Informatics

Suchen von IoT Geräten mittels einer Mobilen Android Applikation

Erkennen von Geräten in IP-basierten Netzwerken und über Bluetooth

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Michael Bernd Stöger, BSc
Matrikelnummer 11778261

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Privatdoz. Mag.rer.soc.oec. Dipl.-Ing. Dr.techn. Edgar Weippl
Mitwirkung: Dipl.-Ing. Christian Kudera
Univ.Lektor Dipl.-Ing. Dr.techn. Georg Merzdovnik
Univ.Lektor Dipl.-Ing. Michael Pucher

Wien, 20. Februar 2023

Michael Bernd Stöger   Edgar Weippl

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

TU WIEN Informatics

Scanning for IoT Devices Using a Mobile Android Application

Detecting Devices in IP-Based Networks and the Bluetooth Environment

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Michael Bernd Stöger, BSc

Registration Number 11778261

to the Faculty of Informatics

at the TU Wien

Advisor: Privatdoz. Mag.rer.soc.oec. Dipl.-Ing. Dr.techn. Edgar Weippl
Assistance: Dipl.-Ing. Christian Kudera
Univ.Lektor Dipl.-Ing. Dr.techn. Georg Merzdovnik
Univ.Lektor Dipl.-Ing. Michael Pucher

Vienna, 20th February, 2023

Michael Bernd Stöger   Edgar Weippl

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Michael Bernd Stöger, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 20. Februar 2023

Michael Bernd Stöger

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

Special thanks to Christian Kudera, Georg Merzdovnik and Michael Pucher for their advice throughout the writing of this thesis and for lending me IoT devices to test my work on. Also, thank you to my friends and family for supporting me morally for the last few months. Lastly, I would also like to thank TU Wien, whose resources and services have been very helpful during research for my work.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

The number of IoT devices has been growing in the last few years. Many devices that usually had no connectivity whatsoever, are now internet connected. With this high number of devices, it is possible to lose track of which devices are connected to a network. In this work, approaches used to locate and classify devices in the local environment using an unmodified Android device are evaluated. The developed procedures remove entry barriers like expensive equipment or complicated setups for scanning, as only an installed Android application is needed. To increase the number of recognized devices, methods to create device fingerprints from offline Linux firmware images have been researched and implemented. This allows devices to be recognized, which have never been in our hands while developing this application. But not only devices connected to an IP-based network are of concern for this work, but also Bluetooth devices can be recognized by the application. Using this feature allows the user to get an overview of surrounding portable devices, including modern speakers and smartwatches. One major finding of this work is that most active scanning methods found in the literature can be used on Android as well. Passive methods are often unsupported or restricted on this mobile platform. This does not have to negatively affect the results, as detailed information often has to be probed actively anyway.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Abstract **ix**

Contents **xi**

1 Introduction **1**

2 Background **3**
2.1 Computer Networks 3
2.2 Host Detection 11
2.3 Service Discovery in IP Networks 13
2.4 Device Scanning 18
2.5 Bluetooth 20

3 Related Work **25**
3.1 IP-based Detection 25
3.2 Bluetooth 28

4 Methodology **31**
4.1 Investigating Devices 31
4.2 Scanning Strategies for IP-Based Networks 46
4.3 Bluetooth Classification 50
4.4 Android Implementation 53

5 Results **69**
5.1 Found Properties - IP-based Devices 69
5.2 Fingerprints - Bluetooth Devices 73
5.3 Android Application 76
5.4 Firmware Image Analysis Tool 81

6 Discussion **83**
6.1 Device Classification and Recognized Devices 83
6.2 Fingerprinting Techniques 84
6.3 IP-based Scanning Approaches 86
6.4 Future Work 87

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

7 Conclusion **89**

List of Figures **91**

List of Tables **93**

Bibliography **95**

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 1

Introduction

In the last years, the total number of Internet of Things (IoT) devices increased to approximately 12.3 billion in 2021 [1]. These devices can be of various types: Cameras, Wearables, Gaming consoles, and even refrigerators or coffee machines might be internet-connected today. Considering the enormous amount of devices it can be easy to lose track of every piece of equipment connected to a network. A system administrator even might not know all the machines connected, as employees might have brought them from home for their personal use. Either way, to ensure network security, it is important to know what devices are running and if they have any security vulnerabilities. Finding them might be time-consuming for an experienced network administrator and undoable for the average home user. Network scanning software is designed for professional users and their output might not provide all the information needed [2]. Device-specific information like manufacturer, model and firmware version still needs to be collected manually. Gathering information manually, devices might be missed during the process. An automated approach can improve network security by automatically detecting and flagging potentially vulnerable devices. Vulnerable IoT devices might not only affect the security of the own network, but potentially also harm other internet-connected devices as well [3]. A low-effort method to supervise a network combined with a low entry barrier might greatly improve overall network security and can help to preserve privacy and prevent data loss or other damages to internet-connected devices.

The main contribution of this thesis is to analyze several approaches for detecting IoT devices in the nearby environment and applying them to an Android application. Therefore some techniques need adaption to work within the restricted Android ecosystem. The focus of this research lies on IP-based detection approaches, as the Bluetooth capabilities of Android and the built-in receivers are rather limited. To present the findings of this work an Android application has been written, that is capable of using the described approaches. It is built using the Android 11 API, as it was the most recent version of Android when research has been started for this work.

The remaining of this thesis is structured as follows: In chapter 2, the necessary background, along with terms and technologies, will be described. Chapter 3 contains related research, including the state of the art. Scientific work, essential for the methods used in my work, is presented there. The methods found during the literature review are put to use in chapter 4, where it is explained how the results of this work are achieved. The actual results can be found in chapter 5, where they will be discussed and outlined. Afterwards, these results will be finally discussed in chapter 6, including a look at possible future work and current limitations. At the end, in chapter 7, the whole work will be summarized and concluded.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 2

Background

This chapter is about the concepts and technologies used in this work. Also, the theoretical background that is needed to understand the content of this thesis, will be explained.

2.1 Computer Networks

Computer networks are designed to provide connectivity among a set of computers [4]. A major part of this thesis is concerned with IP networks. Here it will be explained how networking works and how scanning tools get their information. This thesis only uses IPv4, therefore only topics regarding IPv4 will be discussed here.

For this work, we were mostly concerned about how network protocols work and how technologies work together to provide useful functionality. Designing and building a network is not of interest for this thesis.

2.1.1 OSI Network Layer Model

The OSI layer model categorizes networking functionality into seven layers [5]. OSI stands for "Open Systems Interconnection". The layer model defines how its seven layers interoperate to transmit data from one person to another. A visual representation of this layer model is shown in figure 2.1. Following the purpose of the layers will be explained including the relevance for this thesis.

Layer 1 The first layer of the OSI model is called the physical layer [7]. It represents the physical medium that the data is going through. This can be e.g. copper, fiber, or wireless technology.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

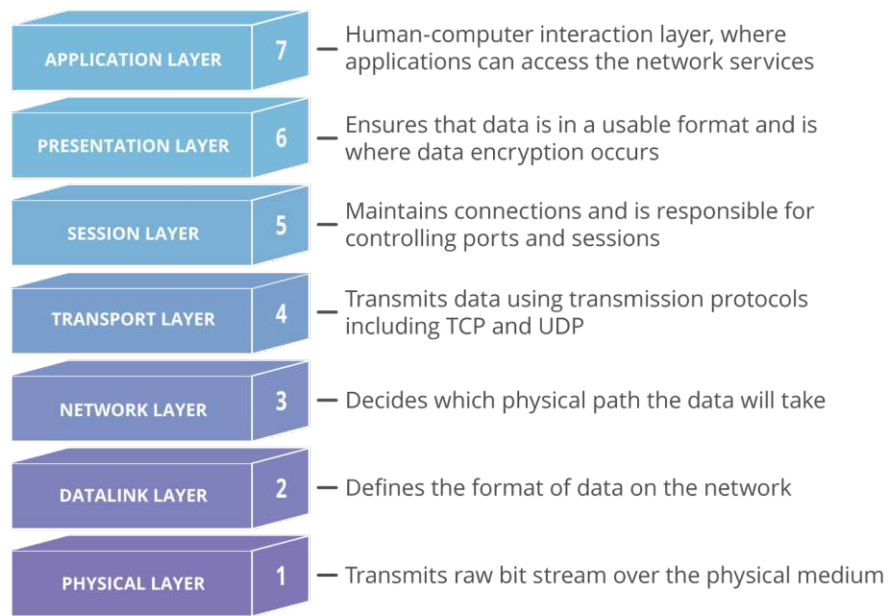


Figure 2.1: OSI 7 Layer Model [6].

Layer 2 On the second level, the data link layer is used to transfer data between two neighboring network entities [7]. Here it is checked that no transmission errors have occurred in the layer below [5]. Also, the MAC address, described in more detail in section 2.1.2, is introduced here and used to address the direct neighbors in the network.

Layer 3 Routing of network packets is introduced in the third layer, the so-called network layer [7]. Using layer three technologies it is possible, that a network packet traverses multiple networks before reaching its destination. For routing, the IP address, as explained in section 2.1.3, is being used.

Layer 4 The transport layer is responsible for delivering complete messages from one endpoint to the other [5]. Too long messages will be split up into multiple chunks - called segments - and reassembled on the other side of the connection [6]. Two important protocols for this thesis reside in this layer: TCP and UDP [8]. Details about these protocols follow in section 2.1.4.

Layer 5 As the name suggests, the session layer is being used to handle sessions [5][6]. This does not only include establishing, maintaining, and terminating them, but also creating checkpoints for e.g. file transfers. That way it is not necessary to retransfer the whole data when the session gets interrupted during transfer.

Layer 6 The presentation or translation layer transforms data from and to the application layer (layer 7) [5]. These transformations can be translations (e.g. between different

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

encodings), en-/decryption, or compression.

Layer 7 On top of the OSI layer model, the application layer can be found [5]. It is responsible for collecting data from the user and presenting received information to the user [6].

2.1.2 MAC Addresses

A Media Access Control (MAC) address is a 48-bit long value [9]. As mentioned before it is used on layer two of the OSI model. For human readability, it is usually denoted as a hexadecimal value like "aa-bb-cc-dd-ee-ff". The structure can be seen in figure 2.2.

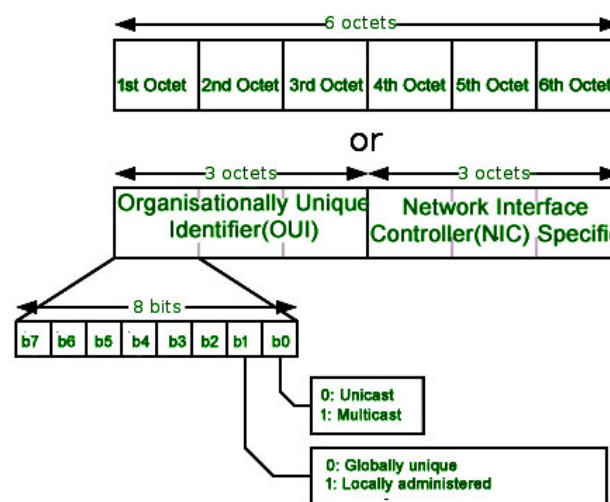


Figure 2.2: MAC Structure [10].

Every network interface card in a computer has a globally unique MAC address [10]. It is set permanently during the manufacturing process of the network interface card (NIC) and can therefore also be called the physical address. Every NIC has its own physical address, which means a computer can have multiple MAC addresses [11]. Nevertheless, this address can be overridden by software. As denoted in figure 2.2, the type of the MAC address can be seen in its first octet.

For this thesis, we mostly care about globally unique unicast MAC addresses. Looking at the "organizational unique identifier" (OUI) part of the MAC address, it can be deduced from which vendor the device in question was built. A list of OUIs and the affiliated vendors is publicly available at [12]. Following, a few examples are given from this list:

- 60-8B-0E: Apple

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- 6C-5E-3B: Cisco
- 70-DA-17: Austrian Audio

It is possible that one organization has been assigned multiple OUIs. E.g. "80-C5-E6" and "48-86-E8" are (among others) assigned to Microsoft [12].

2.1.3 IP Addresses and Subnets

IP addresses are another important aspect of modern computer networks. As mentioned before, only IPv4 will be covered in detail for this thesis, as IPv6 is generally not used here.

Internet Protocol (IP) addresses are 32-bit values, which are unique within an IP network [13]. For easier readability, the address is usually split into four parts - four times eight bit - each ranging from 0 to 255. A valid IP address would be e.g. 192.168.0.1. That means theoretically 2^{32} possible valid addresses, from 0.0.0.0 to 255.255.255.255 [14]. Practically, this number of possible network devices is too big for any network. Therefore, a mechanism called subnetting is being used, to split up this huge address space into smaller subnets.

An IP address can be split into two parts, the network id and the host id [14]. Using the subnet mask, it can be determined how many bits belong to the host id and the network id respectively [13]. As the IP address, also the subnet mask is 32-bit long. It has two common denotations. The first one is similar to the IP address, with four values between 0 and 255. An example of an IP address with a subnet mask, translated to their respective bit patterns, can be seen in figure 2.1.

```
1  11000000.10101000.01111011.10000100−IP address
      (192.168.123.132)
2  11111111.11111111.11111111.00000000−Subnet mask (255.255.255.0)
```

Listing 2.1: Sample IP Address and Subnet Mask [13].

The subnet mask masks the bits from the original address that belong to the network id. All remaining bits belong to the host id. The result can be seen in listing 2.2.

```
1  11000000.10101000.01111011.00000000 −
2      Network address (192.168.123.0)
3  00000000.00000000.00000000.10000100 −
4      Host address (000.000.000.132)
```

Listing 2.2: Result of masking with subnet mask [13].

Another way of denoting a subnet is the CIDR (Classless Inter-Domain Routing) notation [15]. In this notation, the length of the network id gets specified by the number of

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

bits written after an IP address, e.g. 192.168.123.132/24 for the example above.

The number of possible hosts in a (sub-)network is determined by the $2^n - 2$, with n being the number of bits used for the host id [13]. Not all possible addresses can be used for hosts because the first address (host id all zeros) is reserved for the network id. The last address (host id all ones) is used for broadcast.

Private and Reserved Addresses

Before 1993 IP addresses were assigned to classes, as can be seen in figure 2.3 [16]. These classes are not technically relevant anymore, but useful for understanding which IP addresses can be assigned to hosts and which not. Following the address ranges of the different classes [17]:

- Class A: 0.0.0.0 -> 127.255.255.255
- Class B: 128.0.0.0 -> 191.255.255.255
- Class C: 192.0.0.0 -> 223.255.255.255
- Class D: 224.0.0.0 -> 239.255.255.255
- Class E: 240.0.0.0 -> 255.255.255.255

With every class, the remaining address space is cut in half.

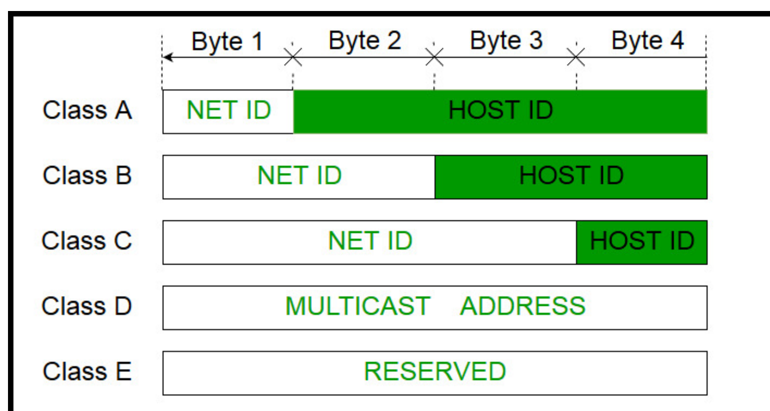


Figure 2.3: Network Classes [18].

Not all IP addresses can be assigned to internet-connected devices [19]. Class D and E addresses are reserved for special purposes [18]. Also the remaining address space can not be freely assigned. Especially noteworthy are the loopback addresses [20] (127.0.0.0 -> 127.255.255.255) and the private address ranges [19]:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- 10.0.0.0 -> 10.255.255.255
- 172.16.0.0 -> 172.31.255.255
- 192.168.0.0 -> 192.168.255.255

Private addresses are not routed and will not be assigned by IANA [19]. That means they are not globally unique, but only valid in the local network. During our research for this thesis, the above-mentioned private address spaces are the ones we usually encountered during our testing.

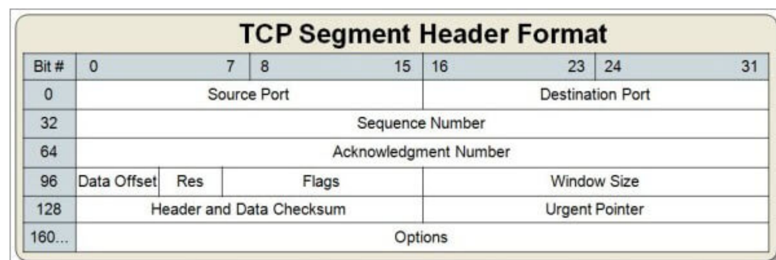
2.1.4 TCP and UDP

Both protocols, TCP and UDP, are located on OSI layer four and provide end-to-end transmission of data [8]. There are other protocols that work on layer four, but these two are the most important ones for this thesis.

