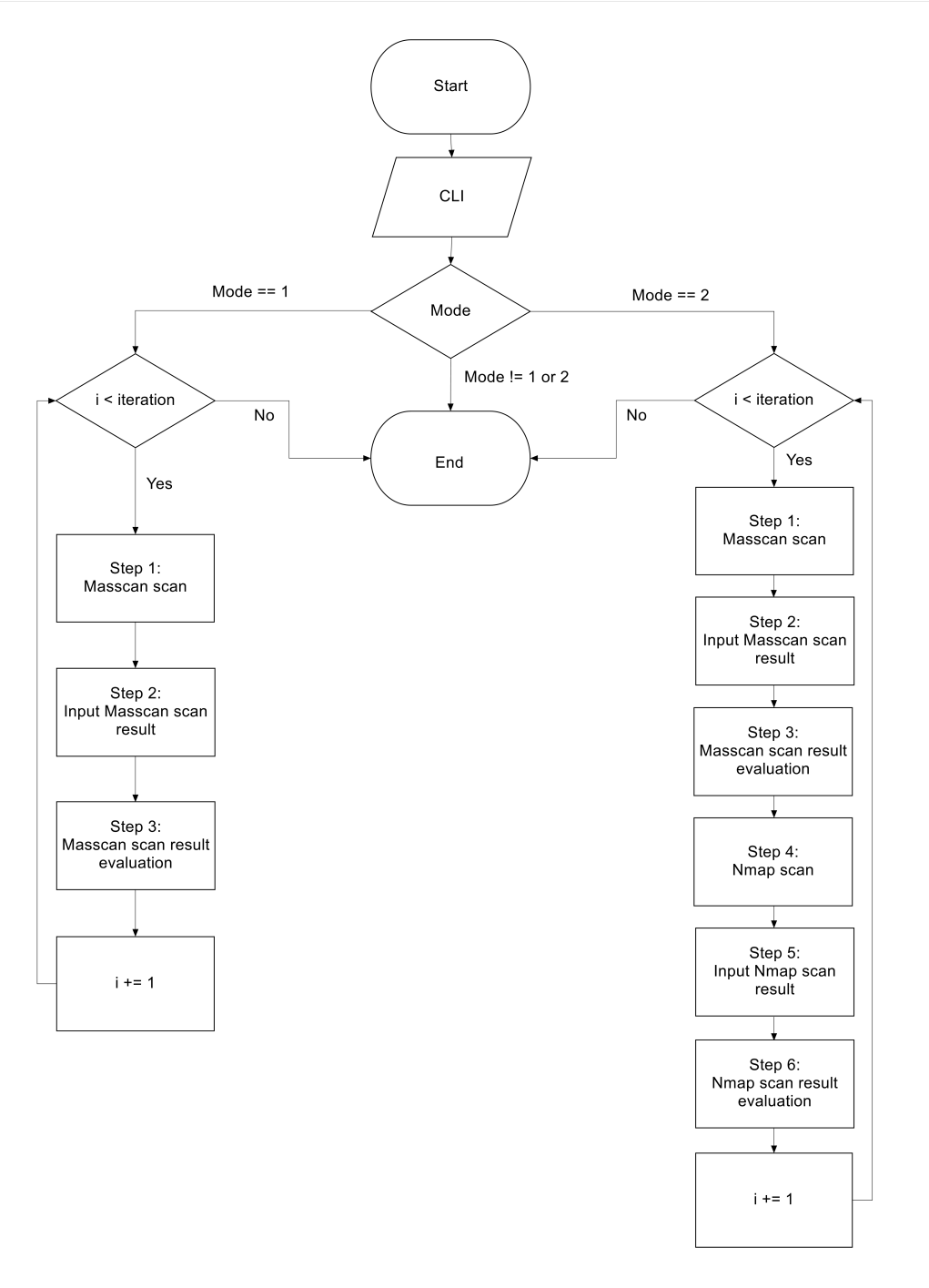

Figure 5.1: General overview about the workflow of the Fast Port Scanning Tool

Then the current time and date and the tool's directory is fetched. As the tool is started via console arguments, these are defined, when starting the tool. To prevent exceptions

through, for example, wrong parameter formatting, we wrote *define* functions for each parameter, which are described additionally with their respective parameters:

Path

Defines the path to the local Masscan executable. To enter the path is mandatory as the Masscan executable can be in different directories, depending on the user's choice where to save the Masscan folder to. The *define_path_masscan()* method tries to navigate into the directory of the Masscan executable and if that is possible, the path is returned. Otherwise the program exits, writing an error message to the console.

Mode

Defines the mode the program should be executed in (fast or complete). The *define_mode()* function allows only mode *1* or mode *2*, with *1* being the fast mode and *2* the complete mode. If any other argument is entered, an error message is written to the console and the tool exits.

Config

Sets the name of the configuration file for Masscan, that the tool will then use. The *define_config()* function checks if the entered Masscan configuration file exists in the *masscan/bin* directory. If there is no such file, an error message terminates the program. Otherwise the name of the configuration file is returned.

Start time

Entering a starting time makes the tool wait until that time. The *define_startime()* method allows only a start time in the format yyyy:mm:dd:hh:mm:ss. In case the format was correct, the difference between that date and the current time is calculated and saved in seconds to the variable *waittostart*, which is returned. If any other format is entered, an error message is shown on the console and the program exits.

Repeat

This argument defines the number of times the tool should be executed in total. The *define_iteration()* function checks the amounts of iterations for the Fast Port Scanning Tool and if the tool should run for a very long time, the parameter *-1* sets the number of runs to *4,000,000,000*. In the end the number of iterations is returned or an error occurs upon entering any other value than an integer.

Waiting time

The time the scanner waits in between the iterations of the scans. The *define_interval_time()* function checks the correct format hh:mm:ss, e.g., 168:00:00 for 168 hours of waiting time, being an entire week. The entered value is transformed to seconds and returned as interval waiting time and as above, if the format was wrong, the tool exits with an error message.

Additional file

Changing this enables the user to get different output (xml or txt) from the scanners. Furthermore, he can define, if he wants to keep the output from the scanners. The *define_additional_file()* method is responsible for output checking. Input can be: “xml”, “txt” or “none”. If one of these values were entered, *additional_file* is returned and if not, an error message is written on the console and the tool exits.

Outputmasscan

As the name says, the name of the Masscan output can be set here. The *define_outputmasscan()* function checks if the output of Masscan already exists in the tool’s folder with the exact same name. If so, the program prints a corresponding error message and exits and if not, the name of the Masscan output is returned.

Outputnmap

Naming the output of Nmap is made possible through this parameter. The *define_outputnmap()* method checks for an existing Nmap output file in the script’s directories, printing an error message if there is one, or simply returns the entered name.

These user entered parameters are stored in variables and checked, if they match the conditions. For that purpose the *define()* functions are used, described above in this section. If no exceptions were caused due to, for example, wrongly entered parameters by the user, the variable *i*, counting the iterations of the tool and the variable *scanresult* for storing output from the scanners, are initialized.
Figure 5.1 provides a rough overview about the Fast Port Scanning Tool workflow. In both modes, first a Masscan scan is performed, the results passed on to the *masscan_input()* function where the scan results are parsed. This information is then evaluated in *masscan_evaluate()* and the results written to a csv file. In mode two, in the complete mode, the discovered ports and hosts by Masscan are passed to Nmap, which performs a scan (*nmap_scan()*) and returns the results to *nmap_input()*. This method parses the information and returns it to *nmap_evaluate()*, which finally writes the Nmap scan results to a csv file. The functions for Masscan scan, input and evaluation and the functions for Nmap scan, input and evaluation are described below in detail.

5.2.4 Masscan_scan function

The function is responsible for performing Masscan scans. First the directory is changed to that of the Masscan executable and, depending on the user’s choice for output, the scans can produce different output.
If a xml output is required, Masscan is called via *subprocess.call()* (imported from the *subprocess* library) with the scan parameters from the configuration file defined in the

variable *config*, the command for checking for SSL 3.0 fallback [1] and the command for writing to a xml file[2] applied.
In any other output cases than xml, Masscan is executed via *subprocess.Popen*, imported from the *subprocess* library, with the same parameters, except `-oX name`. Then the scan results are stored in a String variable, which is returned for further processing.