TCP

The Transmission Control Protocol (TCP) is used to transfer data reliably and in order between two endpoints [21]. That means the sender gets a confirmation, that the data he sent, has been received on the other side. TCP also automatically handles retransmissions on errors and makes sure that fragmented data gets pieced together in the correct order on the receiver side. Additionally TCP is a connection-oriented protocol. That means that a connection has to be established before data can be sent.

In order to control TCP's features and transfer needed metadata the TCP protocol header is used [21]. How the TCP header is structured can be seen in figure 2.4. Following, an explanation of the important fields for this thesis.



TCP Segment Header Format

Bit #	0 … 7	8 … 15	16 … 23	24 … 31
0	Source Port		Destination Port	
32	Sequence Number			
64	Acknowledgment Number			
96	Data Offset, Res	Flags	Window Size	
128	Header and Data Checksum		Urgent Pointer	
160...	Options			

Figure 2.4: TCP Header [22].

Source/Destination Port To handle multiple connections on the same host, ports have been introduced [21]. These allow multiple processes to simultaneously transmit data across a network. On both sides, a TCP connection has an assigned port number. This number is 16 bits long and therefore its range is 0 - 65536.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Flags/Control Bits Flags are used to indicate and control connection states or transfer additional information [23]. Following, a selection of possible flags and their meaning:

- SYN: Is set on the first packets to initialize a connection. Connection initialization will be explained below.
- ACK: An acknowledgment flag is sent to indicate to the sender of a packet, that the data has been successfully received.
- FIN: Transmission of data is finished. No more data will be sent.
- RST: Connection should be reset. This is usually sent if a packet with unexpected data has been received.

As stated before, when using TCP a connection has to be established prior to data transfer. To do so, the three-way-handshake, as seen in figure 2.5, is used.

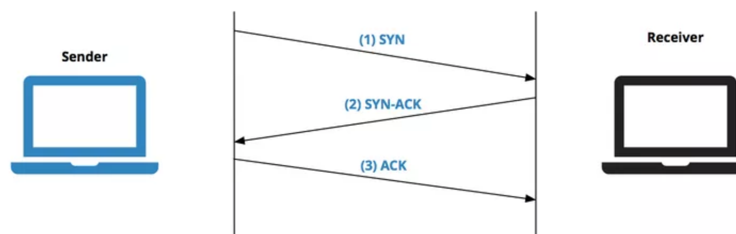


Figure 2.5: TCP Three-Way-Handshake [23].

To open a connection, the initiating side sends a TCP message, with the SYN flag enabled, to the desired communication partner [21]. When a service is listening on the target host, it will reply with a packet with the SYN and ACK flags set. It is possible to split this packet into two (one with ACK and one with SYN), but usually, this is done in one step. The initiator responds with an ACK message one more time. After this procedure, the connection has been successfully established. If the port is closed on the receiver system, it will respond with an RST instead of a SYN message.

When a communication partner has sent all data and the connection is no longer needed, the closing of the connection can be requested [21]. To do so a packet with the FIN option is sent. This will be acknowledged (ACK) by the communication partner. The connection will remain in a half-closed state until the second partner also sends the FIN message to close the channel. After the final acknowledgment, the connection will be deleted on both sides. As can be seen in figure 2.6, the procedure is similar to the initiation sequence.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

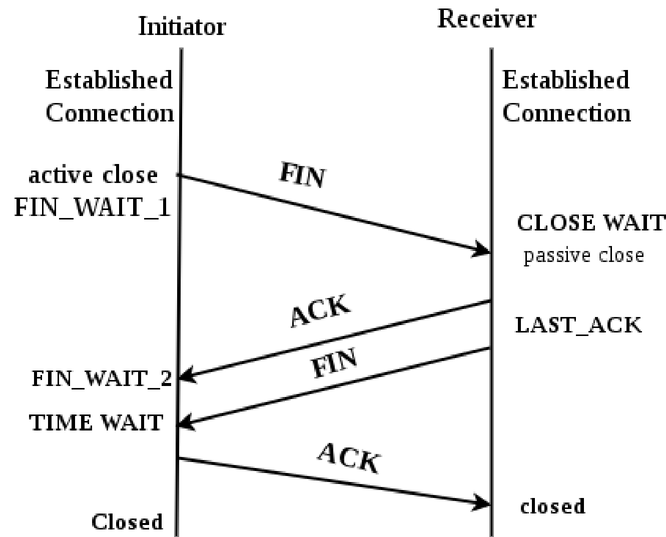
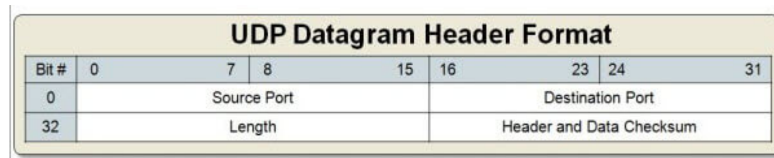


Figure 2.6: TCP Finishing Connection [24].

UDP

TCP provides a lot of features for reliable communication, but these are not always needed. When speed is prioritized over reliability for data transfer the User Datagram Protocol (UDP) can be the better choice [25]. Also, the header, as seen in figure 2.7, is much simpler than TCP's.



UDP Datagram Header Format

Bit #	0 – 15	16 – 31
0	Source Port	Destination Port
32	Length	Header and Data Checksum

Figure 2.7: UDP Header [22].

Source and Destination ports have basically the same meaning as for TCP, but the source port is optional here [25]. If no answer is expected, this port can be set to zero. Also, the checksum can be set to zero, if the sender decides not to calculate it.

Unlike TCP no connection has to be established for UDP data transfer [25]. Data is sent to the target with no confirmation of retrieval or error correctness. Since answers are not always sent, it is difficult to know whether a service is listening to a specific port or not. This problem will be discussed in greater detail in section 2.4.1.

2.1.5 ICMP

The Internet Control Message Protocol (ICMP) is a protocol used to transport error messages back to the source of a datagram [26]. It is an important part of every IP

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

module and therefore mandatory to implement. Nevertheless, it does not make IP reliable as there is no guarantee that an error message is sent.

ICMP messages are sent using IP datagrams, the content is stored directly after the IP header [26]. The first byte determines the ICMP type. Subsequent information needs to be interpreted according to this value. Following, a selection of ICMP messages that are important for this thesis.

Echo and Echo Reply When an echo message is received, the echo reply message is constructed by swapping the source and destination address [26]. The type field is set to 8 for echo and to 0 for echo reply. Any received data has to be sent back to the source. Informally this mechanism is often called ping and is used to check the reachability of a remote device. Ping is also the name of a standard tool in many operating systems, that sends ICMP echo messages.

Destination Unreachable The "Destination Unreachable" message is indicated by the value 3 in the type field [26]. It can be caused by various reasons, and every reason has its own code. This code is stored in the byte directly next to the type value. Following possible error codes and their meaning [26]:

- 0 - net unreachable: Sent by the gateway to indicate that the target network can not be reached.
- 1 - host unreachable: Sent by the target network gateway to indicate that the target host can not be reached.
- 2 - protocol unreachable: Sent by the target host to indicate that it does not speak the desired protocol.
- 3 - port unreachable: Sent by the target host to indicate that there is no service listening on the specified port.
- 4 - fragmentation needed: Sent by a gateway to indicate that the packet is too big for its MTU, but the do not fragment flag is set.

As stated before there is no guarantee that ICMP messages will be sent when an error occurs [27]. It is possible that the host or gateway in question does not generate these messages. Also, firewalls in the network path can block ICMP messages from reaching their destination.

2.2 Host Detection

Before any classification can occur, hosts have to be found first. For this thesis, the detection techniques need to work on the local network, without any firewall intercepting

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

packets. This does not implicitly limit their usefulness across the internet, Nevertheless, internet scanning was no concern for this work. The methods presented below have been already available in the program Nmap [28] at the time of writing.

Before any hosts can be probed, the search space has to be defined. In this thesis, the local network is scanned. How to determine the search space in the local network can be read in section 2.1.3.

ICMP Ping Scan When sending an ICMP echo message to a remote host, an ICMP echo reply message is expected in return. Detail about ICMP echo messages can be found in section 2.1.5. Probing remote hosts this way is not always successful [29]. Despite this behavior not being in line with internet standards, ping requests are often ignored or blocked across the internet. Nevertheless, in the local network, it can be expected that all active hosts are answering correctly to those requests.

```
[michael@GamingOnLinux ~]$ ping 10.0.0.1 -c 1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.429 ms

--- 10.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.429/0.429/0.429/0.000 ms
```

Figure 2.8: Pinging host in local network on Linux.

A ping scan can be conducted using the system tool ping, which is included by default on Linux, Windows, and macOS. An example of the output of this tool can be found in figure 2.8.

TCP SYN Ping Another way to probe for an alive host, is to try to establish a TCP connection to it [29]. As can be seen in section 2.1.4, a TCP connection is started by sending a packet with the SYN flag set. If the target port is closed, the host under investigation will send a packet with the RST flag set to indicate an error. Both, a normal ACK/SYN and RST answer will reveal an alive host.

After an ACK/SYN answer, Nmap will send an RST packet, to close the connection and not waste any resources on the target machine [29]. If the TCP handshake would be completed by sending another ACK packet, the connection could only be deleted by going through the entire connection termination sequence. This would take more time and consume more networking resources.

TCP ACK Ping Instead of trying to initiate a connection, directly an ACK packet is sent to the host under investigation [29]. If the other host is alive, it should respond with an RST packet, as the previously sent ACK packet was unexpected in this situation. This method has multiple advantages over the SYN scan:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- No connection is established, as an ACK packet is not the correct way to initialize a connection.
- Scanning is quicker, as only one packet has to be sent.
- Some stateless firewalls can be circumvented.

Modern stateful firewalls will block this scan, as an ACK packet without an established connection is invalid [29]. But in the local network, without any firewall between the hosts, this method can provide good results.

UDP Ping As explained in section 2.1.4, UDP has no connection concept. Still, sending a message to a closed port should result in an ICMP destination unreachable message [29], as also mentioned in 2.1.5. As the ICMP message is sent from the host under investigation, the host in question reveals its existence in the process.

It is essential for this technique to work to hit a UDP port that is closed, otherwise, the higher level protocol in use needs to be known, or no message will be sent in return [29]. As with the methods described before, firewalls or hosts that do not comply with the RFC standards might lead to incomplete results.

2.3 Service Discovery in IP Networks

For this thesis, finding live hosts is not enough. By using offered services, more information about the previously found hosts should be acquired. For user convenience, modern home networks get automatically configured using DHCP [30]. That means assigned IP addresses are usually unknown to a network user. Still, some devices might provide services interesting for a particular user. To automatically find these services, service discovery protocols are used. In this section, the two service discovery protocols used in this thesis will be described.

2.3.1 SSDP

The Simple Service Discovery Protocol (SSDP) is part of the UPnP specification [31]. It is used to advertise and discover services in the current network. There are many other functions and features covered by the UPnP specification, but this section is limited to SSDP and fetching the respective service definitions.

General Architecture

In figure 2.9, the different ways of discovering a service using SSDP can be seen. All UPnP devices listen (per default) on UDP port 1900 for multicast messages [31]. Clients are called control points, while servers are root devices.

Root devices use UDP multicast to advertise their services in intervals [31]. Control points can listen to these messages and compose a list of available services and devices. If

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

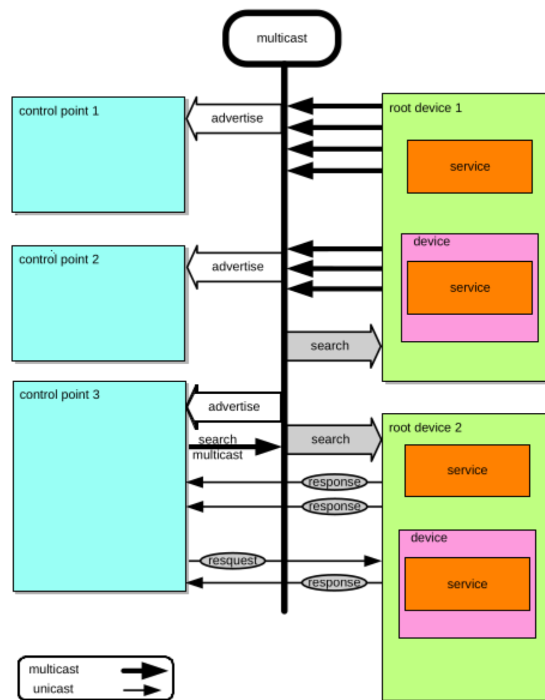


Figure 2.9: SSDP Service Discovery Architecture [31].

a control point does not want to wait, it can issue a search request to the same multicast port, as the advertisements are sent to. The search can be narrowed by adding search criteria. If a root device serves a matching service, it will send back a message using unicast directly to the searching control point.

Active Query for Services

Especially interesting for this work is the possibility to query devices and services on demand. In figure 2.10, the process from searching for a service to interacting with it can be seen. Here the process of finding active UPnP servers in the network will be explained.

```
1 M-SEARCH * HTTP/1.1\r\n
2 HOST:239.255.255.250:1900\r\n
3 ST:upnp:rootdevice\r\n
4 MX:2\r\n
5 MAN:"ssdp:discover"\r\n
6 \r\n
```

Listing 2.3: MSEARCH Message [32].

First, an M-SEARCH message is sent by the control point [32]. Since the servers are not known yet, this message is sent via UDP to the broadcast address 239.255.255.250 on

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

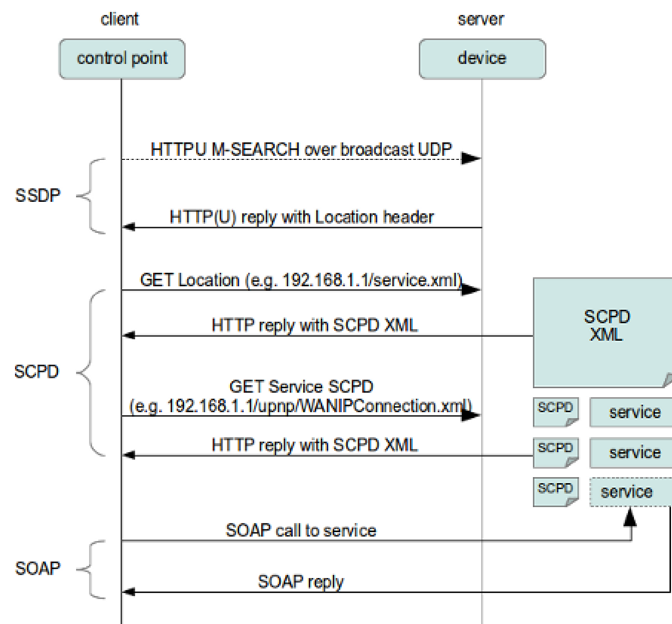


Figure 2.10: Full UPnP Search [32].

port 1900. This example is about finding root devices. The full search message can be seen in listing 2.3. The protocol in use is based on HTTP, but not fully compliant with the standard and furthermore transmitted over UDP [31].

Devices which receive the search message and match the search criteria should send a response directly to the host [31]. An arbitrary number of hosts can respond. An example from the response of a home router can be seen in listing 2.4.

```
1 HTTP/1.1 200 OK
2 LOCATION: http://10.0.0.1:49000/fboxdesc.xml
3 SERVER: FRITZ!Box 3490 UPnP/1.0 AVM FRITZ!Box 3490 140.07.30
4 CACHE-CONTROL: max-age=1800
5 EXT:
6 ST: upnp:rootdevice
7 USN: uuid:....::upnp:rootdevice
```

Listing 2.4: SSDP Response.

Among all the information transferred by the server, the location header is of the most interest for this work. It contains an URL to an extended description of the device. As it is a normal HTTP URL, the description can be downloaded with any HTTP client. A (shortened) description can be found in listing 2.5, taken from an AVM FRITZ!Box 3490 router.

```
1 <root>
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
2  ...
3  <device>
4  <deviceType>urn:schemas-upnp-org:device:fritzbox:1</deviceType>
5  <friendlyName>FRITZ!Box 3490</friendlyName>
6  <manufacturer>AVM Berlin</manufacturer>
7  <manufacturerURL>http://www.avm.de</manufacturerURL>
8  <modelDescription>FRITZ!Box 3490</modelDescription>
9  <modelName>FRITZ!Box 3490</modelName>
10 <modelNumber>avme</modelNumber>
11 <modelURL>http://www.avm.de</modelURL>
12 <UDN>uuid:...</UDN>
13  ...
14 </device>
15 </root>
```

Listing 2.5: Device Description.

Only the first four operations from figure 2.10 have been explained in this sample. Interacting with via UPnP advertised services is out of scope for this thesis and therefore will not be further explained in this chapter.