5.2.5 Masscan_input function

In this method, the important information from either a variable or a xml file, depending on the user's output format, described in section 5.2.4, is parsed.
As explained in section 5.1.1, we need only IP, port, service and banner as information about SSL 3.0 fallback is stored in the service section of Masscan. Via regular expression all appearances of these four parts are found either by reading in the xml file and saving the file's content in a variable *data* or by directly evaluating the scan result by processing the returned variable *scanresult* from *masscan_scan()*.
First an IPv4 address consists of four blocks of one to three decimals, separated by a dot. Secondly, a port consists of one to five decimals and is located always at the same spot. Thirdly and fourthly service and banner can be any letter or decimal in Unicode (usually in UTF-8 formatting). But as a service in Masscan is usually not described with more than twelve letters and even long entries in the banner section don't use more than 10.000 letters, these two upper boundaries should suffice to capture any important information.
All the information is saved in a list *compare*, with each IP, port, service and banner in one entry. Finally the list *compare* is returned.

5.2.6 Masscan_evaluate function

The returned list *compare* from *masscan_input()* is split up into a list of *MasscanTargets*, each *target* in the list storing one IP address, port, service and banner.
Furthermore in these processes, new unique IP addresses are written into a separate list for sorting the information by IP. This list is sorted and a new list *ipfinal* is created. Every time there is a match between an IP address in the sorted IP list and an IP address of a *target*, the *target* is appended to *ipfinal* thus filling *ipfinal*.
But due to the Masscan work flow, duplicate information is still in *ipfinal*. Which indicators were there for a duplicate entry and how to traverse so many elements (more than 160.000 in *ipfinal*) efficiently was the next problem. The easy approach by building an inner and outer loop, both traversing all elements in the list, resulting in an effort of $O(n^2)$, was too slowly, as it took about 4452 seconds with that approach with 160,000 elements per list.
A better approach was to use the advantage, that the list is already sorted by IPs. So if

[1] `--script=poodle`
[2] `-oX name`

inner and outer IP and port match and one *item* has an entry for service and port, and the other has not, then remove the item with the lesser information. With this approach, only parts of the list starting with the same IP had to be traversed.
Through this method, removing duplicates could be speed up to 973 seconds for n = 160,000 elements.
Vulnerable hosts and ports are sorted in a help list, which is later returned for further processing in the Nmap functions.
At that point writing to the .csv file starts:
In the first step the .csv file is named with the user specified name and "_evaluation.csv". Then a file with this name is created and opened. After setting the fieldnames ("IP address", "Port", "Name of Service", "Banner", "Vulnerability", "SSLv3 [poodle] vulnerable") writing starts via a *DictWriter*. Each item's IP, port, service and banner in *ipfinal* is written out and depending on "vuln" or "safe" entries in the service section, the host and port are marked as vulnerable to SSL 3.0 fallback or not. With closing *file_out* and returning the vulnerable IPs and ports via the help list, this method ends.

5.2.7 Nmap_scan function

Nmap is only appended in the complete mode and for speed reasons checks only hosts, which were detected by Masscan in its scan. Thus Nmap can never detect more hosts than Masscan due to the workflow of this tool.
The method takes the *compare* list from Masscan, in which IP, port, service and banner are stored separately. With regular expressions each individual IP address and each single port get extracted from *compare* and stored to two different lists, *portlist*, containing only distinct ports and *iplist*, containing only distinct IPs.
All IPs are joined in the correct format for a Nmap scan, divided by a space, while ports are joined with a comma between them. By scanning a larger amount of hosts, it became clear, that Nmap and *subprocess.Popen()* can't accept too many arguments. The limitation was at 32,768 letters, while every detected port and IP together had more than 300,000 letters. So IPs were written to a separate file which is deleted after performing the Nmap scan. If more than 4000 different ports were detected, Nmap should scan the entire port range[3] and the IPs are read in from the file "selectedNmapIPs.txt"[4]. We chose the number of 4000 different ports to prevent crossing the 32,768 letter limit.
With these properties, the output of the Nmap scan is named correctly by appending the file format, either ".xml" or ".txt", to the user defined name.
Concerning the properties of the Nmap scan, these are set as follows:

[3] `-p-`
[4] `-iL selectedNmapIPs.txt`

```
nmap -PN -n -sV --version-light -O --osscan-guess -T4
--min-rate 10000 --max-rate 230000 -oX name.xml
--randomize-hosts -p ports -iL IPsofHosts.txt
```

So Nmap does not perform host detection, no DNS resolution, scans for services with an intensity of three, has six retries per host , a packet rate of 10,000 to 230,000 per second, saves the output to an xml file and randomizes the hosts. Regarding speed, we decided to use Nmap's speed regulation option *-T4* because when we scanned with the faster option *-T5* too many hosts were not detected [4]. This Nmap scan is saved to a file, which is made executable and executed, resulting in performing the Nmap Scan. We decided to use this method to ensure, that even long scan commands with about 30,000 letters can be processed.
In case the user requires an additional txt file, this parameter is appended to the Nmap commands and Nmap writes two files.
After the Nmap scan has finished, the temporary files are deleted.

5.2.8 Nmap_input function

As well as the Masscan input method, the Nmap input function follows the same principle. This scan output file is read in to a variable called *data* and divided into a list by a regular expression.
If the user didn't want a xml file, this file is deleted, as requested.
Following the same logic as described above, the important information including an OS is extracted and returned, using the variable *information*.

5.2.9 Nmap_evaluate function

This function is very similar to *masscan_evaluate()*. The information in the list *information* is stored into a list of lists called *iplist*, each item in the list represents a new IP.
In the next part, the information in the list of lists is written to *NmapTarget* objects, making writing to the .csv file very easy and equal to the method *masscan_evaluate()*.
The next part of sorting the entries of *iplist* by IP follows the same logic as in *masscan_evaluate()*.
The fieldnames used here for the .csv file are "IP address", "Port", "Name of Service", "Banner", "OS", "Vulnerability" and "SSLv3 [poodle] vulnerable". Iterating through the list of *NmapTargets* allows sorting each *target*'s IP, port, service, banner and OS to the respective column. If the IP of the current item is in the list *ssl_vuln*, returned from *masscan_evaluate()*, the host is listed as "vulnerable" otherwise as "safe".
By closing the *csvfile*, the function ends.

5.2.10 Get_time function

The current time is fetched and then formatted into day_month_year-hour_minute_second, as the names of the default output files start with this date-time part for further data maintenance.

5.3 Hints for working with the tool

For additional explanation how to work with this tool, we provide background material in this section.

5.3.1 General advice

As this tool is based on Masscan, which is able to send 7,000,000 packets per second, it is a good idea to tell the responsible administrators of the respective networks of the scanning plans.

5.3.2 Tweaking Masscan

As this tool uses Masscan and especially the script for checking for POODLE [5], some files in the *masscan/src* directory can be altered slightly to match the requirements of this tool. In the file "proto-banner1.c" we find the following lines:

```
b->tcp_payloads[80] = &banner_http;
b->tcp_payloads[8080] = &banner_http;
...
b->tcp_payloads[8140] = (void*)&banner_ssl;   /* puppet */
return b;
```

We changed these as follows, because every port should be checked for SSL 3.0 fallback:

```
for(i=0; i<65536; i++)
    b->tcp_payloads[i] = (void*)&banner_ssl;
return b;
```

Now Masscan does not only check the default ports for SSLv3 fallback, but every port, if any connection via the SSL/TLS encryption was found.

5.3.3 Summary of arguments

Here an overview of the possible arguments is given:

1. help: Shows the help message and exits.

[5] --script=poodle

2. `-p PATH`, `--path PATH`: Defines the path to your local Masscan Executable. This path is mandatory! E.g., `--path /root/masscan/bin` if there is the Masscan executable.

3. `-m 1,2`, `--mode 1,2`: Defines the mode the program should be executed in. Default values are "1" for fast mode (only Masscan scan), "2" for complete mode (Masscan and Nmap scans executed in a row).

4. `-c CONFIG`, `--config CONFIG`: Defines the configuration file for Masscan the tool will use. The default name is myscan.conf E.g. --config myscan.conf. All parameters for the Masscan scan come now from myscan.conf.

5. `-st STARTTIME`, `--starttime STARTTIME`: Defines the date and time, when the scan should start. The default value is no waiting time until scan starts. The correct format is yyyy:mm:dd:hh:mm:ss, e.g., 2015:01:06:17:11:00 would be 6th January 2015 at 5:11:00 pm.

6. `-r REPEAT`, `--repeat REPEAT`: Defines the amount of scans, the program should perform. The default value is one iteration for one scan E.g. `--repeat 2` for 2 iterations or `--repeat -1` for 4,000,000,000 iterations.

7. `-w WAIT`, `--wait WAIT`: Defines the waiting time between scan iterations. The default value is no waiting time between scan iterations. Correct formatting is hh:mm:ss E.g. 100:50:30 would make the program stop between scans for 100h, 50min and 30 seconds.

8. `-af ADDITIONALFILE`, `--additionalfile ADDITIONALFILE`: Defines the selected additional output file formats from the scanners. The default file is the xml file and other file formats can be entered like , e.g., `--additionalfile txt` for txt file.

9. `-om OUTPUTMASSCAN`, `--outputmasscan OUTPUTMASSCAN`: This This parameter is responsible for renaming the output of Masscan scans of the tool. The default name is date_time-masscan_scan.type, e.g., 10_01_2015-14_00_00-masscan_scan.xml. For use: Simply enter a name without type, e.g., masscan_scan. The output is saved to the Tool's directory.

10. `-on OUTPUTNMAP`, `--outputnmap OUTPUTNMAP`: This parameter is responsible for naming the output of Nmap scans of the tool. The default value is date_time-nmap_scan.type, e.g., 10_01_2015-14_00_00-namp.xml. For use simply name the file without giving it a type: E.g. nmap_scan. The output is saved to the Tool's directory.

Chapter 6

Evaluation of MWN scans

As the most important research insights came from scanning the MWN, in this chapter we give an overview about the results. First we mention the scanning preparations and how scanning was performed. Then the results are gathered and evaluated.

6.1 Scan preparations

Following the guidelines from chapter 4.1 concerning setting parameters for the test environment, we altered the parameters as follows:

1. Network: the scanned network is the MWN with 1,640,832 possible hosts and 461,184 IPs for scanning. We selected these, as they are mostly used by institutions like the TUM or the LMU as, for example, servers (see chapter 3.1). These 461,184 IPs are separated into the following subnets: seven /16s, one /22s, five /24s and one /25 subnets.

2. Location: all scans were performed from the virtual machine "lxdps04.srv.lrz.de" located at the LRZ.

3. Rate Limiting: regarding Masscan and Nmap, all scans were performed at a maximum of 100 Mbps, being equal to 231,481 packets per second (see chapter 4.1.2).

4. Output: every scan consisted of one Masscan and one Nmap scan, resulting in five xml and csv files, containing the desired information.

5. Files as parameter input source: for Masscan, the scan parameter are saved to a configuration file while Nmap's parameters are set in the Fast Port Scanning Tool (see section 5.1.2).

6. Plausibility and reliability: as the three different scan runs contained very similar results, the results can be regarded as plausible.

With these parameters being set, scanning started in a “tmux” session [40] from the virtual machine to prevent connection loss and thus terminating the tool. Table 6.1 contains the parameter used for scanning with Masscan and Nmap.

Scanner	Parameters
Masscan	`masscan -p0-65535 -oX outputname.xml --max-rate 230000 --script=poodle IP_ranges`
Nmap	`nmap -PN -n -sV --version-light -O --osscan-guess -T4 --min-rate 10000 --max-rate 230000 -oX outputname.xml --randomize-hosts -p ports -iL IPsofHosts.txt`

Table 6.1: Scan parameters for scanning the MWN

Furthermore Masscan’s parameters were set in a configuration file via the `-c configurationfile.conf` command. Of course, the parameter `outputname.xml` is a wildcard for a real name for scan outputs, `configurationfile.conf` a wildcard for a real configuration file name and `IP_ranges` a wildcard for the real IPs of the MWN.
The standard procedure was to open a tmux session, set the parameters and detach it. Every five to ten hours the scanner was checked for problems, like scanning disruption.

6.2 Results of the MWN scans

After two weeks of scanning, we gathered the results of the three scans. Regarding all following figures and results, we used Masscan and Nmap TCP SYN scans for gathering information about hosts and ports, Masscan’s banner detection and Nmap’s service detection for service and version detection, Masscan’s POODLE script for detecting hosts vulnerable to POODLE and Nmap’s OS detection for collecting information about used OSes in the MWN. The following subsections provide statistical results from the scans.

6.2.1 Overview of total results

The first Masscan output file contained 161,003 entries, resulting in 123,428 entries without duplicates, the second Masscan output 163,968 file with 125,501 entries without duplicates and the third 163,621 with 125,416 entries without duplicates (see chapter 3.2 and 4.3 for duplicate occurrence explanation).
With the Fast Port Scanning Tool, Nmap scanned ports and hosts in the first scan. As taking too long, the second scan was interrupted, but the third scan provided good results again.
An important fact to remember for the following is, that the Nmap input comes from the Masscan output so Nmap can never detect more hosts or ports than Masscan.
Interesting was especially the first Nmap scan because of the very low diversity in ports. We consider it as highly probable that this is due to the start time of the scan, as it started

Scan	Duration	Hosts	Ports
Masscan scan 1	33h 31m 54s	9377	46982
Nmap scan 1	11h 52m 27s	1320	638
Masscan scan 2	33h 21m 28s	10186	46747
Nmap scan 2	0 (interrupt)	0 (interrupt)	0 (interrupt)
Masscan scan 3	32h 17m 54s	10230	47040
Nmap scan 3	68h 44m 42s	2360	46671

Table 6.2: Overview of scanning duration and results regarding Masscan and Nmap

on a weekend at night and finished before the begin of a work day. Furthermore the high diversity of ports comes mostly from three hosts with more than 30.000 open ports and these were not running during the first Nmap scan time. Also the large difference in amount of detected hosts between Masscan and Nmap was unexpected. The only explanation for this is that due to the host-timeout parameter set in Nmap, Nmap gave up on most of the ports, which were unavailable at that time. As the scans were also performed during night time, many hosts were probably shut down.

6.2.2 Summary of basic results

Beginning with the hit rate of detected hosts, referring to all detected hosts divided by all scanned hosts, we calculated, that Masscan's hit rate is $\frac{10500}{461384} = 2,2\%$. This value seems plausible as in the scanner evaluation phase (chapter 4), this value was $\frac{61}{2048} = 2,98\%$. Another sign of the plausibility of the low total hit rate can be found in the papers [6,15], as these papers compared the hit rate of Masscan to that of Zmap and found an equal hit rate for Masscan.
Astonishing was the diversity of detected distinct ports, as about 70% of all possible 65536 ports were listed by each Masscan Scan (46982, 46747 and 47040 distinct ports). Here we provide the corresponding calculation: $\frac{46923}{65536} = 71.6\%$.
Continuing with the evaluation of the POODLE results leads to an interesting fact: Only 1,7% of all detected ports or services were vulnerable to SSL 3.0 fallback, which means that an average of 18% of all detected hosts by Masscan and only 0,39% of all scanned hosts are vulnerable to POODLE. We think the low hit rate for the POODLE vulnerability of 0,39% regarding all scanned hosts is due to the low total hit rate of Masscan.
With these results, the Fast Port Scanning Tool seems to fulfill its properties leading to good results and being $\frac{5*30*24h}{36h} = 100$ times faster than before, with Nmap alone
Two other outcomes of the scans were that a web server in the department of medicine was not reachable during the Nmap scans and needed therefore updating, because a Nmap scan with a very low packet rate compared, e.g., to Masscan, was enough to make this server unreachable. Another revelation was that the RBG network's ARP table at the department of informatics and mathematics is too small for a large amount

of requests (> 200.000 per second) sent from the Internet. The responsible staff for the RBG network were able to fix the issue, therefore reinforcing the university's network architecture.