Interesting Description Fields

As can be seen in listing 2.5, a lot of identifying information can be transmitted via the XML description files. Following selected fields and their meaning from the UPnP specification documents [31]:

- **friendlyName:** Short name to display to the user. (Required)
- **modelDescription:** Long description of the device, for displaying to the user. (Recommended)
- **manufacturer:** The manufacturer's name. (Required)
- **modelName:** The model name. (Required)
- **modelNumber:** The model number. (Recommended)
- **manufacturerURL:** URL to the manufacturer's webpage. (Allowed)
- **modelURL:** URL to the model's webpage. (Allowed)

When comparing the specifications [31] to the values present in e.g. listing 2.5, it can be seen that not all vendors fill in these fields validly and completely. Therefore information received from these descriptions must be handled with caution.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.3.2 DNS-SD

DNS - Service Discovery is another way of finding services in a network [33]. It uses the DNS infrastructure to search for so-called SRV entries [34]. A search request e.g. might look like this [34]:

```
1  _ldap._tcp.example.com
```

It can be seen that this search is composed of a service name, a protocol, and the search domain. The complete SRV DNS entry stored by the DNS server contains more information to fully locate a service and for load balancing [34]:

```
1  _Service._Proto.Name TTL Class SRV Priority Weight Port Target
```

Following an explanation of the stored fields derived from the RFC document [34]:

- _Service: Name of the service. Always starts with "_".
- _Proto: Used protocol for this service. As with the service name it always starts with "_". There are only two possible options "TCP" for TCP-based protocols and "UDP" for everything else (even if UDP is not in use).
- TTL and Cache: Same meaning, as for normal DNS entries. Not of special interest for this work.
- Priority: A 16-bit unsigned number, indicating the priority of the hosts when multiple similar entries are specified. Lower numbers have higher priorities.
- Weight: Also a 16-bit unsigned number, used for load balancing. If the priority of two entries is the same, the one with the higher weight should be chosen more often, relative to its value.
- Port: Same as in section 2.1.4, this value indicates the port number for the transport protocol.
- Target: Host name of the server which provides the service in question.

The mechanism described in this section enables network devices to find services in specific networks, only knowing their name and transport protocol. This can reduce the need for port scanning to find services. Also, there is no guessing which service is running on a specific host and port, as it is already defined by the given service name.

mDNS

In the local network, a mechanism called multicast DNS (mDNS) can be used instead of classic DNS servers. It is a variant of DNS that only works locally and does not need any specified servers [35]. The top-level domain ".local" is reserved for local use, can not

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

be assigned globally, and has to be always resolved using mDNS. Other names can be resolved using mDNS when no conventional server is available.

To use multicast DNS, DNS requests have to be sent to the multicast address 224.0.0.251, using UDP with the port number 5353 [35]. Answers are usually also sent using multicast. With little modifications, already existing DNS resolver software can be used for multicast DNS. Nevertheless, the standard [35] describes some extra features for fully compatible clients. These extra features will not be explained here, as they are not important for this work.

2.4 Device Scanning

Finding ways to communicate with remote network devices is essential for this thesis. In this section it will be explained, how network communication might be used to gather more information about the devices connected to the local network.

2.4.1 Port Scans

In section 2.3, finding services using automatic service discovery mechanisms has been covered. This is not always possible, as a specific device needs to support the mentioned protocols. This section is about scanning the TCP and UDP port range on network devices, without any helping service discovery protocol.

Many techniques presented here have similarities to those in section 2.2. Most differences can be found in the type of responses that are expected and how they are interpreted. As before, these techniques are already implemented in the program Nmap [36].

TCP SYN Scan As outlined in section 2.1.4, the first part of the connection initialization sequence is a TCP packet with the SYN flag set. Here the destination port matters, as it has to be set to the port under investigation. Depending on the returned answer multiple conclusions are possible [36]:

- ACK/SYN Packet: The port is open and waiting for connections.
- RST Packet: No service is listening to this port, it is closed.
- No response or ICMP unreachable: A firewall might be dropping the packets. The Port has to be considered closed, as it is not possible to communicate with any service there.

No matter which answer is received, there will be no packet sent to complete the handshake, therefore no connection is established [37]. Since no full connection is established, there should be no logs on scanned servers about many incoming connections. Nevertheless, circumventing firewalls and manipulating logs are of no interest for this thesis.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

TCP Connect Scan This scanning methodology is similar to the previously described SYN scan, but instead of using RAW sockets to send SYN packets, the system API is used to connect to the target [36]. This makes scanning slower, as a full connection is established and the system API also needs to be used to get information on the connection status [38]. For an average system, the difference in traffic volume can often be neglected, as only open ports cause additional traffic. Also, this approach has a major advantage: No system privileges are needed for performing this scan.

UDP Scan UDP scanning significantly differs from TCP scanning [27]. No connection is being established, as UDP is a connectionless protocol (refer to section 2.1.4). That means, to get a response the used protocol has to be known by the scanning software. Nmap does have scanning samples for some ports built in. Otherwise, the sent packet will be empty. This leads to mostly different outcomes while scanning [27]:

- Answer received: The port is open. This is only possible when the used protocol is known.
- No answer: The port might be open, but since the protocol is not known there is no way to talk to the underlying service. This outcome is also possible when a firewall drops the packets.
- ICMP port unreachable (code 3): As described in section 2.1.5, this message is sent when the port is closed.
- Other ICMP messages: This port is blocked by a firewall.

Therefore, the information gained from scanning UDP ports is rather limited. This is especially true, when the protocol is unknown that should be running on the port under investigation, as this makes it impossible to distinguish between an open port and a blocked port.

2.4.2 Banner Grabbing

After services have been found, some of them can be used to get information about the remote system. Banner grabbing simply means receiving the banner data from running services [39]. This can be done without any login credentials.

Banner grabbing is achieved by connecting to the desired service using TCP and waiting for any data to arrive [39]. After a specific amount of time, the connection can be closed and the banner should then already have been received. Following a few examples of services sending banners are provided - together with procedures on how to capture them:

Telnet/SSH For Telnet and SSH, a newly established connection is enough to receive the banner. Here an example of an SSH banner grabbed using Netcat on Linux:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
1 # nc scanme.nmap.org 22
2 SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.13
```

Listing 2.6: Grabbing SSH banner with netcat.

It can be seen that many properties of the remote system are revealed, including the OpenSSH version and operating system. This information could potentially be used by an attacker, to narrow down possible exploits. In this thesis, it is used to create fingerprints of IoT devices.

Using Telnet, the banner might look different, as usually the content of the "/etc/issue" file is sent [40].

HTTP HTTP is a bit more complicated, as it does not automatically send data to connecting hosts. Here any data and a newline have to be sent to retrieve information:

```
1 # nc scanme.nmap.org 80
2 A
3 HTTP/1.1 400 Bad Request
4 Date: Fri, 02 Dec 2022 18:30:06 GMT
5 Server: Apache/2.4.7 (Ubuntu)
6 Content-Length: 306
7 Connection: close
8 Content-Type: text/html; charset=iso-8859-1
```

Listing 2.7: Grabbing HTTP banner with netcat.

Like before the current software version (in this case Apache) and the operating system are revealed. Nevertheless, this information can not be taken for granted, as it is possible to spoof/change these values.

Other services might also send banners upon new connections. Gained information will vary from service to service and is also dependent on the actual implementation installed on the remote device.

2.5 Bluetooth

Even though IP-based networks are the main focus of this thesis, also Bluetooth devices are of concern for us. Both, Bluetooth Classic and Bluetooth Low Energy (LE) will be covered here.

2.5.1 Bluetooth Classic

Since 1999 Bluetooth Classic has been adopted by many vendors in a various range of devices [41]. Up until now, many revisions have been published.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Device Detection

Similar to IP-based networks, devices have to be found first, before they can be investigated. But because of Bluetooth's security architecture, it should - per definition - only be possible to find devices that are in discoverable mode. Devices and software, that are capable of sniffing Bluetooth classic connections do exist [42][43], but these procedures are not usable with standard Bluetooth adapters as found in Android devices. In this section, it will be explained why it is not possible to listen to foreign communications with a standard Bluetooth adapter.

Bluetooth uses, as other communication standards as well, the 2.4GHz band [44]. But not only a single frequency is used. Bluetooth communication hops over 79 channels. The current frequency is determined by the master device's MAC address and its clock value. The clock is a 28-bit counter and wraps approximately every 23 hours. The frequency is changed 1600 times per second, making it difficult to listen to longer parts of communication. A visual representation of this mechanism can be seen in figure 2.11.

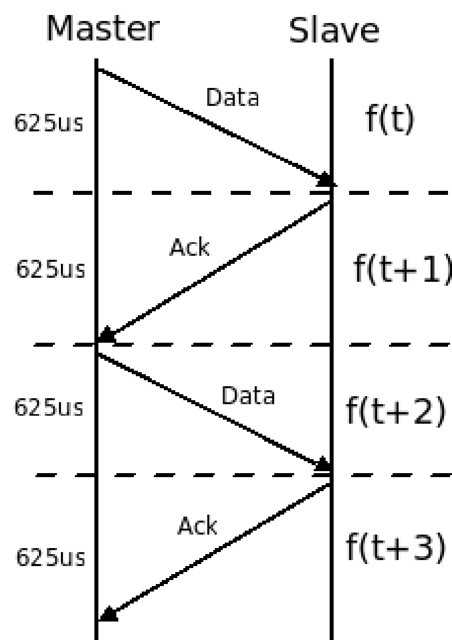


Figure 2.11: Frequency Hopping over Time [44].

Without knowing the master device's clock it is impossible to know the hopping pattern and therefore impossible to eavesdrop on a whole communication [44]. Additionally, Bluetooth packets are not transmitted in clear text, introducing additional barriers for listening to foreign communications. Standard Bluetooth adapters are not capable of monitoring all channels at once. To the best of our knowledge, at the time of writing, no attack is known enabling standard Bluetooth adapters to listen to foreign communications around.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Service Discovery

Bluetooth Classic uses the Service Discovery Protocol (SDP), to discover remote services [45]. For this thesis, the built-in Android Bluetooth and therefore SDP implementation is used. This section is about the data structures needed to exchange service descriptions between Bluetooth devices.

The SDP server holds a so-called service record for each registered service [45]. These can be requested and searched by a client. Each service record contains service attributes. One of these service attributes is the service record handle, a 32-bit number unique per SDP server. A visual representation of how the data is organized can be seen in figure 2.12.

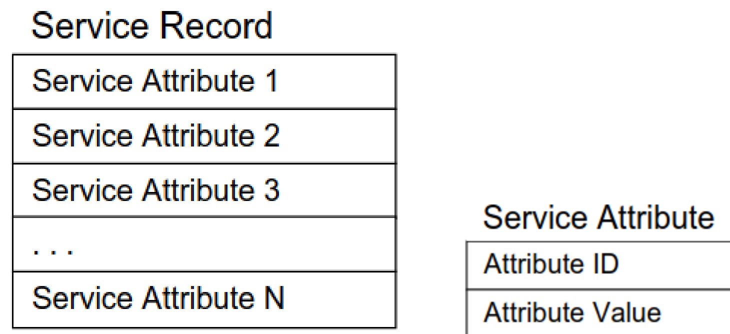


Figure 2.12: Stored Data for each Service [45].

There are many possible service attributes [45]. But especially interesting for this thesis is the so called "ServiceClassIDList". As the name suggests, it contains a list of service class IDs. These IDs are 16-bit UUIDs and indicate service types, that the advertised service implements.

Comparing UUIDs UUIDs should be unique across all space and time [45]. Usually, it is a 128-bit value, but as explained before there are shorter variants like the 16-bit above. Two UUIDs of the same length can be compared directly, otherwise, the shorter one has to be extended. From 16 to 32 bits the shorter value has to be zero-extended, to match the bit number. For conversions to 128 bit the formula is [45]:

```
1 128_bit_value = 16_bit_value * 296 + Bluetooth_Base_UUID
2 128_bit_value = 32_bit_value * 296 + Bluetooth_Base_UUID
3
4 Bluetooth_Base_UUID = 00000000-0000-1000-8000-00805F9B34FB
```

2.5.2 Bluetooth Low Energy

Bluetooth Low Energy (LE) is designed to consume far less power than its classic counterpart [41]. Since the specification has been released in 2010, Bluetooth LE has

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

been used in many devices including sports and medical equipment and automation systems.

Detecting devices for Bluetooth LE is less problematic than for Bluetooth Classic, as many devices always send their advertisement packages. Android provides API calls for finding nearby BLE devices and notifies the application when a device is found [46].

Generic Attribute Profile

The attribute protocol has been defined to store and read small amounts of data (little number of octets) [45]. Each attribute is identified by an UUID. These can be self-assigned or officially reserved as explained below in section 2.5.2. The protocol defines a client-server architecture, while a device might hold both roles at the same time. On the server-side attributes are stored, which can be read and written by a client.

GATT (Generic Attribute Profile) uses the attribute protocol underneath and introduces services and characteristics [47]. Services can contain a collection of characteristics, while those characteristics contain a single value and an arbitrary number of descriptors for the stored value [45]. Descriptors add additional information, such that a user can understand and use the stored value.

Assigned Numbers

As stated before, certain services and characteristics own officially assigned UUIDs [45]. This makes them recognizable across many different devices and allows for standardized interfaces. These assigned numbers can be searched for on the official Bluetooth document [48]. All assigned UUIDs are of 16-bit length. If a longer version is needed, conversion can be done as explained above.

For this thesis, especially the following UUIDs are of interest:

- 0x180A: Device Information Service
- 0x2A29: Manufacturer Name String
- 0x2A24: Model Number String
- 0x2A28: Software Revision String

The first UUID identifies a service itself, while the others point to characteristics located within this service.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 3

Related Work

This thesis uses and relies on concepts and ideas from other people's research. Also, there are many papers, that are related to this work, but their methodology is out of scope for our work. In this chapter the related research and state of the art will be presented and explained how it contributes to this work.

3.1 IP-based Detection

Classifying IoT devices in the local network is a nontrivial task, as they most of the time do not simply state their type over some service. This thesis is not the first one to deal with IoT classification. In this section, related work concerning IoT classification and remote device detection will be presented.

3.1.1 Active Discovery

Android devices are usually highly mobile and have limited runtime due to their battery capacity. Therefore active scanning methods are most suited for them, as they can be launched on demand and are usually finished quickly.

In the current literature, there are many approaches for classifying network devices. One has been presented by Sivanathan et al. [49]. In their work devices are classified based only on their open and closed TCP ports. Before any devices can be recognized by this method, they need to be scanned first. The whole TCP port range has to be scanned per device, resulting in a hierarchical structure of open and closed ports for the whole set of devices. When trying to classify a network device, this hierarchical structure is used to determine the device type using a much smaller number of port probes. In their testing, Sivanathan et al. have been able to distinguish 19 devices in their proposed way.

Kumar et al. [50] have to deal with similar problems in their work compared to this thesis. A major part of their paper is dedicated to ways of detecting devices in the local

TU Bibliothek, WIEN, Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

network. For finding devices, basically mDNS, UPnP and ICMP Ping are used. These technologies also play a major role in this thesis. Furthermore, device classification using other properties than mentioned is also discussed in their work. E.g. the vendor name is determined by looking up the first 24 bits of the MAC address in the IEEE OUI table. Data gathered from mDNS or UPnP is added to further gain information. Also, many security considerations are outlined in the paper, which are generally important in the emerging IoT ecosystem but are not a primary concern for this thesis.

Bajpai et al. [51] also search for IoT devices in IP-based networks. Many networks consist of combinations of IoT and non-IoT devices. To deal with this problem they have come up with an algorithm to separate these device categories. Using a combination of UPnP device descriptors, NetBIOS names and MAC-to-vendor mapping a specific device is marked as IoT or non-IoT device. If this does not lead to a result Bajpai et al. suggest using Nmap for OS identification and banner grabbing. In their work, these tasks have been done manually. For this thesis, many of the presented approaches have been adapted for automatic evaluation using an Android application, or for manual investigations of network devices.

Other IoT detection research has been done in the Cern network by Agarwal et al. [52]. During their research, they developed two tools, that are - to the best of my knowledge - not publicly available. Nevertheless, some of the internals are described in the paper. The first uses ICMP ping to check for live devices on the network. After an active host is detected, a reverse DNS lookup is carried out to retrieve the device's hostname. Filtering for IoT devices has to be done manually in the resulting list. The second tool is a Python program that uses Selenium with a web driver to render pages that need JavaScript. It is used to query the detected IoT devices for web pages. After the page has been fully rendered, the tool extracts various properties from the page's source code.

A related thesis has been published by Knabl [53] from JKU Linz. In his thesis, he generally tries to locate network devices and then tests them for security vulnerabilities. In an advanced view, detailed information about the detected devices is displayed. Many of these information-gathering technologies used by Knabl are also in use in my thesis. The main difference between Knabl's and my thesis is that his thesis is focused on the security aspects of IoT. Using his application, vulnerabilities should be detected automatically and displayed to the user for further investigation. My thesis is about recognizing the type and firmware of devices and providing a fine-grained classification of devices in the network.

3.1.2 Security

While not an imminent concern for this thesis, security is an important part of IoT. Manufacturers often neglect security to be able to push out new products faster into the market. These devices sometimes never get security updates or any other form of maintenance. Nevertheless, often the security research also contains concepts usable for device classification.

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

In 2020 Amro [54] conducted a survey about IoT vulnerability scanning. His work is mostly about internet scanning, which is not a concern for this thesis. But, in his work, he reveals many services that potentially run on a high number of IoT devices. Some of these services can potentially give away a device's identity and type. Amongst other tools, the search engine Shodan [55] is being used. Shodan periodically scans the internet and grabs protocol headers and banners. Some of the search results can be used as inspiration about what to look for on local devices.

Another IoT security paper has been published by Al-Alami et al. [56]. As before Shodan is being used to find vulnerable services and devices on the internet. Their search has been restricted to the public internet of Jordan, which resulted in over 40000 IoT devices under investigation. During their research, they found multiple vulnerable services resulting in many attackable devices. Also, some mitigations are proposed to close many of the found vulnerabilities and therefore make the devices more secure.

3.1.3 Traffic Collection and Machine Learning

A lot of research exists about collecting traffic. This collected traffic is then analyzed to find and classify devices around or in a particular network. This approach has various advantages. Purely passive listening does not leave any traces and does not interrupt ongoing operations. Also, it is possible to detect and classify devices that do not expose any services. Nevertheless, there are also some downsides. Passive devices, only waiting for requests, will not be detected by such methods. Also, depending on the activity of the devices, finding them all might take a long time.

Machine Learning can be used to identify network devices based on various properties. This technology makes it possible to add and detect devices without manual investigation. Nevertheless, new devices will not be detected until they have been added to the machine-learning model. Both mentioned methods - traffic collection and machine learning - did not make it into my work. Nevertheless, in this section, I will present related research, that contains interesting approaches to IoT classification, which uses these methods.

One approach using data collection has been researched by Guo and Heidemann [57]. They passively collect internet traffic at an open campus network and from an internet exchange. From the recorded connections they infer what devices might have caused them. This approach can only detect devices, which have been investigated first. In a controlled environment, IoT devices are monitored and their outgoing connections are recorded. If enough of these connections are also found on the data under investigation, it is assumed that an IoT device has been found. Guo and Heidemann's approach does not manipulate the data in any way, it is a completely passive way of detecting and classifying IoT devices.

Le et al. [58] used machine learning over collected traffic to identify IoT devices. In their research, they came to the conclusion that unsupervised learning leads to overfitting of the underlying decision model. Their results greatly improved from using expert knowledge and manually choosing properties for training and recognition. In their opinion, these

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

findings result from the great diversity of IoT devices, which makes it difficult to find meaningful criteria for device classification.

Another approach has been researched by Khandait et al. [59]. They use deep packet inspection (DPI) and investigate captured packets for previously determined keywords. If the IoT device sends unique keywords, this method works very accurately. Although, this method has some limitations. Using the described method, it is impossible to detect devices exclusively using encrypted communication. Also, devices from the same vendor or family often send the same keywords in their communication. This might lead to inaccurate results. Like in the research presented before, Khandait et al. state that IoT device detection and classification is difficult because of the wide variety of devices.

3.2 Bluetooth

This thesis also researches scanning the environment for nearby Bluetooth devices. But just detecting them is not sufficient. The target is to find their type and their firmware version. As with IP-based networks, this thesis is not the first work on research about this topic. A selection of related research concerning Bluetooth will be presented here.

3.2.1 Bluetooth Classic

Bluetooth connections differ from IP-based networks in many ways. Therefore fingerprinting and classification algorithms, as used to analyze IP-based networking devices, can not be used for Bluetooth. This section is about related work concerning Bluetooth Classic.

Herfurt and Mulliner [60] researched Bluetooth device fingerprinting based on their MAC addresses and advertised services. They use the service descriptors from service discovery and the first 24 bits of the MAC address to generate a fingerprint. With this fingerprint, it should be possible to distinguish devices and even different firmware versions of the same device type. Even though the research has been published in 2004, Herfurt and Mulliner's approach to fingerprint generation is the base for the fingerprinting algorithm in this thesis.

Listening to surrounding Bluetooth communications is a nontrivial task. Spill and Bittau [44] developed a method to sniff on foreign Bluetooth communications. Their method does not involve listening to all possible Bluetooth channels but provides a way of figuring out the needed parameters. This simplifies the needed hardware and makes an attack much more practical. Still, the hardware is specialized and - at the time of writing - it is not possible to replicate their approach using an off-the-shelf Android device.

The impact of Bluetooth security vulnerabilities is different compared to IP-based devices since they can only be exploited by an attacker nearby. Dunning [61] researched different threats to Bluetooth devices and connections. As mentioned before, attacking devices is not a target for this thesis, but Dunning's research also is useful for understanding the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

possible limitations of my work. Nevertheless, attacking modern devices might not only lead to them not functioning, but might also leak personal information that is stored on e.g. smartwatches.

3.2.2 Bluetooth Low Energy

Bluetooth Low Energy is, compared to the classic variant, designed to use less energy and therefore ensure longer battery life in situations where power is a highly limited resource. To achieve that goal, many mechanisms work differently in this mode of operation. Therefore, the methods and approaches often differ between the two Bluetooth variants. This section is about related research concerning the Low Energy variant of Bluetooth.

A major difference between Bluetooth Classic and Low Energy are the GATT profiles. While these can be implemented for Classic Bluetooth, they are mandatory in Low Energy. Celosia and Cunche [62] use these profiles in their work to fingerprint Bluetooth Low Energy devices. During fingerprint creation, it has to be taken into account that some fields are subject to change (e.g. heart rate or step counter in smartwatches). When no fingerprint is available, sometimes still valuable information can be displayed to a user. Celosia and Cunche describe fields in GATT profiles, that allow for identification of the device in question without any fingerprint created previously. This is especially useful for new devices that never have been in the hands of application developers. Also, it is impossible to fingerprint all possible devices, since the amount is enormous and new ones are released regularly.

In another paper, Celosia and Cunche [63] research Bluetooth Low Energy's privacy features. Continuing advertisements from peripheral devices could make users trackable. To prevent this, the Bluetooth standard includes privacy features that allow randomizing the device's MAC address. Celosia and Cunche's research shows that this randomization is not always correctly implemented. Also, the content of the advertisement packets often reveals a device's identity. Tracking devices is not part of my thesis. Nevertheless, it is interesting for future research, that the advertisement packets alone might contain identifying information.

Modern peripherals can expose a lot of personal data. Therefore it is important for many users to keep this data private. Barua et al. [64] published a survey on Bluetooth Low Energy security and privacy issues. While breaking or manipulating communication is of no interest to this thesis, getting information from nearby devices is. Like Celosia and Cunche, Barua et al. write - amongst other vulnerabilities - about openly available data using GATT profiles. Depending on the devices these profiles can reveal a lot of information about the device and its user.

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 4

Methodology

This chapter deals with the methods and procedures used to answer the research questions for this thesis. Also, the decision path to the finally applied methods will be described. This includes problems and failed solutions. The questions I am trying to answer, with the methods described in this section, are:

- Which properties can be used to recognize remote devices?

  This includes reasoning about if these properties are good indicators for a device's identity.

- Which approaches/algorithms can be used to classify remote devices?

  This thesis is limited to non machine-learning approaches. While machine learning might yield good results in some scenarios, it is out of scope for my work.

- What limitation poses the Android operating system to the found approaches?

  Due to security and privacy considerations, access to some of the system's functions is restricted. Also, different hardware and software stacks compared to traditional desktop systems pose additional limitations.

4.1 Investigating Devices

Before any device can be recognized remotely, properties have to be found first, from which the device can be identified. Finding these properties is a nontrivial and time-consuming task. The accuracy of device classification is directly dependent on the quality of the found properties. These properties should have the following characteristics:

- Unique

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- Observable
- Stable

Uniqueness is important because it ensures that there are fewer (or no) misclassifications. That does not necessarily mean that a single property needs to be unique, but the combination should be. Otherwise, it might be possible to mistake two devices that (partially) share their properties.

Especially for this thesis, observability is an important characteristic. Not all possible device properties are detectable remotely. Also scanning using an Android application poses additional restrictions to scanning capabilities. Detailed explanation about detectable properties can be found in section 4.4.3.

Some properties, e.g. the advertised name for Bluetooth devices, are user changeable. Since these are subject to change, they can not be considered stable. Stable properties have a constant value across different devices of the same type and during the lifetime of the device/firmware combination.

This section is about finding features that are usable and sensible to use for remote classification.

4.1.1 Manual Analysis

The first approach tried and used in this thesis, is the manual analysis. Therefore the device of interest is manually investigated using a PC and any necessary resources needed and available. This includes any software that helps during analysis. This section contains explanations of the steps that have been taken to analyze the devices used for testing in this thesis. Nevertheless, this section has no claim of being exhaustive about the possible ways of analyzing devices. The goal is to find enough good properties that allow for the creation of a unique fingerprint. For this section, it is assumed that it is possible to physically access the device and it is currently not needed for any production environment.

Finding Documentation

Before doing any technical analysis, it can be useful to find and read any available documentation. Following examples of what to look for:

- Supported Protocols (e.g. UPnP, mDNS,...)
- Way to display/find the IP address
- Available APIs

This list is not exhaustive but gives an idea of what to have an eye on when searching a device's documentation. The information found in this step might also be useful for the other steps below.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Locating Device

Before any tests can be conducted on a new device, it has to be found on the network first. For this part, I assume that the device is connected and has a valid IP address configured (e.g. by using DHCP). The target of this step is to acquire the IP address of the device under investigation. During the writing of this thesis, all tests have been conducted in a home network environment. This means there are no firewalls in between the devices, the amount of connects and disconnects to the network is very low and there is no VLAN separation between parts of the network.

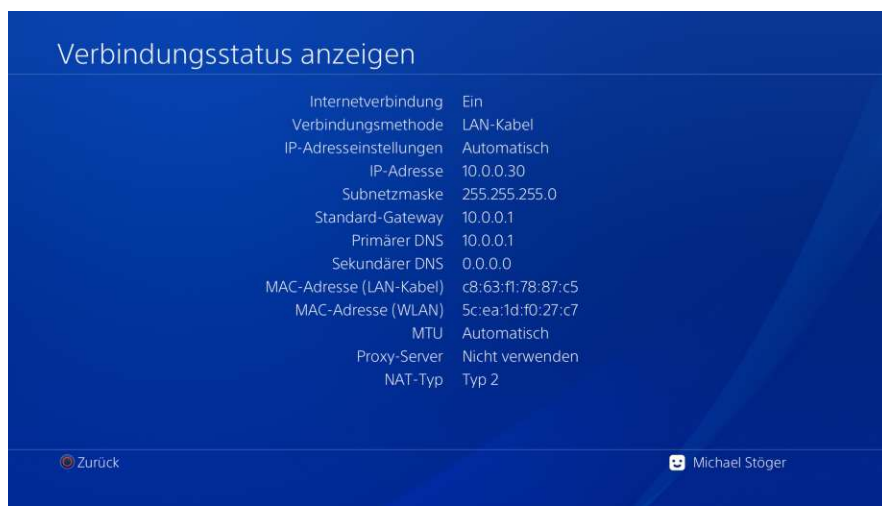


Figure 4.1: Network Information from a PS4.

Possible ways to discover the device's IP address are:

Device display: Some devices provide an option to display their current network settings. An example from a Playstation 4 can be seen in figure 4.1. Usually, the device manual contains instructions on how to access this function. This method is only possible if the device has some kind of graphical output.

Home router: Not all devices have a display built-in or can output to some external device. Still, the DHCP server included in modern home routers might be able to provide the wanted information. When the new device uses DHCP options [65], it is possible to provide a name to the DHCP server. Using this name it is often trivial to find the device in question using the router's web interface, as can be seen in figure 4.2.

Network scanning: If the above-mentioned approaches do not work, the device can be found using network scanning software. A possible approach using Nmap [28] will be described below.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Name	Verbindung	IP-Adresse	Eigenschaften
Diese FRITZ!Box			
fritz.box	DSL , deaktiviert	10.0.0.1	WLAN 2,4 / 5 GHz
Aktive Verbindungen			
PS4-7887C5	LAN 2 mit 1 Gbit/s	10.0.0.30	

Figure 4.2: Network Information from Home Router.

All tests are conducted on a machine running Arch Linux. At the time of writing the software is at the state of December 2022, the commands and results shown here might not be the same for other operating systems or software versions.

Before scanning the network, the search space has to be determined. This can be done by running the command from listing 4.1.

```
ip addr
```

Listing 4.1: Command to get information about the current network.

The output will be similar to the example in listing 4.2.

```
2: enp4s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq
    state UP group default qlen 1000
link/ether b4:2e:99:0e:3d:21 brd ff:ff:ff:ff:ff:ff
inet 192.168.0.10/24 brd 192.168.0.255 scope global dynamic
    noprefixroute enp4s0
```

Listing 4.2: Information about a connected network.

On the last line, the current IP address and the subnet mask in CIDR notation can be found (refer to 2.1.3 for details). Using Nmap no further calculations are needed. To find live hosts in the current network, the command from listing 4.3 can be used. This will start a ping sweep [29] in the local network and print the results to the console.

```
nmap -sP 192.168.0.10/24
```

Listing 4.3: Host discovery using Nmap.

The results contain the IP address of the live hosts, their MAC addresses and the vendor name. The vendor name makes it usually easier to determine the device in question. The final result might look similar to listing 4.4. To save space, unimportant lines have been cut out.

```
Nmap scan report for 192.168.0.1
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
2  Host is up (0.00028s latency).
3  MAC Address: 9C:3D:CF:1D:79:30 (Netgear)
4  ...
5  Nmap scan report for 192.168.0.102
6  Host is up (0.75s latency).
7  MAC Address: 30:CD:A7:17:9E:B0 (Samsung Electronics)
8  ...
9  Nmap scan report for 192.168.0.10
10 Host is up.
11 Nmap done: 256 IP addresses (6 hosts up) scanned in 5.25
       seconds
```

Listing 4.4: Host discovery result from Nmap.

If it is not obvious which device is the correct one, unplugging it and starting the scan again might quickly reveal the device in question. This step also reveals one important piece of information: The MAC address of the device, and therefore its vendor prefix. This alone is not a unique classifier for a device type, but often a good starting point for further analysis.

Port Scan

After the device has been located, gathering more information about the device in question is necessary to create a fingerprint. Port scanning is one way of finding running services on the device. The technological details have been explained in section 2.4.1. Here it will be presented which tools to use and how to interpret the results. Both, TCP and UDP port scans will be covered here.

Again the software Nmap [28] will be used to conduct the scans. To start the TCP port scan, the command from listing 4.5 can be used [36].

```
1  nmap -sS <IP of device> -p-
```

Listing 4.5: Scanning whole TCP port range with Nmap.

This will start a TCP port scan - using the SYN scanning method as described in section 2.4.1 - on the specified host, covering the whole TCP port range (1-65565). On my home network, this scan takes less than 15 seconds. As a result the ports accepting a TCP connection will be displayed. The results might look similar to these in listing 4.6. Again, the results have been shortened to save space.

```
1  Not shown: 65525 closed tcp ports (reset)
2  PORT      STATE     SERVICE
3  53/tcp    filtered  domain
4  68/tcp    filtered  dhcpc
5  80/tcp    open      http
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
6 515/tcp    open     printer
7 631/tcp    open     ipp
8 ...
9 18188/tcp  open     unknown
```

Listing 4.6: TCP port scan results from Nmap.

The scanned device in this example is a Samsung network printer (some lines have been omitted). For easier analysis, Nmap also prints the names of the services usually running on the found ports. A list of assigned ports by IANA can be found here [66]. In the example, it can be seen that some ports usually used for printing are open. Also, a web service is running. More information about analyzing web services can be found below in section 4.1.1.

Launching a UDP port scan works almost similarly to TCP. But, as described in section 2.4.1, results might be less meaningful than for TCP. The scan [27] can be started with the command from listing 4.7.

```
1 nmap -sU <IP of device>
```

Listing 4.7: UDP port scanning using Nmap.

According to the Nmap documentation [27], scanning the whole port range might take several hours. During the testing for this thesis, I only scanned the default port range automatically chosen by Nmap (1000 ports). This scan took approximately three and a half minutes on my network. As expected, the results - as seen in listing 4.8 - look slightly different than for TCP.