6.2.3 Most noticeable ports, hosts and operating systems

Figure 6.1 shows the three hosts, which had the most open ports by far. As detected in three scans over a week, these hosts seem to be permanently up and have 16,000 respectively 36,000 of all their ports open.

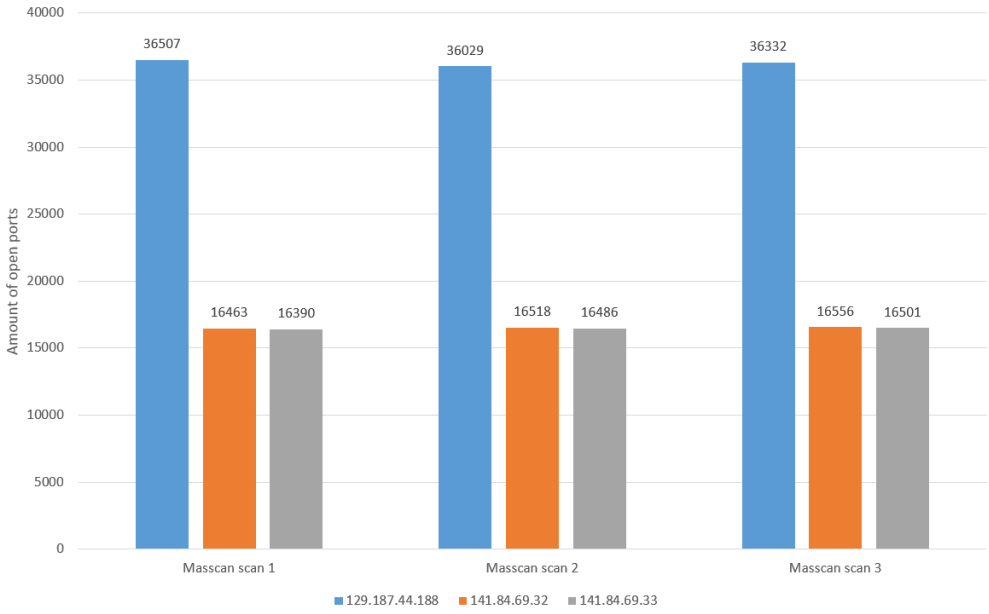

Figure 6.1: Hosts with the most open ports, detected by Masscan and Nmap

These hosts, shown in figure 6.1, regarding the many open ports on those three machines, are very likely servers as "Apache HTTP server" runs on several ports and also OpenSSH. Additionally the last Nmap scan detected their OSes as Linux 2.6.18 (129.187.44.188) and Linux 2.6.38 - 3.2 (141.84.69.32 and 141.84.69.32) all suited for a server machine.
Another possibility would be that the user or administrator responsible for that machine uses a white listing policy instead of a black listing policy for, e.g., a firewall, disguising the true port states. Still it seems as if these machines pose a security hazard, because too many open ports were detected by all Masscan and Nmap scans. This allows an attacker to take many attack points and the question remains if all of these open ports are required simultaneously.
Figure 6.2 gives an overview about how many hosts had which amount of ports open and is based on the three Masscan scans.
Beginning with more than 2800 hosts with only one port open, the amount of hosts with more and more open ports diminishes with a small peak at 47 open ports regarding

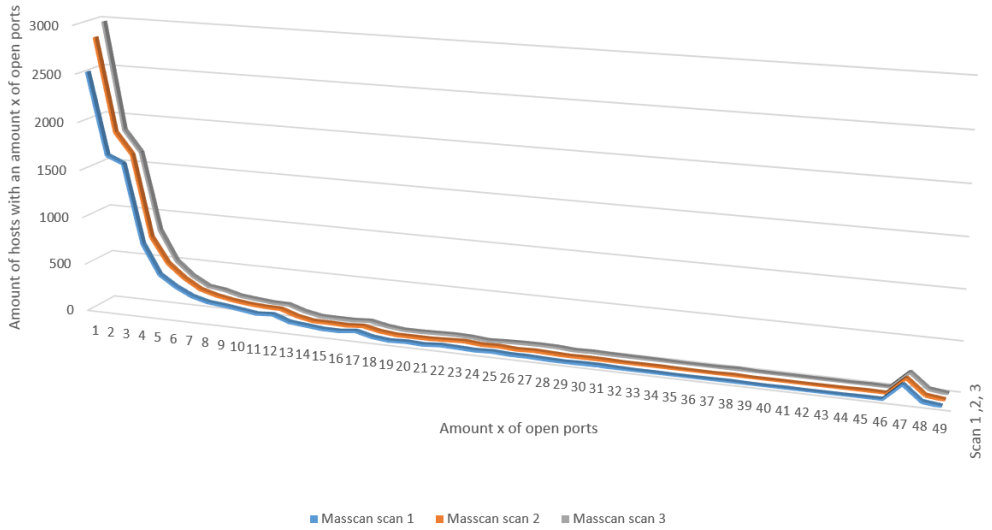

Figure 6.2: Average amount of open ports per host, detected by Masscan

all scans. As most of the hosts with 47 open ports were in the same subnet with the same ports open, it is very likely, that the machines were configured by the same staff, fulfilling a certain task. As Nmap detected on almost all of these hosts “McAfee Email Gateway” as embedded OS from “McAfee”, we regard them with a high likeliness as mail servers.

Regarding ports, the most as open detected ports, taken from an average value of all three Masscan scans, are summed up in figure 6.3. Concerning services, Masscan didn’t find a corresponding service to a port in about 90% of all cases. But the remaining 10% of services is shown in figure 6.4. They were all detected via the Masscan “banner” method and evaluation was performed on all scan results with a self developed Python tool. We calculated the values in figure 6.4 and 6.5 by building an average value of detected services in all three scan runs and divided it to the total amount of known services, which was 9619 regarding Masscan and 2240 regarding Nmap. Thus we conclude that Nmap has a higher service detection rate than Masscan, which seems plausible due to the different detection methods of Nmap (see chapter 3.3.1).

For comparison reasons, the services detected by Nmap are shown in figure 6.5. Nmap also revealed the different used operating systems, leading to figure 6.6, where the ten most used operating systems are rated in percent.

We compared figure 6.3 and 6.4 and found a significant coherence between port and service. The most services appear to be running on the Internet Address Name Allocation (IANA) defined ports [46]. Port 80 correlates with the HTTP service, port 443 with SSL and port 22 with SSH. Additionally to port 443, SSL was detected to be running