```
1 Not shown: 992 closed udp ports (port-unreach)
2 PORT      STATE          SERVICE
3 53/udp    open|filtered  domain
4 68/udp    open|filtered  dhcpc
5 161/udp   open           snmp
6 1900/udp  open|filtered  upnp
7 ...
```

Listing 4.8: UDP scan results from Nmap.

Again, the detected open ports can give a hint about what services are running on the device under investigation. For some services, Nmap has test patterns (here only for port 161), that can determine if a service is actually running and the expected one [27].

Although it is possible to classify devices by only looking at their open ports [49], analyzing services is usually necessary to receive more details about the device. More details include certain model number, firmware version and device-specific properties such as e.g. the supply status for printers. Nevertheless, the mechanisms and internals of most protocols are out of scope for this thesis. The amount and complexity of possible

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

protocols is too high and also not the main focus of this work. Following, approaches used in this thesis to get more information about services will be described.

DNS-SD over mDNS

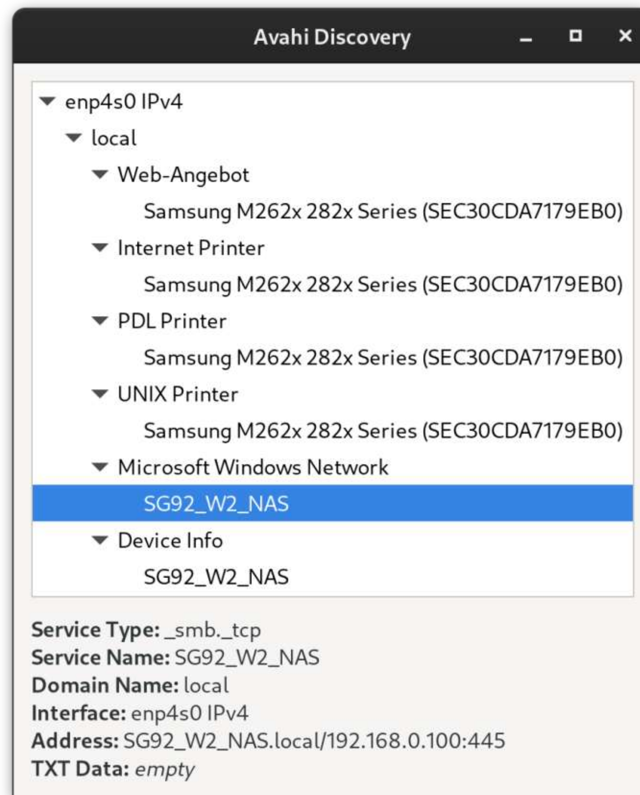


Figure 4.3: Avahi Discovery in Home Network.

DNS-SD is a protocol allowing for service discovery using the DNS infrastructure. mDNS enables DNS usage without any fixed servers. The mechanisms of DNS-SD and mDNS have already been covered in more detail in section 2.3.2. Furthermore, these two protocols enable network members to browse the network for available services [67]. One application enabling such browsing is included in the Avahi suite [68], called "avahi-browse" [67]. For further convenience, also a graphical version - "avahi-discover" - exists. A test scan of my home network can be seen in figure 4.3. A lot of information can be extracted from the presented services:

Devices A device that advertises a service over DNS-SD/mDNS, also reveals its presence in the network. Generally, device discovery has been covered before. Nevertheless, this mechanism can be used as an additional data source.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Open Ports and Protocols As explained in section 2.3.2, a service definition contains the port number and the information if the service uses TCP or something else. Therefore, a list of open ports can be composed, e.g. for use with other classification algorithms. Nevertheless, not all open ports need to be advertised this way, so results might be incomplete.

Service Names When conducting a port scan, a list of open ports will be the final result. The registered service list [66] from IANA can give hints on which service is running on a specific port, but this is not guaranteed. With DNS-SD the service records contain a service name. This name is often a protocol name or contains useful information on what service is running. Since this information is directly provided by the device under investigation, it is assumed not to be spoofed for this work.

While for TCP, port scanning works reliably and fast, it is more problematic for UDP. Services found through DNS-SD do not have to be scanned for. Also, more information is revealed, than by a port scan alone. All these advantages make DNS-SD over mDNS a useful mechanism to use for device analysis.

SSDP

As described in section 2.3.1, SSDP is a serverless discovery protocol defined in the UPnP specifications [31]. In this thesis, it is useful for device detection and analysis.

For service detection, a helper program called "gssdp-discover" exists. It is part of the GUPnP Gnome project [69]. To use it for root device discovery, the command as seen in listing 4.9 can be used.

```
1 gssdp-discover -t upnp:rootdevice
```

Listing 4.9: Command to discover UPnP devices using SSDP.

This will send a search request, only requesting information about root devices. To end the search, the program has to be exited. The result will look similar to listing 4.10.

```
1 resource available
2 USN:      uuid:...::upnp:rootdevice
3 Location: http://10.0.0.16:5001/description/fetch
4 resource available
5 USN:      uuid:...::upnp:rootdevice
6 Location: http://10.0.0.1:49000/fboxdesc.xml
7 ...
```

Listing 4.10: SSDP scan results.

After a few moments - less than one second on my test setup - all devices should be displayed. The URLs provided can be fetched with any HTTP client available. Usually,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

interesting fields from these documents have been described in section 2.3.1. But, due to the high amount of different devices, the usefulness of specific fields can vary per device type.

Using the SSDP, finding root devices is reliable and fast. Detecting control points in the current network is a bigger challenge, as they only reveal themselves when they search for services [31]. This behavior can also be seen in figure 2.9, where control points only send messages to search for devices and services. To read these search messages, the program Wireshark [70] can be used. Wireshark is a program that supports sniffing packages going through a network interface. It also provides various options to analyze the captured packets.

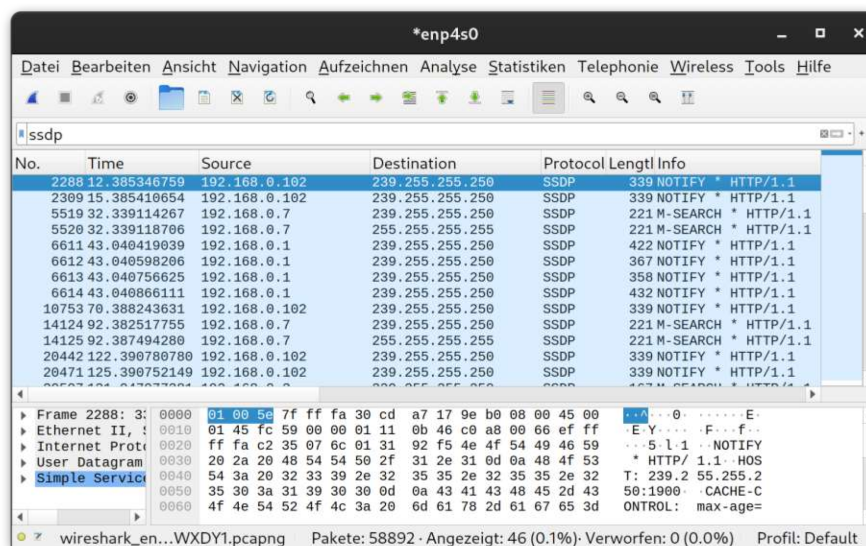


Figure 4.4: Wireshark Displaying Captured SSDP Packets.

In figure 4.4 an example of Wireshark with captured packets is shown. The filter options are set to SSDP, to display only relevant packages. Over time this will reveal many devices and their user agent strings. A captured packet might look similar to listing 4.11.

```
1 M-SEARCH * HTTP/1.1
2 HOST: 239.255.255.250:1900
3 MAN: "ssdp:discover"
4 MX: 1
5 ST: urn:dial-multiscreen-org:service:dial:1
6 USER-AGENT: Chromium/106.0.5249.119 Windows
```

Listing 4.11: SSDP search request.

The output makes clear, that the captured packet does not originate from an IoT device, but from a Windows PC. Not all devices send a user-agent string. E.g. the Panasonic

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

DMP-UB900 Blu-Ray player (which was available to me during my test runs) does not send this field at all.

After discovering a device matching its search criteria, a control point will usually fetch the descriptor to learn more about the device. For testing purposes, I ran a media server (UMS [71]) on my computer and again recorded the traffic using Wireshark. A Playstation 4 Pro has been used for this test. After launching the "Media Player" application on the device, the Playstation uses SSDP to search for media servers and - as assumed - fetches the descriptor from the locally running media server. The captured HTTP request contains the content as seen in listing 4.12.

```
1 GET /description/fetch HTTP/1.1
2 Host: 10.0.0.16:5001
3 Date: Thu, 29 Dec 2022 17:22:46 GMT
4 User-Agent: MediaPlayer/4.01 UPnP/1.0 DLNADOC/1.50 (PlayStation
      4 Pro)
```

Listing 4.12: HTTP GET request wuth user-agent.

Here also a user-agent string is revealed, stating the device type, but not the concrete model (there are multiple submodels of the Playstation 4 Pro available) or firmware version. Again, depending on the device under investigation, the usefulness of the result will vary.

Using SSDP for device analysis, multiple device properties can be revealed:

Device itself As with DNS-SD over mDNS, a device sending multicast messages reveals its existence in the process. This can be useful if other detection techniques are unavailable or the results are incomplete.

Device descriptor Root devices provide a device descriptor, that usually contains information about the device, including manufacturer and model. This document usually exposes a lot of information that can be used for fingerprinting.

Device type Listening to the search messages, it might be possible to deduce which type of device is sending them. For example, a media render will usually be searching for media servers. Nevertheless, this type of information gain is not used in this work.

Web Services and REST-APIs

Many devices, e.g. network printers, expose web services to the connected network. These can be used to gain a lot of information about the device under investigation. Also, REST interfaces often can be found running on these devices, enabling other machines on the same network to get information for various purposes, e.g. displaying them in other places like apps.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Sometimes manufacturers provide documentation on how to use the provided interfaces. This section is about partially reverse engineering these interfaces when no documentation is provided. I do not aim to provide a full guide for reverse engineering web interfaces here but to give a few starting points for further investigations. The whole topic is very large and complex and out of scope for this thesis. For the investigations, Firefox - at the time of writing version 108.0.1 - is being utilized. As testing device, a Samsung M262x 282x Series printer is investigated. Its web service - called SyncThru - is running on port 80.

Figure 4.5: Supplies Displayed in Printer's Web Interface.

The web browser is being used to find a page displaying interesting information. Below an incomplete list of items to search for in the printer's web interface can be found:

- Device information (Manufacturer, Model,...)
- Firmware information (Version, Variant)
- Supply information (Toner level, Drum,...) (printer-specific)

This list is incomplete because the needed and wanted information might differ from device to device. Also, there is no guarantee that the wanted information is provided by the device's web interface. As can be seen in figure 4.5, on the printer's main page information about the supply status can be found.

For analysis, Firefox's web developer tools are used. With F12 these can be enabled. Setting the filter options to XHR in the network tab will display all XMLHttpRequests issued using JavaScript. The page has to be reloaded to record the requests. Usually, the request fetching the wanted information is among the displayed lines after a few moments (about 5 seconds for the printer used in this test). The displayed results might - dependent on the device under investigation - look similar to figure 4.6.

Among other information, it can be seen that a request is sent, requesting a ".json" file. The contents of this file are (partially) displayed on the right side. Below a cut version of this file can be found in listing 4.13, displaying only the relevant lines for this work.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

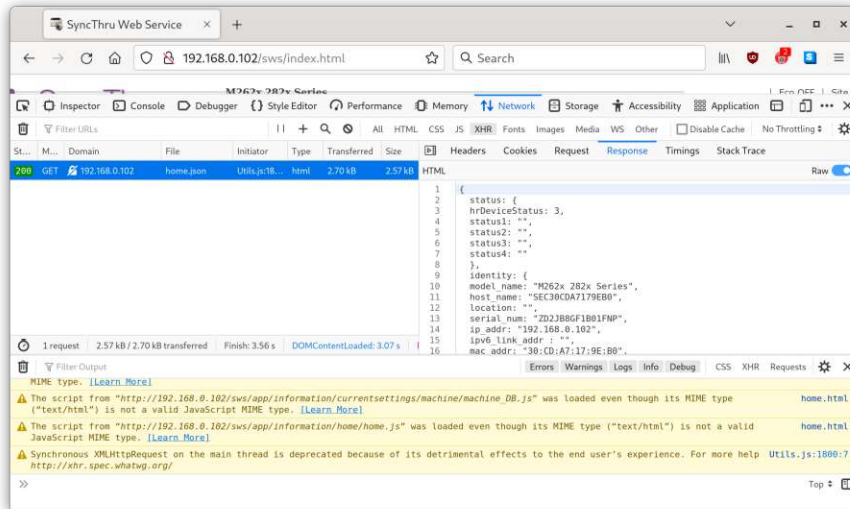


Figure 4.6: Firefox Developer Tools.

```
1  {
2   identity: {
3    model_name: "M262x 282x Series",
4    host_name: "SEC30CDA7179EB0",
5    ip_addr: "192.168.0.102",
6    mac_addr: "30:CD:A7:17:9E:B0",
7   },
8   toner_black: {
9    opt: 1,
10   remaining: 0,
11   cnt: 2601,
12   newError: "C1-1150"
13  },
14  drum_black: {
15   opt: 1,
16   remaining: 94,
17   newError: ""
18  },
19  options: {
20   wlan: 0,duplex: 1
21  }
22  ...
23 }
```

Listing 4.13: Printer information in JSON format.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

It can be seen that not only the supply status but also capabilities and general system information are transmitted. As the data is in machine-readable form (JSON), it is easy to programmatically extract information and display it to the user. Without any manufacturer documentation, the format and range of the provided data have to be assumed. Again, this section does not aim to be a complete guide for reverse engineering web interfaces but to give starting points for further investigations.

NetBIOS Names

NetBIOS is an old protocol, designed by IBM in 1983 [72]. Originally, the protocol did not use TCP/IP for data transport, but that changed soon as TCP became more popular. NetBIOS over TCP was standardized in 1987 [73] and the old transport protocols were no longer mandatory from Windows 2000 upwards.

Although being an old protocol, NetBIOS over TCP can reveal some information, like vendor and model, from a device using the NetBIOS name [74][51]. Even today, many devices still expose a NetBIOS name. For example, the Samba software still has the option to set a NetBIOS name [75]. Also newer versions of Windows and Windows Server support NetBIOS [74]. Nevertheless, the usefulness of the provided information varies from device to device. To get the NetBIOS names from a remote device, the command from listing 4.14 can be used.

```
1 nmblookup -A <IP of device>
```

Listing 4.14: Resolve IP address to NetBIOS names.

Nmblookup is part of the samba software [76]. For Windows a similar program, called nbtstat [74], is available. The output of the command above will look similar to listing 4.15.

```
1 VUZERO          <00> -          B <ACTIVE>
2 ...
3 VUPLUS          <00> - <GROUP> B <ACTIVE>
```

Listing 4.15: NetBIOS names lookup result.

It can be seen, that the device used for this test - a VU+ Zero satellite receiver - reveals its vendor (VUPLUS) and its concrete model (VUZERO) through its advertised NetBIOS names. But, as stated before, not all devices expose such detailed information over this protocol.

A major advantage of the manual fingerprint generation technique is the possibility to analyze any device, without any restrictions regarding operating systems and firmware image or documentation availability. Done carefully, this approach can lead to very specific and unique fingerprints that do not easily produce misclassifications.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

4.1.2 Firmware Image Analysis

As described before the manual approach might yield good results, but is not applicable to a large number of different devices. In this section, an automated approach will be described using a Python program. This program takes firmware images, analyzes them and extracts properties that can be used to automatically determine a remote device's type.

Targets

To speed up fingerprint generation, minimal user interaction should be necessary in order to generate fingerprints. That means, only the source directory needs to be specified. During the program run, no interaction with the user is needed. Any archive format supported by the python-libarchive [77] Python library can be used with the program. Other archive types need to be converted first. Also, raw firmware images (as produced by e.g. the Unix tool dd) can not be used without manual work first.

The target here is to automatically find properties of devices, without the long manual process presented in section 4.1.1. Also, the three target characteristics for properties - as explained at the beginning of section 4.1 - have to be considered when gathering information.

Only Linux firmware images conforming to the standard folder and file naming conventions [78] will yield useful results. Also, other similar operating systems (e.g. other unixoid systems like FreeBSD or macOS) might work, provided that the image is in a supported archive format. Nevertheless, the program has only been tested against Linux firmware images. Any sort of encryption will also lead to no result.

File Structure

The results are stored in a JSON-structured text file. This file can then be read by any compatible scanning tool. Each file contains an array of fingerprints. Listing 4.16 shows an example of a single fingerprint entry.

```
1  {
2   "dns_sd": [
3   "_ssh._tcp",
4   "_sftp-ssh._tcp"
5   ],
6   "hostname": "SampleDevice",
7   "telnet": "SampleDevice 1.0.1\n",
8   "netbios": "SAMPLEDEVICE",
9   "upnp":{
10    "friendly_name":"Sample Device",
11    "manufacturer":"Sample Manufacturer",
12    "model_description":"Sample for paper"
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
13   }
14   "names": [
15   "SampleDevice_V1.tar",
16   "SampleDevice_V2.tar"
17   ]
18 }
```

Listing 4.16: Sample fingerprint entry

Nearly every entry is optional. Per fingerprint, only one name in the names array is needed, and at least one property has to be provided. Multiple names are supported, because multiple firmware images may result in the same fingerprint. If that is the case, both fingerprints get combined into a single one with multiple names.

Extracted Properties

In this section, the patterns used to find interesting files and information will be presented. It will also be stated which information in the archive file is transformed to which part of an automatically generated fingerprint.

Samba Configuration Samba is software for Microsoft Windows compatibility in other operating systems [76]. It also implements the NetBIOS protocol. As stated before, each NetBIOS device has its own name in the network. This name can be read over the network. In a system image, this name can be found in the Samba configuration. Samba configurations usually have the name "smb.conf", this is also what the fingerprint generation software looks for. The file is structured in a key-value format, separated by sections. An example of such a configuration file can be seen in listing 4.17. If an option called "netbios name" is found in the file, the value is copied into the fingerprint.