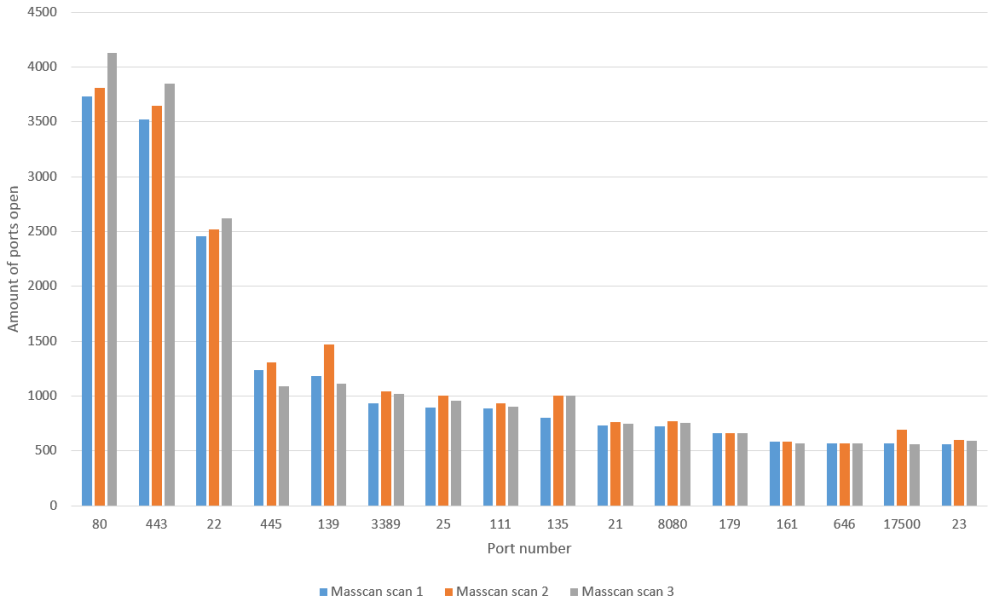

Figure 6.3: Most frequently detected open ports in the MWN, scanned by Masscan

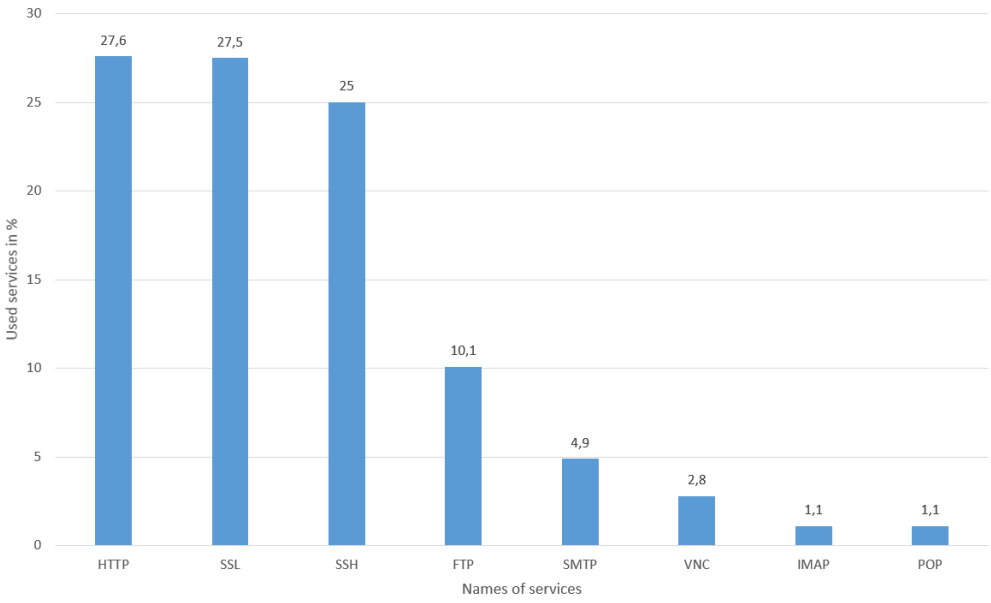

Figure 6.4: The most used services in the MWN, detected by Masscan

on 21, 25, 2222, 7512, 7780, 17500, 34463 and several other ports not mentioned due to their irrelevance. Apart from running on port 80, HTTP was also detected frequently on port 8080, SSH was detected on 22, 23, 22157, 22394, 55122 and on port 21 the File Transfer Protocol (FTP) service was running. As Masscan could not name the most

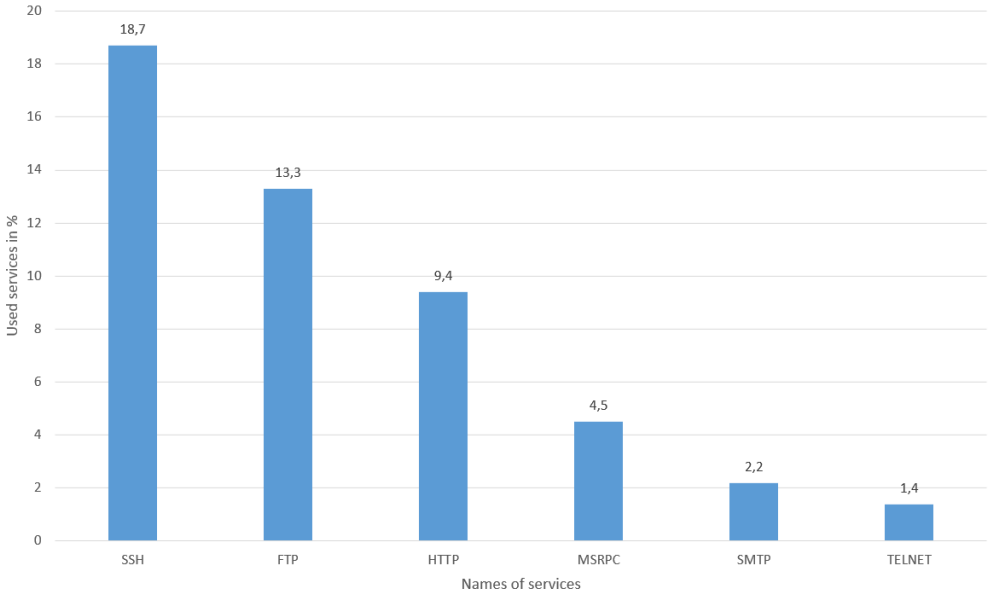

Figure 6.5: The most used services in the MWN, detected by Nmap

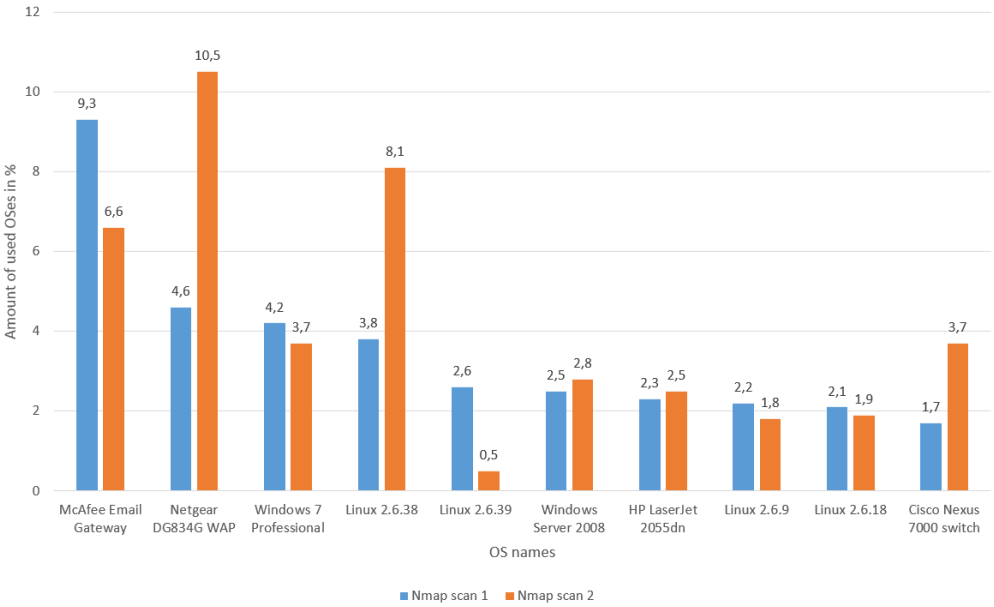

Figure 6.6: Top ten most used operating systems in the MWN, detected by Nmap

services running on the other ports, Nmap was used to get information about port 25, 111, 139, 161, 179, 445, 646 and 3389. As port 25 is assigned to the SMTP protocol by IANA, it was no surprise to detect mostly the SMTP protocol running on port 25. On port 111 no instance of Sun's Remote Procedure Call (RPC) as defined by IANA [46]

could be verified by Nmap but also no other service was found here.
On port 139 the service “Samba smbd” was detected several times but also in half of all cases no specific service could be evaluated by Nmap. Port 161 also remained empty in terms of detected service by Nmap, while on port 179 the Border Gateway Protocol (BGP), as IANA assigned this port to that service, was not detected implicitly by Nmap. Also for the ports 445, 646 and 3389, Nmap revealed no services. Regarding the services Virtual Network Computing (VNC), Internet Message Access Protocol (IMAP) and Post Office Protocol (POP) we found that “IMAP” was detected on port 143 by Masscan and Nmap, following IANA’s guidelines [46]. POP was detected mostly on port 110 and VNC on port 5900. With this summary we showed the correlation between the most open ports and most detected services.
When comparing figure 6.4 and 6.5, thus comparing the various services detected by Masscan and Nmap, we found a significant difference in the results. A part of the large difference can be explained in the overall difference of detected hosts, as Nmap discovered only 13% of all Masscan detected hosts in the first run and 23% in the second run. Furthermore service detection of Nmap can detect up to 3000 unique services, while Masscan is yet limited to 11 different services [4, 7]. Thus the results of Nmap are distributed over a larger variety of services, while Masscan focuses on the few services it can detect.
The next issue is the correlation between OSes and ports and services, starting with figure 6.6, showing the most used OSes in the MWN As, for example, “SSH” or “FTP” are Linux based services, the four Linux operating systems in the top ten most used OSes seem plausible. Also with many services related to E-Mail transfer, the most used OS, the embedded “McAfee Email Gateway” is realistic. As the rest of the most detected services can run on any of the other systems, these results correlate well between OS and services.

6.2.4 POODLE results

For this purpose the Masscan script “poodle” was used and the Masscan code was tweaked as described in chapter 5.3.2. After the first Masscan scan 2534 ports/services were detected to be vulnerable for SSL 3.0 fallback, the second Masscan scan detected 2683 and the third 2722. When filtering all vulnerable ports/services from all three scans, 2182 ports/services remained. Filtering was performed once again with a self written python script, taking the three Masscan scans as input and compared all vulnerable IPs with each other, returning the IPs shown as vulnerable from all three scans.
The different hosts and ports (or services running on that port) vulnerable to POODLE are shown in figure 6.7, while the most vulnerable hosts in terms of amount of ports susceptible to POODLE are listed in figure 6.8.
A ranking for the ports most mentioned as vulnerable to POODLE, is seen in figure 6.9 and a discussion about the results is presented below in this chapter.
Concerning figure 6.7, showing the amount of different hosts and ports detected as

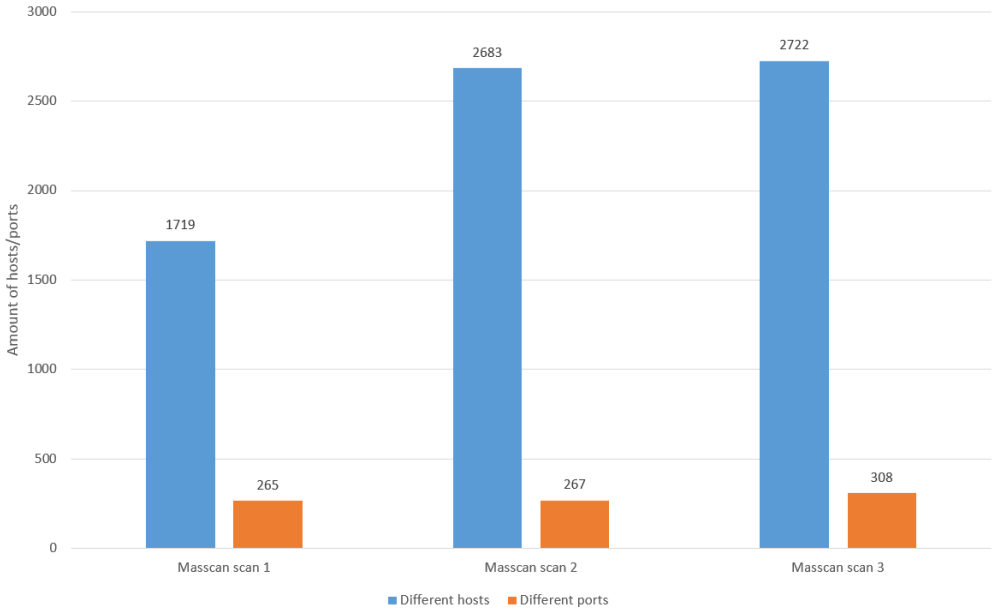

Figure 6.7: Overview of different hosts and ports vulnerable to POODLE, detected by Masscan

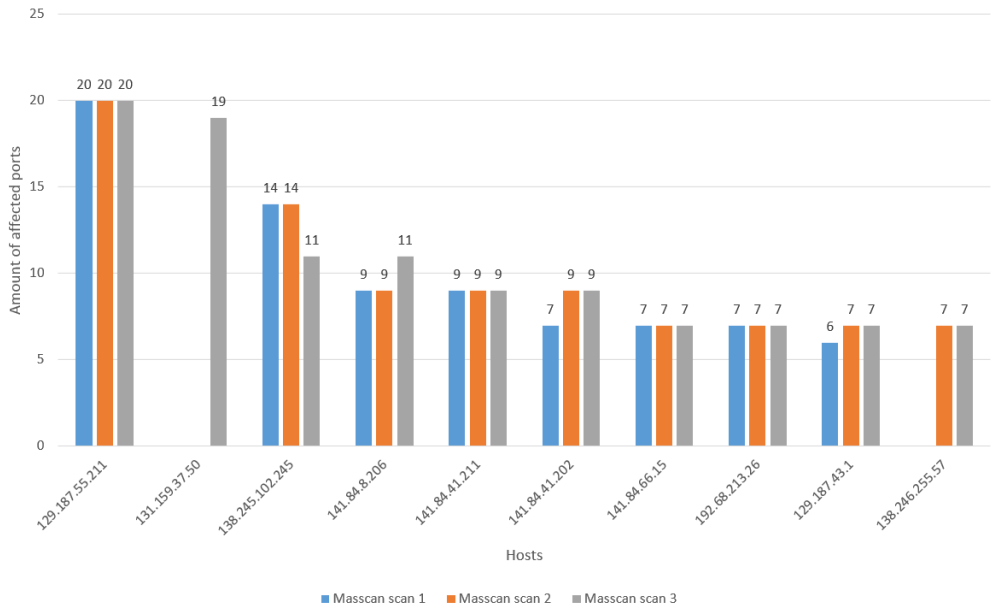

Figure 6.8: Hosts with the most services vulnerable to POODLE, detected by Masscan

vulnerable to POODLE by Masscan, the difference in terms of host detection between the three scans is probably due to the different scanning times. We performed the first scan on a weekend, while scan two and three were performed during the week.
Figure 6.8 shows the most affected hosts in terms of amount of affected ports. In the first