```
1 [global]
2 workgroup = WORKGROUP
3 netbios name = RT-AC88U-F000
4 server string = RT-AC88U-F000
5 unix charset = UTF8
6 ...
```

Listing 4.17: smb.conf from an ASUS-RT-AC88U Firmware

Hostname When using DHCP, it is possible to send a hostname to the DHCP server [79]. This hostname is usually stored in "/etc/hostname" on UNIX machines. This file is read by the fingerprint creation tool and the full content is written into the "hostname" field of the fingerprint. This property is detectable when the DHCP server is connected to a DNS server, which is capable of reversing IP addresses back to hostnames [80]. The firmware archive may not start with the Linux system root folder as its own root. Therefore the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

tool looks for files whose path ends with "etc/hostname". That way it is also possible to detect hostnames in nested system folders.

UPnP Descriptions UPnP descriptions are usually located in XML files. Since checking all advertised services might lead to very large fingerprints, only device information is being searched for. All devices that advertise UPnP services also supply a device description [81]. To find these descriptions all files ending with ".xml" are checked if they belong to the correct namespace. To do so, all XML files are checked for the header as can be seen in listing 4.18. When this check succeeds the three fields "friendlyName", "manufacturer" and "modelDescription" are copied to the new fingerprint as a dictionary.

```
1 <?xml version="1.0"?>
2 <root xmlns="urn:schemas-upnp-org:device-1-0">
```

Listing 4.18: UPnP Device Descriptor Header

Avahi Services Avahi is software that can find and advertise services on the local network [68]. Advertised services are usually stored in "/etc/avahi/services" inside files ending with ".service". For this work, primarily the service type (e.g. "_nfs._tcp" for a network share) is interesting. This is also the only information that will be copied to the generated fingerprint. As multiple services can be advertised by a single device, the resulting output will be an array of service types that can be expected from the device. An example service definition, advertising an HTTP service, can be found in listing 4.19.

```
1 <service-group>
2  <name replace-wildcards="yes">%h</name>
3  <service>
4   <type>_http._tcp</type>
5   <port>80</port>
6  </service>
7 </service-group>
```

Listing 4.19: HTTP Service Definition from a FLIR Camera

Telnet banner Despite the security implications, some devices still have a telnet service enabled. Without any login credentials needed, the server will send the content of the "/etc/issue.net" file [40] to any connecting client. Therefore this file is automatically extracted from any firmware image and copied to the fingerprint as it is.

4.2 Scanning Strategies for IP-Based Networks

Before, two approaches for information gathering and generating fingerprints have been described: manual and firmware analysis. To use these fingerprints, different scanning strategies are being utilized. Currently, two scanning strategies are used in sequence.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

First, the tree-based approach is being applied with the manually gathered information. Any devices that could not be classified by this approach will be checked again using the automatically generated fingerprints. In this section, the two scanning strategies will be explained.

Device scanning assumes that the remote devices already have been detected. Network scanning is therefore not part of any scanning strategy. Background information about device detection can be found in section 2.2. Details about the Android implementation are described later in section 4.4.2.

4.2.1 Tree Based Approach

As stated above the tree-based approach is used first on any alive host. It uses manually found properties and is usually more flexible than the second scanning approach presented below in section 4.2.2. Compared with the second approach, tree-based scanning has the following characteristics:

Higher-quality classification Device properties are hand-picked and therefore should provide more accurate classification, with very few false positive hits.

Low bandwidth usage Only requests necessary for the current detection step are sent. This should minimize the network traffic necessary to classify devices.

Individual results A hit using this classification approach does only output a single result.

Nevertheless, the amount of detectable devices does not scale very well and every device or device type needs to be inserted manually into the application source code. As mentioned before, a possible solution to this problem is introduced in section 4.2.2.

Detection Levels

As the name of the approach suggests, multiple detection steps can be performed on a single device. After every detection step, the device gets assigned a detection level. The meaning of the different detection levels can be found in table **??**. Only devices with detection levels of 1 or upwards get shown to the user.

Most interesting are the detection levels 1 to 4, as they deal with the classification of potential IoT devices. The levels below are only used for the internal flagging of devices.

Detection Procedure

The detection sequence is separated into detection steps and pools. A pool might contain an arbitrary number of steps but has to contain at least one. A detection step is always part of a pool and can have another pool as a successor. This successor is used when the detection step is successful.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Detection Level	Meaning	Example
-1	Non IoT device	PC/Smartphone
0	Device detected	-
1	Device and vendor found	HP device
2	Device type detected	Printer
3	Device model detected	Specific model name
4	Additional information available	Supply levels

Table 4.1: Detection Levels

Based on the vendor found in detection level 1 the first pool is selected and the first detection step is started. This means the initially selected pool is based on the MAC address of the device under investigation. If no pool is registered for a MAC prefix, the device remains in detection level 1. No further investigation is done by the tree-based scanning method. But it will be looked at again by the next scanning strategy described in section 4.2.2.

If a detection step fails, the next step in the same pool will be executed. If the step succeeds another bit of information about a device has been found. That means that the detection level will be increased and additional information will be stored and displayed in the Android App. If another pool is set as the successor to a successful step, the first step in this pool will then be started. If no next step or pool can be found the detection sequence will be quitted. An illustration of the scanning process can be seen in figure 4.7.

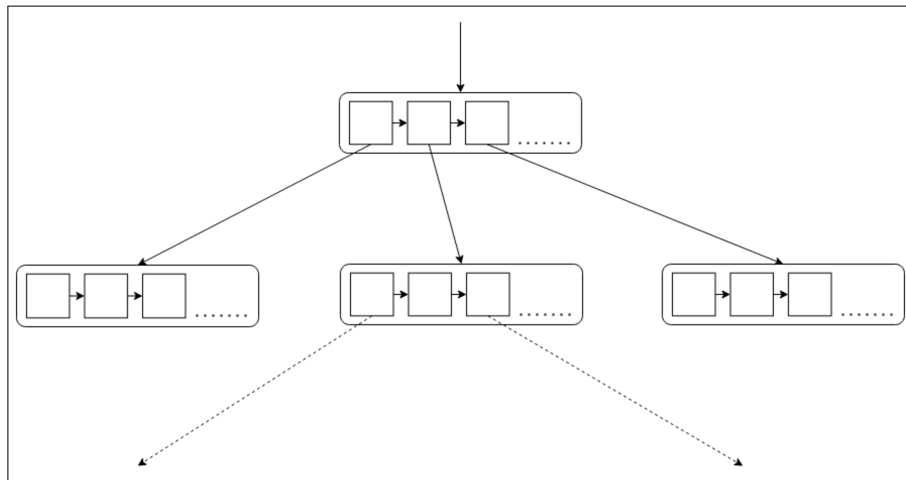


Figure 4.7: Tree Based Scanning.

The detection level does not necessarily need to be increased by one. It is acceptable that a single detection step raises the level from 1 to 3 or even 4. That can happen if, e.g. a web server runs on the device and directly sends the model or other information to the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

application. All devices reaching a detection level higher than 1 will not be investigated by the next detection strategy described below.

Additional Information About Devices While detection level 3 means, that the device model has been determined, a detection level of 4 indicates that even more information about a device can be displayed. This might be all sorts of information, like supply status from a network printer, or even screenshots from an internet-connected satellite receiver. The Android application provides a framework for filling in this additional information and displaying it to the application user. The way this information is acquired is individual per device. The detection level of 4 only indicates that additional information can be obtained, not that it is already in some form of cache.

4.2.2 Approach Using Automatically Generated Fingerprints

When no successful classification procedure is available in the tree-based approach, a second approach, explained in this section, will be tried. Here the generated fingerprints from section 4.1.2 are being used and compared against the actual values received from the scanned devices. Any device that has a detection level below 2 counts as basically unclassified and will be analyzed using the method described here. This detection method has some advantages over the tree-based classification:

Scalability Due to the automatic generation of fingerprints, new devices can be added quickly to the fingerprint database. This allows for a high amount of recognized devices, with minimal manual work.

Constant effort The steps needed to classify a device are known in advance (with a few exceptions). The data requested is the same per device. Therefore this approach is easier to debug and maintain.

A major downside of this approach is that a device may match multiple fingerprints. In that case, the fingerprint with the most properties stored wins, as more information could be extracted from the corresponding firmware image. A fingerprint with less properties is therefore considered less meaningful. Another option is that multiple firmware images resulted in the same fingerprint. If this happens, a list of possible types will be assigned to the device. Details about this detection strategy will be explained below.

Scanning Procedure

Compared to the tree-based scanning procedure, a more static approach is being used here. The following steps are being executed in order to search for matches using the automatically generated fingerprints.

Device collection As described before, only devices that have a detection level below 2 will be looked at. These are collected from the general list generated in section 2.2. As with the tree-based approach, device detection is not part of the classification strategy.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

General data acquisition All properties described in section 4.1.2 only need to be detected once. In this step and before any fingerprint matching occurs, all properties are queried and stored in the local memory for further comparison. Only the DNS-SD services need special treatment. It is not possible - with the default Android implementation - to get all advertised services from a device at once (no browsing capabilities in the standard API). Details about the implementation can be found in section 4.4.3.

Comparison with fingerprints In this step all found properties are compared against the values stored in the generated fingerprints. After the comparison a value between 0 and 1 will be calculated, which indicates how well the scanned device and the fingerprint match. A value of zero indicates no match, while a value of one indicates a perfect match. All data points in the fingerprint must be found on the actual device. If a data point is present in the fingerprint, but not on the actual device the result is always zero. If a data point does not match the observed value the result is also always zero. If all data points on the fingerprint match, the total score is calculated as:

$$\frac{dataPointsInFingerprint}{datapointsFoundOnDevice} \tag{4.1}$$

Detecting Avahi services As mentioned before, Android's default API can not detect all advertised DNS-SD services from a single device. Therefore each service in the fingerprint list has to be checked individually. This means every other property will be compared first. If all these checks succeed, the services listed in the fingerprint will be tested sequentially. As with the other properties, detected services will be stored in memory and not queried again in the network.

Choosing best match When all fingerprints have been checked, the one with the highest match value, calculated as explained above, will be chosen. All names (if multiple) stored in this fingerprint will be returned as a list. Two fingerprints can not have equal properties, as they would have been merged.

4.3 Bluetooth Classification

The Android application not only supports scanning the current WiFi-connected IP-based network but also the Bluetooth environment. In this section, it will be explained how surrounding devices can be found and the strategies to classify them. Both, Bluetooth Classic and Bluetooth Low Energy (LE) will be covered in this section.

4.3.1 Bluetooth Classic

Despite being published in 1999, the classic variant of Bluetooth is still widely in use today [41]. To keep up with changing device and user requirements, many revisions and changes have been published since then. As it is the older standard overall, Bluetooth Classic will be treated first.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Detecting Devices

As explained in section 2.5.1, it is not possible to detect devices in undiscoverable mode with an unmodified Android phone. Therefore this work is limited to discoverable devices only.

Classification

Also for Bluetooth, the hardware address (MAC address) only references the hardware vendor, but not the concrete model. Bluetooth devices usually advertise a name, that can be used by a user to identify them [45]. This display name can often be changed. Therefore both properties should not be used to identify the device model, as they do not conform to the goal characteristics for classification properties.

The method used in this section is based on Blueprinting [60] by Herfurth and Mulliner. In their work, SDP is used to get the advertised profiles from a remote device. After service discovery is complete the RecHandle and the Channel are extracted from every SDP profile. Details on the data structures used during this process can be found in section 2.5.1. These retrieved values are then used for checksum calculation as seen in figure 4.8.

RecHandle	Channel	Product
0x1000b	2	131094
0x1000c	9	589932
0x1000d	1	65549
0x1000e	15	983250
0x1000f	3	196653
0x10010	13	852176
0x10011	12	786636
		3605290

Figure 4.8: Blueprinting checksum calculation[60].

The whole fingerprint consists of the vendor part (first 24 bits) of the MAC address and the previously presented checksum (e.g. 00:60:57@2621543 for a Nokia 6310i) [60]. At the time of writing, it is not possible to retrieve the necessary information on Android. Instead, all services are identified by their UUID [82]. For my Android application, the procedure has been modified, such that it uses the UUIDs provided by the Android API instead of the values originally used by Herfurth and Mulliner.

For this modified procedure, first, all UUIDs are fetched from the remote device. In the next step, they are sorted alphabetically ascending. Finally, all UUIDs get concatenated and the resulting string is hashed with SHA256. This modern hash function ensures

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

collision freeness, while still retaining a constant size for all fingerprints. The vendor part of the MAC address is being put in front of the hash value, as in Herfurth and Mulliners work. An example of a fingerprint from my headphones can be found in listing 4.20.

```
1 80:C3:BA;05
    f4a45cb6cdc90302f47b1c745381ee382cde0e34614e55ae8b7c1885a11edc
```

Listing 4.20: Sennheiser Momentum True Wireless 3 - Fingerprint

Internally fingerprints are stored, like the automatically generated fingerprints from section 4.1.2, in a JSON data structure. This data structure stores the hash value and a device name for every data set. An example of such an entry can be seen in listing 4.21.

```
1 {"name": "Sennheiser Momentum True Wireless 3", "vendorMac":
    "80:C3:BA", "hash": "<Hash as above>"}
```

Listing 4.21: Stored Fingerprint Example.

For every detected device, the previously presented hash value is computed. After the scan is completed, the results of all found devices will be compared to the list of known fingerprints. If a match is found, the stored device name is displayed.

Sometimes a hardware vendor uses more than one MAC prefix. In this case, the fingerprint would not match, even if the calculated hash values are the same. To prevent this from happening, the MAC table is searched for the names of the device vendors (fingerprint and device under investigation). If these names match, the fingerprint matches, even if the MAC vendor prefixes are different.

4.3.2 Bluetooth Low Energy

Bluetooth Low Energy is used for devices that need a lower energy consumption than classic Bluetooth [41]. Therefore, some processes are done differently for this type of device. This section deals with classifying devices using the Low Energy variant of Bluetooth.

Detecting Devices

Detecting devices for Bluetooth LE is less problematic than for Bluetooth Classic, as there is no such thing as an un-/discoverable mode. If a device does not want to be found it stops sending advertisement packages. In my testing, devices continued sending advertisements, even after they were connected to some other device. Android provides API calls for finding nearby BLE devices and notifies the application when a device is found [46]. Details about the implementation can be found in section 4.4.4.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Classification

Service discovery in Bluetooth LE works differently compared to Bluetooth Classic. As explained before, the service discovery protocol is being used for the Classic variant of Bluetooth. For the newer Low Energy variant, this is replaced by GATT [83]. More information about GATT can be found in section 2.5.2. GATT uses a client-server architecture, where the Low Energy device acts as a server, while the Android phone has the client part. Using the GATT server not only information about advertised services can be acquired, but also so-called properties can be read and written.

The classification method for this work has been derived from Celosia and Cunche [62]. They stated that many devices reveal much information about themselves openly via their GATT server. Therefore, Bluetooth LE device classification does not rely on some checksum or fingerprint, but on a standardized service called "Device Information Service" [84]. From this service the following values are read:

- Manufacturer Name
- Model Number
- Software Revision

The used service and the values have standardized identifiers, which can be found in the Assigned Numbers Document [48]. As the retrieved information represents what the Android application tries to find out eventually, no further processing is needed.

4.4 Android Implementation

Before, it has been discussed which properties can be utilized to detect IoT devices in the current environment. Also, approaches to make use of these properties have been described. In this section, it will be presented how this knowledge is being transformed into an Android application.

4.4.1 General Information

The Android application is developed using the IDE Android Studio in version "Dolphin". As testing device, a OnePlus 5 [85] with LineageOS 19 (based on Android 12) [86] is being used. The application is designed with research in mind. This means that the user interface might not be fully polished for public release and might not adhere to Google's user interface guidelines. The interface is built to provide quick access to necessary application functionality and to display information useful for the research done in this work.

As programming language, Kotlin has been chosen and only official Android functions are used strictly. The goal is to make the application executable on off-the-shelf Android

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

phones, that do not have modified software or hardware. Therefore, no external hardware, custom kernels, or other software modifications have been used. Also, root access is not allowed (with a single exception explained later).

The focus of this section lies on the technical details of remote device property detection using Android. Therefore, any code concerned with graphical output, or simple calculations and decisions are cut out from the code samples. This is done to make the samples better understandable and to highlight the important parts of the code.

4.4.2 Device Detection

Prior to investigating, devices have to be detected first. Possible ways of device detection have been explained in section 2.2. Since root access is unwanted, many paths described in the mentioned section are not usable in Android. This section is about determining the search space and how the actual search is conducted using the limited Android API.

Determining Search Space

Before scanning can start, the search space has to be determined. All addresses that are not the standard gateway or the scanning device itself should be tested. Only IPv4 addresses are of concern for this thesis. First, the current link address of the WiFi interface has to be determined. This is done using the code from listing 4.22.

```
1  val connectivityManager = getSystemService(Context.
      CONNECTIVITY_SERVICE) as ConnectivityManager
2  val currentNetwork = connectivityManager.activeNetwork
3  val linkProperties = connectivityManager.getLinkProperties(
      currentNetwork)
4  var myAddress: LinkAddress? = null
5  linkProperties!!.linkAddresses.forEach { current ->
6   if (current.address is Inet4Address) {
7    myAddress = current
8    return@forEach
9   }
10 }
```

Listing 4.22: Get current WiFi IP address.

Prior to this code running, it is ensured that a WiFi network is connected. The code uses official Android API functions to interact with system services. These API calls are not part of the standard Java API. If somehow multiple addresses are configured, only the first one is investigated. Determining the search space can be done as described in section 2.1.3, but for the Android application an external library function is being used. The SubnetUtils from the Apache Commons Net [87] library provide functions to get all IP addresses from a IP address and subnet combination.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

After all potential IP addresses have been determined, the own address and the gateway (also supplied by the Android API) are removed. The remaining addresses are targets for scanning.

Device Scanning

Following the previous steps, all IP addresses are available in a string format. The Android API provides functionality to check, if a network host is alive [88]. To speed up device detection, coroutines are used for parallel scanning. The code used can be found in listing 4.23.

```
1  fun ping(ip: String, timeout: Int = 100): Boolean {
2   return InetAddress.getByName(ip).isReachable(timeout)
3  }
4
5  repeat(filteredSearchSpace.size) { i ->
6   launch {
7    val current = filteredSearchSpace[i]
8    if (ping(current)) {
9     val device = DetectedDevice(current)
10    //UI code...
11    treeStrategyCoordinator.classifyDevice(device)
12   }
13  }
14 }
```

Listing 4.23: Testing devices using coroutines.

The list "filteredSearchSpace" contains the IP addresses in the search space as a list of strings. Every entry gets investigated in its own coroutine. If an active device is found, the first scanning strategy - the tree-based scanning from section 4.2.1 - is launched for that device. Other methods - as described in section 2.2 - are not usable on unmodified Android. To use them, raw rockets need to be opened, for which a program needs special permissions to do so] [29].

Alternative Ping Function Before upgrading to LineageOS 19, the "isReachable" method used above did not work. It always returned false, even for the own or loopback IP addresses. To work around this restriction, the program "ping" as provided by the system has been used. This variant can be seen in listing 4.24.

```
1  fun pingAlternative(ip: String): Boolean {
2   val cmd = "ping -c 1 -W 0.1 $ip"
3   val process = Runtime.getRuntime().exec(cmd)
4   if (process.waitFor(100, TimeUnit.MILLISECONDS)) {
5    return process.waitFor() == 0
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
6   }
7   process.destroyForcibly()
8   return false
9 }
```

Listing 4.24: Device search using ping.

The "ping" program is used to issue a single ICMP ping request to the target. Despite having a timeout of 0.1 seconds specified, the program terminates after one full second of waiting. To get the waiting time down to the wanted value, "ping" is terminated by the Android application after the specified waiting time. If Android terminates the program, the scanned host is assumed to be unreachable. When the "ping" terminates in time, the result as returned by the program is being used.

4.4.3 Property Detection

In the sections 4.1.1 and 4.1.2 detectable properties on IoT devices have been introduced. Since Android does not allow access to all system APIs, not all scanning methods are available like on a desktop machine. In this section, it will be explained which properties of a remote machine can be detected and where the limits are with Android.

MAC Address

The MAC address has to be treated in a rather specific way on modern Android versions. Since Android 10, access to the ARP table is restricted [89]. Without root access, it is no longer possible to fetch the MAC address of network neighbors, as this information is hidden due to privacy reasons. For this work, not being able to access the MAC addresses from network neighbors is a significant restriction. As explained in section 4.2.1, the MAC address is the starting point for analysis using the tree-based scanning approach. Since the testing device is also being used as my daily driver, the software version was not constant during the writing of this thesis. In the beginning, the Android version was below 10. Therefore, it was possible to access the ARP table without any restrictions. After updating Android, the tree-based approach was no longer functional. To avoid losing an interesting approach for classification, I decided to use root to access the ARP table anyway. This is the solely exception to the "unmodified Android" rule for this thesis. Also, the scanning using automatically generated fingerprints - as described in section 4.2.2 - is designed to avoid using the MAC address. The code used to access the MAC address from Android 10 upwards is shown in listing 4.25.

```
1 fun getMAC(ip: String): String? {
2   if (!ping(ip)) { //Make sure host is in ARP table
3     return null
4   }
5   var mac: String? = null
6   //Use root to read ARP table
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
7     val process = Runtime.getRuntime().exec(arrayOf("/system/bin/
         su", "-c", "ip neigh show"))
8     process.waitFor()
9     val output = process.inputStream.bufferedReader()
10    output.forEachLine { line ->
11     val split = line.split(" ")
12     if (split[0] == ip) {
13      mac = split[4]
14      return@forEachLine
15     }
16    }
17    return mac
18  }
```

Listing 4.25: Get MAC address using root.

Pinging the target host ensures that the MAC address will be stored in the ARP table. If the target host is not active, or any other error occurs, this method will not deliver any result. After acquiring root privileges, the system program "ip" is used to list the content of the ARP table. An example of an ARP table entry can be seen in listing 4.26.

```
1   10.0.0.1 dev enp0s31f6 lladdr 38:10:d5:18:23:2c REACHABLE
```

Listing 4.26: ARP table example entry.

Most important here are the first and the fifth column, as they represent the IP address and the MAC address accordingly. After buffering the output from the program "ip", the table is searched programmatically and the first matching line is returned.

Port Scanning

TCP and UDP services running on a network device need to be accessed using their corresponding ports [21][25]. Both protocols can be used with Android using the built-in functions. When scanning TCP ports it can be clearly distinguished between open and closed ports, as a connection has to be established before data can be transferred [36]. Searching for open UDP ports is more complicated. No connection has to be established and for getting responses it is necessary to know the protocol used [27]. On non-rooted Android, there is no way to send raw data packets over the network interface [90]. Therefore, some quick ways of scanning through entire port ranges, as implemented in Nmap, are not available.

TCP As no raw sockets are available, only the TCP connect method as seen in Nmap [36][38] can be used. The different scanning methods have already been explained in 2.4.1. To check a port range, the Android device tries to establish a full TCP connection with every port that should be scanned. As the Android API for sockets [91] needs to be used, two outcomes are possible:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Open When the socket construction succeeds, the connection is fully usable. As already done with the TCP handshake, packets can be sent in both directions without any firewall dropping them.

Error When the socket creation throws an exception, it is not fully clear what caused it. ICMP error messages or timeouts are among the causes of a socket creation failure. The Android API does not expose more information. The socket is not useable and no communication is possible.

Despite all the limitations, it is still enough information to know if a service can be reached or not. The code used for this functionality can be seen in listing 4.27.

```
1  fun isTcpPortReachable(ip: String, port: Int): Boolean {
2   return try {
3     val sock = Socket(ip, port)
4     sock.close()
5     true
6   } catch (e: IOException) {
7     false
8   }
9  }
```

Listing 4.27: TCP port check.

UDP As described in section 2.4.1, it might be difficult to get useful results for UDP scanning. Despite the difficulties, not much information is lost when comparing the achievable scan results from Android with those from Nmap [27]. Looking at the built-in API for UDP communication in Android [92][93], UDP packages can be composed, sent, and received without any restrictions. Also, as a difference to TCP, it is possible to detect an ICMP port unreachable message when using UDP on Android. The source code, used for testing UDP ports on Android, can be found in listing 4.28.

```
1  fun isUdpPortReachable(ip: String, port: Int, timeout: Long =
       100L, testData: ByteArray = "PortCheck".toByteArray()):
       Boolean {
2   return try {
3     val sock = DatagramSocket()
4     val addr = InetAddress.getByName(ip)
5     sock.connect(addr, port)
6     val testPacket = DatagramPacket(testData, testData.size)
7     sock.send(testPacket)
8     sleep(timeout)
9     sock.send(testPacket)
10    sock.close()
11    true
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
12  } catch (e: PortUnreachableException) {
13   false
14  } catch (e: IOException) {
15   false
16  }
17 }
```

Listing 4.28: UDP port check.

To trigger a "PortUnreachableException" using UDP, two packets need to be sent. The first one will be sent without any errors. When trying to send the second packet, the exception will be thrown due to an ICMP port unreachable message received during the waiting time in between. As stated before, there is no guarantee that an ICMP message will be sent. If both messages can be sent without any errors, the method will return true. This indicates that a service is listening to the port under investigation, or that all packets are dropped silently. If any error occurs, the port is assumed to be unreachable. In the example above, no special action is taken when an ICMP port unreachable message is received, as for my testing the reason for unavailability is of no concern.

Service Responses

As explained before, in Android TCP and UDP connections can be established and used. Therefore all responses coming from services, reachable with these protocols, can also be captured [94]. Libraries can be used to simplify supporting complex protocols and gain even more information about remote devices. This section focuses on TCP connections, as - in my experience - most services of interest for IoT classification use this protocol.

Banner Grabbing A simple form of receiving information from a service is banner grabbing. This approach has already been thoroughly analyzed in section 2.4.2. The code used for Android can be found in listing 4.29.

```
1  fun getTcpBanner(ip: String, port: Int, timeout: Int = 100):
      String? {
2   var data = ""
3   try {
4    val sock = Socket(ip, port)
5    sock.use {
6     it.soTimeout = timeout
7     val inp = it.getInputStream().reader()
8     while (true) {
9      try {
10      data += inp.read().toChar()
11     } catch (_: SocketTimeoutException) {
12      return@use
13     }
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
14      }
15    }
16  } catch (e: ConnectException) {
17    return null
18  }
19  return data.ifEmpty { null }
20 }
```

Listing 4.29: TCP banner grabbing.

For banner grabbing a TCP socket is created, using the supplied IP address and port. After that data is read until the timeout is reached. If any data has been read, it is returned to the calling function. All opened resources are closed after use.

HTTP Many (IoT) devices expose web servers, that can be used for gathering more information about the device in question. To aid with application development, a library like Retrofit [95] might be adopted. Retrofit is an HTTP client, that also supports the de-/serialization of data passed through the connection in various formats, e.g. XML and JSON. In listing 4.30 an example is shown, demonstrating the use of Retrofit to fetch data from a remote API.

```
1 val retrofit = RetrofitHelper.getInstance("http://$ip/api")
2 val api = retrofit.create(RemoteWebApi::class.java)
3 val data = api.getInfo().execute().body()
4 interface ExampleApi {
5   @GET("/infoEndpoint")
6   fun getInfo(): Call<InfoResponse>
7 }
```

Listing 4.30: Usage example of Retrofit.

The Retrofit library allows to access information distributed using HTTP. Through the use of annotations, the structure of the remote API is declared. Data formats are described using the built-in Kotlin/Java data types. Describing all capabilities and the usage of the library is out of scope for this work, nevertheless, details about the usage of Retrofit can be read in the official documentation [95].

Device Descriptors using SSDP/UPnP

SSDP is a protocol used to advertise and find services on the local network [81]. Since UDP multicast data transfer is being used, the protocol only works in the local network but not across the internet. One big advantage of detecting devices using SSDP is that only a small amount of traffic is generated during service discovery. Nevertheless, not all devices can be found using this method, as not all IoT devices implement the protocol. All needed networking features are fully supported by Android. During App development,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

an external library that only utilizes Java API calls for network communication has been used [96]. It is also advertised as being fully compatible with Android. In my application, this library is used to discover SSDP compatible devices in the current network and to locate their device descriptors. The code used to discover the devices is shown in listing 4.31.

```
1  fun scanUPNP(timeout: Long = 1000): Collection<SsdpService> {
2   val devices = mutableListOf<SsdpService>()
3   val ssdpClient = SsdpClient.create()
4   val listener = object : DiscoveryListener {
5    override fun onServiceDiscovered(service: SsdpService?) {
6     if (service != null) {
7      devices.add(service)
8     }
9    }
10   override fun onServiceAnnouncement(announcement:
       SsdpServiceAnnouncement?) {}
11   override fun onFailed(ex: Exception?) {}
12  }
13  ssdpClient.discoverServices(SsdpRequest.discoverRootDevice(),
      listener)
14  Thread.sleep(timeout)
15  ssdpClient.stopDiscovery()
16  return devices
17 }
```

Listing 4.31: Device discovery using SSDP.

This function actively searches for UPnP devices, ignoring errors or other advertisements captured on the network. After the specified timeout the search ends and a list of found devices is returned to the caller. Using built-in functions from Android, the descriptor for each device is fetched and data of interest is extracted.

mDNS/DNS-SD Services

DNS-SD is another mechanism for advertising and discovering services on the network using DNS mechanisms [33]. Using mDNS this mechanism works in the local network without any additional (DNS) server. Like with UPnP, devices need to implement it so that they can be found using this method. When developing for the Android platform, support for this type of service advertising and discovering is already built into the standard API by Google [97]. In my application, the built-in functions are used to test devices for the presence of a specific service. The code used is shown in listing 4.32.

```
1  fun checkMdnsService(context: Context, service: String, ip:
     String, timeout: Long = 1000): Boolean {
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
2   val nsdManager = context.getSystemService(Context.NSD_SERVICE)
        as NsdManager
3   val addr = InetAddress.getByName(ip)
4   val myThread = Thread.currentThread()
5   var found = false
6   val resolveListener = object : NsdManager.ResolveListener {
7    ...
8    override fun onServiceResolved(serviceInfo: NsdServiceInfo) {
9     if (serviceInfo.host.equals(addr)) {
10     found = true
11     myThread.interrupt()
12    }
13   }
14  }
15  val discoveryListener = object : NsdManager.DiscoveryListener
        {
16   override fun onDiscoveryStarted(regType: String) {}
17   override fun onServiceFound(service: NsdServiceInfo) {
18    nsdManager.resolveService(service, resolveListener)
19   }
20   ...
21  }
22  nsdManager.discoverServices(service, NsdManager.
       PROTOCOL_DNS_SD, discoveryListener)
23  try {
24   Thread.sleep(timeout)
25  } catch (_: InterruptedException) {}
26  nsdManager.stopServiceDiscovery(discoveryListener)
27  return found
28 }
```

Listing 4.32: Service testing using DNS-SD over mDNS.

The "NsdManager" does not search for devices, but for services that have the specified name. If a matching service is found, its location is checked against the specified IP address. When the found one and the target address actually match, the search is stopped, and true is returned. Since not all devices reply equally fast, a timeout can be set. After the timeout the search is also stopped, but with a false result. It is necessary to use this procedure, as Android's API does not allow for browsing network services.

4.4.4 Bluetooth Device Classification

Android uses its own Bluetooth stack [98], which substantially differs from BlueZ [99], the default in many Linux distributions. Therefore, tools available for desktop Linux systems

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

do not work on Android. To use Bluetooth on an Android device, the provided API needs to be used. Both, Bluetooth Classic and Low Energy are supported by Android's Bluetooth stack.

Bluetooth Classic

As mentioned before, the classification method used for my Android application is based on Herfurt and Muliner's work [60]. Details about the modified procedure can be seen in section 4.3. This section is about the Android implementation of the described procedure. In the application, Bluetooth Classic is scanned first. There is no particular reason to prioritize one standard over the other, but it is not possible to search for both device types at the same time.

Device Detection Before any classification can take place, devices need to be detected first. Here the application is limited to - in Bluetooth terms - discoverable devices. To start the discovery on Android, a filter indicating the interesting "Intents" has to be created first. Another important part is the receiver, which handles the incoming intents. With the filter and the receiver ready, the combination of these can be registered to actually receive intents. The last step to actually discover devices is to use the "BluetoothManager" to launch device discovery. The code used to do the described steps is given in listing 4.33.

```
1 val filter = IntentFilter(BluetoothDevice.ACTION_FOUND)
2 filter.addAction(BluetoothDevice.ACTION_UUID)
3 filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED)
4 registerReceiver(classicReceiver, filter)
5 val bluetoothManager: BluetoothManager = getSystemService(
      BluetoothManager::class.java)
6 bluetoothManager.adapter?.startDiscovery()
```

Listing 4.33: Registering receiver for receiving intents.

The receiver has been omitted here, as its content will be explained below, together with other steps of the fingerprinting procedure.

Service Discovery After a device has been found, service discovery can be started. A discovered device will be reported to the receiver declared and registered above. As is shown in listing 4.34, the receiver directly starts the service discovery for the found device.

```
1 override fun onReceive(context: Context, intent: Intent) {
2  when (intent.action) {
3   BluetoothDevice.ACTION_FOUND -> {
4    val device: BluetoothDevice? =
5    intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE)
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
6       device?.fetchUuidsWithSdp()
7     }
8     ...
9   }
10 }
```

Listing 4.34: Fetch services on device discovery.

Only the part of the code dealing with found devices is listed, as it is the only substantial part for starting service discovery. Another intent will be received when service discovery is complete. Details about the Service Discovery Protocol(SDP) can be found in section 2.5.1.

Calculating Fingerprint When the next intent is received, all information needed to calculate the fingerprint has been gathered. The code can be seen in listing 4.35. The function "onReceive" is the same as in listing 4.34, but shows another section.