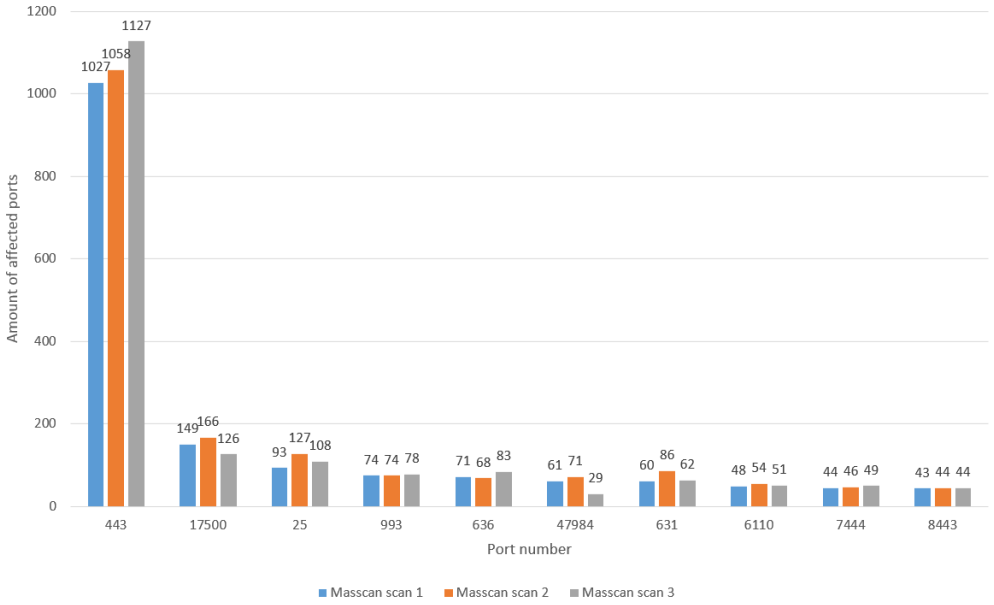

Figure 6.9: Ports most vulnerable to POODLE, detected by Masscan

and second scan, the host 131.159.37.50 could not be detected, as well as in the first run the host 138.246.255.57, which is the reason why we didn't list them in the first/second run. As stated in the paragraph above, these hosts were probably not detected, as the first scan took place on a weekend.
Regarding the top ten most affected hosts, they were running on Linux 3.0, Linux 3.1 and Sun Solaris 9/10 and four were in the same subnet 141.84.0.0/16. With these values, the subnet of 141.84.0.0/16 with the OS Sun Solaris 9/10 is the most vulnerable constellation, followed by Linux 3.0.
We show in figure 6.9. the most affected ports, beginning with port 443 and 17500. This correlates well with the coherence between ports and services explained in section 6.2.2. Also on these ports the services FTP, SSH, HTTPS, IMAP and SMTP were detected. We found another interesting detail, as Masscan detected on many ports just the service SSL, while Nmap could not find any specific service. To provide more reasons for this behaviour, a detailed review of Nmap's and Masscan's service detection would be required, which is beyond the scope of this thesis.
Finally we gave a list of the affected hosts and ports to the responsible administrators, who can now update the respective hosts.

Chapter 7

Conclusion

In this thesis initially the use of port scanners and the challenges implicit in scanning a large research network, the MWN, were pointed out. Possible vulnerabilities regarding the SSL/TLS encryption used in the MWN, which should be detected by the port scans, were also listed.
As no satisfying solution regarding speed and vulnerability detection was available, three port scanners Nmap, Zmap and Masscan were evaluated for their accuracy, speed and use when scanning a decentralized, organized university network. The evaluation revealed, on the one hand, that all scanners were comparable regarding accuracy with only minor differences around 3% to 5% of non detected hosts and ports. On the other hand, the scanner differed strongly in speed and scanning features, as Masscan and Zmap exceeded the speed of Nmap up to 1300 times and Nmap provided by far more scanning possibilities than the other two.
Furthering the methodology, Masscan and Nmap were chosen due to their similar usability and aptitude for the specified tasks. With these scanners a new tool providing a solution to the issues of speed, accuracy and SSL/TLS vulnerability detection was developed, making evaluation of large port scan results easy.
With this tool the MWN was scanned and the results evaluated, revealing that 18% of all detected hosts were vulnerable to the POODLE vulnerability. Also Linux 3.0 and Sun Solaris 9/10 were evaluated as very often mentioned with SSL 3.0 fallback problems, while especially services like “HTTPS”, “FTP”, “IMAP” and “SMTP” were detected to be vulnerable. Furthermore we found some hosts, depicting a security hazard due to many open ports. These results provide now a possibility for the staff, responsible for the respective subnets to enhance security in the MWN, by closing the POODLE breach and all undesired open ports.

7.1 Future work

However the Fast Port Scanning Tool can be extended by making switching between IPv4 and IPv6 address space possible, as the tool only supports scanning for IPv4 yet.

In case IPv6 scanning is required, another solution regarding address exclusion must be developed, as otherwise the address space is simply too large. For further research the question remains how to cope with IPv6 address space as it is 2^{96} times larger than IPv4 and all available port scanner are far too slow for that size. Also firewall and IDS evasion could be added for a more stealthily approach, but the question remains if that is wanted and helpful because scanning a huge amount of addresses fast seems a more realistic scenario than scanning that amount of IPs surreptitiously and therefore far slower.
Furthermore parallelization of Masscan and Nmap can be applied to the tool, allowing for even more accurate information about ongoing activities within the network.
Lastly in January 2015 it was revealed that POODLE is also applicable on TLS 1.0 when load balancer of certain companies are used by SSL/TLS [28]. In the MWN these load balancers were such a small minority, that the amount was negligible but for a total scan for POODLE, they should also be detected by the scanning tool.
So this thesis shows some new areas of application with respect to port scanning, provides a good overview about the danger of SSL/TLS vulnerabilities and leaves a beneficial port scanning evaluation tool for administrators of large research networks

Appendix A

List of abbreviations

Here the meaning of all abbreviations used in the thesis is shown:

LRZ Leibniz Supercomputing Centre 1

MWN Munich Scientific Network 1

POODLE Padding Oracle On Downgraded Legacy Encryption 1

SSL Secure Socket Layer 1

TUM Technical University of Munich 2

NSE Nmap Scripting Engine 3

TLS Transport Layer Security 3

BEAST Browser Exploit Against Secure Transfer 4

BREACH Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext 4

CBC Chaining Block Cipher 4

CRIME Compression Ratio Info-leak Made Easy 4

IV Initialization Vector 4

SIGCOMM Special Interest Group on data COMMunications (ACM) 4

BAdW Bavarian Academy of Science 7

IP Internet Protocol 7

LMU Ludwig Maximilian University of Munich 7

TByte Terabyte 7

DFN Deutsches Forschungsnetzwerk 8

G-WiN Gigabit Wissenschaftsnetz 8

AS Autonomous System 9

DHCP Dynamic Host Configuration Protocol 9

DNS Domain Name System 9

IDS Intrusion Detection System 9

IPv4 Internet Protocol version 4 9

IPv6 Internet Protocol version 6 9

Mbps Megabits per second 9

AES Advanced Encryption Standard 18

VM Virtual Machine 22

MAC address Media Access Control address 27

IANA Internet Address Name Allocation 51

FTP File Transfer Protocol 52

RPC Remote Procedure Call 53

BGP Border Gateway Protocol 54

IMAP Internet Message Access Protocol 54

POP Post Office Protocol 54

VNC Virtual Network Computing 54

Appendix B

Scanner parameters

Regarding Nmap's scanner commands, the following selection was used often during the thesis:

1. -p[range]: This specifies the port range to scan. -p- scans every port.
2. --excludefile list.xml: The IP range given in the list.xml file is excluded from being scanned.
3. -Pn: This option treats all hosts as if they were online.
4. -n: This option forbids DNS resolution, while -R always enables it.
5. -sS/sA/sN/sF/s0: These options specify, which scan technique to use: TCP SYN, TCP ACK, TCP Null, TCP FIN scans the IP protocol scan.
6. -max-retries [tries]: How many outgoing probes are used per port is set here.
7. --host-timeout [time]: After the specified amount of time the host is treated as timed out.
8. --max-rate [number]: The maximal transmission rate of packets is set here.
9. -oN/oX/oG [file]: The output of the scan is stored in either a normal txt, xml or Grepable format of the name specified in file.
10. [X.X.X.X/X]: This sets the IP range(s) for scanning.

We mostly used the following ZMap's commands in the thesis:

1. -p [port]: This is the specified port to scan.
2. -o [name.xml]: This writes the results to the name.xml file. Other output options are possible.
3. -b [path/name.xml]: This enables ZMap to use a blacklist file name.xml.

4. -t [time]: This caps the time in seconds for sending packets.
5. -B [max. bits/seconds]: This sets maximum usable bandwidth on scanner side in bits per second.
6. -c [time]: This specifies how long to wait for an response in seconds.
7. -P [number]: This sets the number of probes to send to each IP address.
8. --config=[path/name.conf]: This enables ZMap to work with a predefined, repeatable conf file (name.conf), which specifies the scan parameters.
9. [X.X.X.X/X]: This simply specifies the IP range, which is to scan. Multiple IP ranges can be appended by putting a space between them.

Concerning Masscan, we used these commands rather frequently:

1. -p[port(s)]: This assigns the ports the scan should test, ranging from 0 to 65535.
2. -oX [file.xml]: This saves the output of the scan in the file.xml file. Other data formats such as csv or txt are also permitted.
3. --excludefile [file.txt]: This excludes every IP range written in the file.txt file. Other data formats are also permitted.
4. --max-rate [number]: This sets the maximum packet transmission rate on scanner side.
5. -c [scan.conf]: With this option a scan with the parameters set in the scan.conf file is performed.
6. --banners: Banner grabbing is enabled with this command.
7. --script=poodle: This enabled Masscan to check if a host is vulnerable to SSL 3.0 fallback.
8. [X.X.X.X/X]: This specifies the interesting IP range. Appending multiple IP ranges is possible by putting a space between them.

Bibliography

[1] LRZ. (2014) Überblick über das Münchner Wissenschaftsnetz (MWN). [Last accessed 28th December 2014]. [Online]. Available: http://www.lrz.de/services/netz/mwn-ueberblick/

[2] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast Internet-wide Scanning and its Security Applications," in *USENIX Security*. Citeseer, 2013, pp. 605–620.

[3] C. Meyer and J. Schwenk, "Lessons Learned From Previous SSL/TLS Attacks-A Brief Chronology Of Attacks And Weaknesses," *IACR Cryptology ePrint Archive*, vol. 2013, p. 49, 2013.

[4] G. F. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, 2011.

[5] G. Conti and K. Abdullah, "Passive visual fingerprinting of network attack tools," in *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*. ACM, 2004, pp. 45–54.

[6] Z. Durumeric, M. Bailey, and J. A. Halderman, "An Internet-wide view of Internet-wide scanning," in *USENIX Security Symposium*, 2014.

[7] R. D. Graham and P. C. Johnson, "Finite State Machine Parsing for Internet Protocols: Faster Than You Think," in *Security and Privacy Workshops (SPW), 2014 IEEE*. IEEE, 2014, pp. 185–190.

[8] E. Seagren, *Secure your network for free: using NMAP, Wireshark, Snort, Nessus, and MRTG*. Syngress, 2007.

[9] G. Lyon. (2012) The official nmap project guide to network discovery and security scanning. [Last accessed 12nd February 2015]. [Online]. Available: www.nmap.org

[10] LRZ. (2014) Flyer über das LRZ. [Last accessed 30th December 2014]. [Online]. Available: http://www.lrz.de/wir/lrz-flyer/lrz-flyer.pdf

[11] B. Möller, T. Duong, and K. Kotowicz, "This POODLE Bites: Exploiting The SSL 3.0 Fallback." Google, 2014.

[12] L. I8. (2014) Informatik VIII: Lehrstuhl für Netzarchitekturen und Netzdienste. [Last accessed 10th February 2015]. [Online]. Available: http://www.net.in.tum.de/de/startseite/

[13] O. Tarabai, "A penetration testing framework for the Munich Scientific Network," 2014.

[14] F. von Eye, W. Hommel, and S. Metzger, "Dr. Portscan: Ein Werkzeug für die automatisierte Portscan-Auswertung in komplexen Netzinfrastrukturen," in *Sicherheit in vernetzten Systemen: 20. DFN Workshop*. BoD–Books on Demand, 2013.

[15] D. Adrian, Z. Durumeric, G. Singh, and J. A. Halderman, "Zippier ZMap: Internet-wide scanning at 10 Gbps," in *Proceedings of the 8th USENIX conference on Offensive Technologies*. USENIX Association, 2014.

[16] R. D. Graham. (2014) MASSCAN: Mass IP port scanner. [Last accessed 30th January 2015]. [Online]. Available: https://github.com/robertdavidgraham/masscan

[17] M. De Vivo, E. Carrasco, G. Isern, and G. O. de Vivo, "A review of port scanning techniques," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 2, pp. 41–48, 1999.

[18] M. Al-Tamimi, W. El-Hajj, and F. Aloul, "Framework for creating realistic port scanning benchmarks," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International*. IEEE, 2013, pp. 1114–1119.

[19] M. Anbar, A. Manasrah, S. Sureswaran Ramadass, A. Altaher, A. Aljammal, and A. Almomani, "Investigating Study on Network Scanning Techniques," *International Journal of Digital Content Technology and its Applications (JDCTA). Volume7 (9.37)*, p. 9, 2013.

[20] N. Hoque, M. Bhuyan, R. C. Baishya, D. Bhattacharyya, and J. Kalita, "Network attacks: Taxonomy, tools and systems," *Journal of Network and Computer Applications*, vol. 40, pp. 307–324, 2014.

[21] G. V. Bard, "A Challenging but Feasible Blockwise-Adaptive Chosen-Plaintext Attack on SSL," in *SECRYPT*, 2006, pp. 99–109.

[22] T. Duong and J. Rizzo. (2011) Here come the XOR ninjas. [Last accessed 11th January 2015]. [Online]. Available: http://www.hpcc.ecs.soton.ac.uk/~dan/talks/bullrun/Beast.pdf

[23] Johnson, "CRIME attack compression ratio info-leak made easy," *Cryptography II*, 2013.

[24] H. Gluck and A. Prado, "BREACH: Reviving the crime attack," *Online at http://breachattack.com/resources/BREACH - SSL, gone in 30 seconds.pdf*, 2013.

[25] Z. Durumeric and A. Kasten, "The matter of Heartbleed," in *Proceedings of the 2014 Conference on Internet Measurement Conference*. ACM, 2014, pp. 475–488.

[26] J. A. Kupsch and B. P. Miller, "Why do software assurance tools have problems finding bugs like heartbleed?" *Continuous Software Assurance Marketplace*, vol. 22, 2014.

[27] A. Langley. (2014) POODLE attacks on SSLv3. [Last accessed 11th February 2015]. [Online]. Available: https://www.imperialviolet.org/2014/10/14/poodle.html

[28] ——. (2014) The POODLE bites again. [Last accessed 11 February 2015]. [Online]. Available: https://www.imperialviolet.org/2014/12/08/poodleagain.html

[29] W. Hommel and H. Reiser, *Das Münchner Wissenschaftsnetz (MWN) Konzepte, Dienste, Infrastrukturen, Management*. LRZ, 2012.

[30] A. Bley and M. Pattloch, *Modellierung und Optimierung der X-WiN-Plattform*. Konrad-Zuse-Zentrum für Informationstechnik, 2005.

[31] J. Pattloch. (2014) Das Wissenschaftsnetz X-WiN. [Last accessed 5th February 2015]. [Online]. Available: https://www.dfn.de/xwin/

[32] K. Sklower, "A tree-based packet routing table for Berkeley unix," in *USENIX Winter*, vol. 1991, 1991, pp. 93–99.

[33] G. R. Wright and W. R. Stevens, *TCP/IP Illustrated*. Addison-Wesley Professional, 1995, vol. 2.

[34] ntop Blog. (2014) Introducing PF_RING ZC (Zero Copy). [Last accessed 30th December 2014]. [Online]. Available: http://www.ntop.org/pf_ring/introducing-pf_ring-zc-zero-copy/

[35] W.-c. Feng, "The case for TCP/IP puzzles," in *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 4. ACM, 2003, pp. 322–327.

[36] N. El-Nazeer and K. Daimi, "Evaluation of Network Port Scanning Tools," in *The 2011 International Conference on Security and Management (SAM'11), July*. Citeseer, 2011, pp. 18–21.

[37] OffensivePython. (2015) Nscan: Fast internet-wide scanner. [Last accessed 10th February 2015]. [Online]. Available: https://github.com/OffensivePython/Nscan

[38] C. Meyer, "20 Years of SSL/TLS Research An Analysis of the Internet's Security Foundation," *Ruhr-University Bochum*, 2014.

[39] Z. Jeelani, "An insight of SSL security attacks," *International Journal of Research in Engineering and Applied Sciences*, 2013.

[40] M. McDonnell, "Terminal Multiplexer," in *tmux Taster*. Springer, 2014, pp. 1–18.

[41] L. Chappell, *Wireshark 101-Einführung in die Protokollanalyse*. MITP, 2013.

[42] B. Ward, *How Linux Works*. No Starch Press, 2014.

[43] R. Blum, *Linux command line and shell scripting bible*. John Wiley & Sons, 2008, vol. 481.

[44] Y. Rekhter, D. Karrenberg, G. d. Groot, and B. Moskowitz, "Address allocation for private internets," *RFC 1597*, 1994.

[45] P. Chandra, M. Messier, and J. Viega, "Network security with OpenSSL," *O'Reily, June*, 2002.

[46] M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire, "Internet assigned numbers authority (IANA) procedures for the management of the service name and transport protocol port number registry," *Work in progress*, 2011.