```
1  override fun onReceive(context: Context, intent: Intent) {
2   when (intent.action) {
3     ...
4     BluetoothDevice.ACTION_UUID -> {
5      val device: BluetoothDevice? =
6      intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE)
7      val uuids: Array<Parcelable>? =
8      intent.getParcelableArrayExtra(BluetoothDevice.EXTRA_UUID)
9      if (device != null && uuids != null) {
10      val hash = generateHash(uuids)
11      ...
12     }
13    }
14    ...
15  }
16 }
```

Listing 4.35: Calculate fingerprint for Bluetooth device.

The "device" variable contains information about the current remote device, including the MAC address. In the variable "uuids" are - as the name suggests - the UUIDs of the advertised services. This is all the information needed to calculate the fingerprint, as described in section 4.3.1.

Bluetooth Low Energy

After the scanning of classic devices has been completed, the low-energy scan can be started. Again, details of the underlying protocols can be found in section 2.5.2 in the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

background chapter of this work. This section aims to describe the Android implementation details and the problems that occurred on the path to a working implementation.

Device Detection Scanning for low-energy devices has many similarities with classic Bluetooth scanning, but there are some noteworthy differences. First, the scan does not stop automatically after a while. This means it has to be ensured that scanning is disabled after a specified time since the search would never end otherwise. Second, devices can appear more than once during a scan, depending on the interval of their advertisements. The code used to control the scan is given in listing 4.36.

```
1 val handler = Handler(Looper.getMainLooper())
2 val bluetoothManager: BluetoothManager = getSystemService(
     BluetoothManager::class.java)
3 val scanner = bluetoothManager.adapter?.bluetoothLeScanner
4 handler.postDelayed({
5   scanner?.stopScan(leReceiver)
6   Thread.sleep(500)
7   scanLeServices()
8 }, SCAN_PERIOD)
9 scanner?.startScan(leReceiver)
```

Listing 4.36: Bluetooth LE device detection.

As with Bluetooth Classic, the Bluetooth manager is used to get a scanner, while a receiver object handles detected devices. A handler is used to stop the scanning after a specified period of time. In my opinion, the Android Bluetooth Low Energy API is sometimes unstable and unreliable. This causes many errors, which will be partially described below. Here, problems with the Bluetooth LE API result in 500ms sleep after the discovery has been completed. During my testing, this was necessary in order to avoid errors during communication with the remote devices later on. The receiver stores found devices in a list and checks beforehand if a found device is already present in that list.

Service Discovery After all available devices have been discovered, they are searched for the "Device Information Service". This service - if present - provides human-readable information about the device type and its software version. As mentioned before, the Low Energy API of Android often causes problems:

- Generic Errors: When trying to communicate with a Bluetooth Low Energy device, often the error 133 is being returned. This is a generic error and the only way to solve it is to retry the operation. It has to be ensured that there are no infinite retries, as these might cause an infinite loop.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- Many Callbacks: Every operation results in a callback, even if the request is launched from a separate thread already. This makes it difficult to handle responses, as the application state has to be shared with the callback.
- Single Operations: Only a single operation per remote device can be started at once. Trying to do multiple ones will result in errors.

To aid with these problems the library "BleGattCoroutines" [100] is being used. It automatically takes care of retries and does not require the use of callbacks. This makes the resulting code, as seen in listing 4.37, much easier to read and write.

```
val settings = GattConnection.ConnectionSettings(false, false,
    true, BluetoothDevice.TRANSPORT_LE, BluetoothDevice.
    PHY_LE_1M)
for (device in leDeviceList) {
 val connection = GattConnection(device, settings)
 try {
  withTimeout(5000) {
   connection.connect()
  }
 } catch (e: TimeoutCancellationException) {
  connection.close()
  continue
 }
 val services: List<BluetoothGattService>
 try {
  withTimeout(5000) {
   services = connection.discoverServices()
  }
 } catch (e: TimeoutCancellationException) {
  connection.close()
  continue
 }
 ...
}
```

Listing 4.37: Service discovery for Bluetooth LE.

When defining the connection settings, it is important to set the transport method to LE explicitly. Otherwise, devices supporting both (Classic and LE) modes will fail to connect. The functions the library provides, do not exit automatically after a specified timeout. This can cause infinite waits on errors. Therefore, the "withTimeout" function shall be used to abort the Bluetooth communication when the timeout is reached. After connecting to the GATT server on the remote device and fetching the advertised services, it is possible to interact with the discovered services.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Fetch Characteristics In this work, characteristics are only read. After finding the - previously mentioned - device information service, the goal is to acquire more information about the device under investigation. This is done by fetching the characteristics of interest. Again, communication with the Bluetooth Low Energy device is aided by the "BleGattCoroutines" [100] library. The code used is provided in listing 4.38.

```
1  for (characteristic in service.characteristics) {
2   val value : String
3   try {
4    withTimeout(5000) {
5     connection.readCharacteristic(characteristic)
6     value = String(characteristic.value)
7    }
8   } catch (e : TimeoutCancellationException) {
9    connection.close()
10   continue@deviceLoop
11  }
12  when (characteristic.uuid.cutTo16BitAsLong()) {
13   0x2a29L -> { // Manufacturer Name
14    bleDevice.vendor = value
15   }
16   ...
17  }
18 }
```

Listing 4.38: Reading characteristics with Bluetooth LE.

Here it shall be assumed, that the service variable contains the device information service. As before, for the reading of characteristics, the timeout has to be handled externally of the used library. As mentioned in section 2.5.2, some characteristics have officially assigned UUIDs. Therefore, based on the UUID of the fetched characteristic, the meaning of the contained value can be determined. The gathered information can then, without any further modification, be displayed to a user.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 5

Results

This chapter examines the results achieved by the methods described in chapter 4. Here, the found results will be presented and explained. Later, in chapter 6, these results will be discussed and interpreted.

5.1 Found Properties - IP-based Devices

During this work, many devices have been analyzed to find properties, usable to identify them within a network. In this section, selected devices will be presented and it will be specifically explained, how the detection process works for each of them. If available for a specific device, also the gathering process for additional information will be demonstrated. This section only covers manually analyzed devices, as there are no deviating procedures for automatically generated fingerprints.

5.1.1 Samsung M262x 282x Printer

Samsung M262x 282x Printer is not a single printer model, but a series of devices with different capabilities. When the Android application encounters a device that has a MAC address starting with "30:CD:A7" (Samsung Electronics), the Samsung device pool is used for device classification. At the time of writing, this pool only contains a single step, which is used to detect the previously mentioned class of printers.

This type of printer advertises its presence using UPnP. In their device descriptors, these devices have set their "friendlyName" to "Samsung M262x 282x Series". This is a very good indicator for a device belonging to the suspected series. The code used to test for the explained characteristic can be seen in listing 5.1.

```
1 val service = treeStrategyCoordinator.getUPnPDeviceFromIP(
     device.ip) ?: return null
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
2 if (getUPnPDescriptionField(service, "friendlyName").startsWith
     ("Samsung M262x 282x Series")) {
3  val updatedDevice = SamsungM262x282xPrinter(device.ip)
4  return Pair(updatedDevice, nextScanPool)
5 }
6 return null
```

Listing 5.1: Test for a Samsung M262x 282x printer.

When the test succeeds, an object representing a device with a higher detection level is returned to the application controller. As described in section 4.2.1, the next scanning pool will be started to continue with the device classification, if it is not empty. To the best of my knowledge, the printer only reveals the model family, but not the concrete model over the network.

As already mentioned in section 4.1.1, a web service called SyncThru is running on this type of printer. To the best of my knowledge, there is no public documentation available for this software. Through manual analysis, I was able to get machine-readable information, provided in the JSON format, from this web service. Since the content header of the HTTP response is not set correctly due to a bug, Retrofit [95] can not be used for fetching and automatic deserialization. Therefore, two internal components of Retrofit, OkHttp [101] for receiving data over HTTP and GSON [102] for deserialization, need to be used manually. The used code is shown in listing 5.2.

```
1 val url = URL("http://$ip/sws/app/information/supplies/supplies
     .json")
2 val client = OkHttpClient()
3 val request: Request = Request.Builder().url(url).build()
4 val call = client.newCall(request)
5 val response = call.execute()
6 val text = response.body()?.string()
7 val gson = Gson()
8 val data = gson.fromJson(text, SamsungSuppliesResponse::class.
     java)
```

Listing 5.2: Fetching data from Samsung printer.

After fetching the data as plain text, it is transformed into an object of the type "SamsungSuppliesResponse". This class defines which properties are read from the retrieved text and therefore defines the names and datatypes of the fields. The data structure definition is shown in listing 5.3.

```
1 data class SamsungSuppliesResponse(
2  val toner_black : SamsungSupply,
3  val drum_black : SamsungSupply
4 )
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
5 data class SamsungSupply(
6  val remaining : Int
7 )
```

Listing 5.3: Defintion of data structure for Samsung printers.

Not all retrieved information is mapped to data fields since it is enough to fetch a small number of values for demonstration purposes. Also, the printer I used for testing only supported monochromatic printing, therefore many provided values had no useful meaning.

5.1.2 VU+ Satellite Receivers

Another interesting type of device are the VU+ satellite receivers [103]. For my testing, I used two different models: the "VU+ Solo" and "VU+ Zero", both with the VTi firmware installed. These receivers are based on Linux and offer a variety of network services. Among them are:

- Telnet
- Samba (Network file storage and NetBIOS)
- Web interface (with machine-readable API)

To check for this device class, the NetBIOS names are used. The code used to do so is listed in listing 5.1.2.

```
1 val names = getNetBIOSNames(device.ip)
2 if (names != null && names.contains("VUPLUS")) {
3  val model = names.first().lowercase().substring(2)
4  .replaceFirstChar { if (it.isLowerCase()) it.titlecase(Locale.
     ROOT) else it.toString() }
5  val vuplusDevice = VuplusDevice(device.ip, model)
6  return Pair(vuplusDevice, nextScanPool)
7 }
8 return null
```

These satellite receivers advertise multiple names over NetBIOS. One of them is "VUPLUS", the vendor name. If this name is found, the concrete model can be determined by looking at the first name advertised, as it will contain the model name, prefixed with "VU". As mentioned before, also a web interface is exposed. It is called "OpenWebif" and not only exposes a graphical frontend, but also a well-documented machine interface [104]. In this work, the web interface is used to fetch the currently active TV station and - more important for this thesis - the installed firmware version.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
val retrofit = RetrofitHelper.getInstance("http://$ip/")
val api = retrofit.create(VuplusVTIApi::class.java)
val data = api.getAbout().execute().body()

interface VuplusVTIApi {
 @GET("/api/about")
  fun getAbout(): Call<VuplusVTIAboutResponse>
}
```

Listing 5.4: Getting data from VU+ receiver.

Additionally, it is possible to fetch a screenshot of the current HDMI output of the satellite receiver. This screenshot is also served by the previously mentioned web interface. This possibility was used to demonstrate the versatility of Android's user interface. The code used is shown in listing 5.5.

```
val screenshot = ImageView(context)
Picasso.get().load("http://$ip/grab?format=jpg&mode=all&r=720")
.memoryPolicy(MemoryPolicy.NO_CACHE, MemoryPolicy.NO_STORE).
    into(screenshot);
screenshot.layoutParams = ViewGroup.LayoutParams(MATCH_PARENT,
    720)
layout.addView(screenshot)
```

Listing 5.5: Fetching screenshot from remote device.

To aid with the application development, a library called Picasso [105] has been used. Among other features, Picasso can load images from HTTP servers and place them in ImageView objects from Android. Screenshots from the Android application can be found in section 5.3.1.

5.1.3 Sony PlayStation 4 (Pro)

The PlayStation 4 is a device that is difficult to detect. Presumably because of its intended use case, it does not expose many services to the network. Also, the installed firmware [106] is in an uncommon (probably proprietary) format, unreadable by any standard tool. Therefore it is out of scope for this work to analyze the firmware for this device. A port scan using Nmap ends with results as seen in listing 5.6.

```
PORT STATE SERVICE
9295/tcp open armcenterhttps
41800/tcp open unknown
```

Listing 5.6: Port scan of a PlayStation 4 Pro using Nmap.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Banner grabbing on these two ports leads to a generic or no result. Nevertheless, the TCP service on port 41800 is advertised as "_spotify-connect._tcp" using DNS-SD over mDNS. The collected information can be transformed into code as seen in listing 5.7.

```
1 if (isTcpPortReachable(device.ip, 9295) &&
2  isTcpPortReachable(device.ip, 41800) &&
3  checkMdnsService(ctx, "_spotify-connect._tcp", device.ip)) {
4   val ps4Device = PS4Device(device.ip)
5   return Pair(ps4Device, nextScanPool)
6 }
7 return null
```

Listing 5.7: Testing for a PlayStation 4.

Running applications might expose different ports, or send out identifiable information (as shown before in listing 4.12). Nevertheless, this behavior has to be triggered by user interaction and is therefore not suitable for automatic device classification. Having the MAC address as first step for classifying devices prevents devices from other vendors to be wrongly classified as PlayStation 4. Nevertheless, I have not found a unique indicator to classify the PS4, nor a way to distinguish the different sub-models.

5.2 Fingerprints - Bluetooth Devices

As with IP-based network devices, many Bluetooth devices have been used for testing throughout the writing of this work. In this section, selected fingerprints of the tested devices will be compared and analyzed. For easier comparison, the devices will be sorted by their vendors. Also, having the MAC address as a major distinguishing feature between devices favors this type of sorting. Details about the fingerprinting algorithm can be read in section 4.3.1. The beginning of this section deals with Bluetooth Classic only. The results are presented in the following format:

- Device name
  - First 24 bits of the MAC address → Registered vendor name
  - Calculated hash value

Sennheiser Multiple wireless headphones from the brand Sennheiser have been tested. The gathered results are shown below:

- Momentum True Wireless 2
  - 00:1B:66 → Sennheiser electronic GmbH & Co. KG
  - 0f1d271bd068db5597abfaf8b993f0350e040b02ba0a258e5ab677bb120e4c4d

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- Momentum True Wireless 3
  - 80:C3:BA → Sennheiser Consumer Audio GmbH
  - 05f4a45cb6cdc90302f47b1c745381ee382cde0e34614e55ae8b7c1885a11edc
- CX 400BT True Wireless
  - 00:1B:66 → Sennheiser electronic GmbH & Co. KG
  - 0f1d271bd068db5597abfaf8b993f0350e040b02ba0a258e5ab677bb120e4c4d
- CX Plus True Wireless
  - 00:1B:66 → Sennheiser electronic GmbH & Co. KG
  - e2eabb150b9e9c9fa7aeaa5fc4b5552b7c7207fe4e9aa6e32de766bdc6a8513e

The Momentum True Wireless 3 headphones are the newest model among them and have a different MAC address characteristic than the other tested devices. Additionally the registered vendor name differs, therefore MTW3 headphones are never compared to the others during a classification run. Looking at the other fingerprints, the previous generation, the Momentum True Wireless 2 headphones can not be distinguished from the CX 400BT. They share the first 24 bit of the MAC address and also the same hash value, resulting in indistinguishable fingerprints. Albeit sharing the first 24 bit of the MAC address, the CX Plus True Wireless headphones have a different hash value compared to the other devices.

JBL For additional testing, JBL devices have been chosen. Instead of headphones, mostly portable speakers have been investigated. Again, the gathered results are shown below:

- Flip 3
  - F8:DF:15 → Sunitec Enterprise Co.,Ltd
  - B8:69:C2 → Sunitec Enterprise Co., Ltd.
  - 665eb72b41ef564d9b63dbd01cb120551878b77c3a6e50face7a98eed61321b0
- Xtreme
  - F8:DF:15 → Sunitec Enterprise Co.,Ltd
  - 665eb72b41ef564d9b63dbd01cb120551878b77c3a6e50face7a98eed61321b0
- Charge 5
  - F8:5C:7E → Shenzhen Honesty Electronics Co.,Ltd.
  - beb950fe47694d5968a9fd0e5525be8e8b23de462d3c391d77a815fdf25af162

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- Wave 100TWS
  - 6C:47:60 → Sunitec Enterprise Co.,Ltd
  - 8fb1d54c7e6974d7df62f219360b843d10bd9a27f9d8d4f8f199e4452604e3e8

The JBL Flip 3 has two entries for the vendor, because I tested two devices that are optically identical, but have a different vendor prefix set on their MAC address. The circumstance, that multiple devices of a single series - the two tested JBL Flip 3 portable speakers - do advertise different vendor prefixes on their MAC addresses, makes clear why not only these first 24 bis of the MAC address are compared, but also the registered short name of the vendor. Nevertheless, the fingerprinting and classification algorithm can not distinguish between the JBL Flip 3 and the JBL Xtreme speakers. Both, the first 24 bits of the MAC address and the hash value match. The remaining two devices, JBL Charge 5 and JBL Wave 100TWS, both have - for this test set - unique hash values and are therefore distinguishable by the approach used in this work.

5.2.1 Bluetooth Low Energy - Device Information Service

As explained in section 4.3.2, for Bluetooth Low Energy there is no fingerprinting algorithm used. Instead, the device information service is used if available, to fetch information about the device and firmware installed. In this section, the tested devices will be listed and it will be stated if they expose the desired service. The results are shown in table 5.1. Devices that do not support Bluetooth Low Energy will are not listed here, also not all devices used for Bluetooth Classic testing have been available throughout the whole time of the creation of this thesis. Therefore it is possible, that a device used before, supports Bluetooth Low Energy, but is not listed here. Most of

Device Name	**Device Information Service**
Philips Hue Iris	✓
Pine64 PineTime	✓
JBL Charge 5	×
Sennheiser Momentum True Wireless 3	✓
Sennheiser CX Plus True Wireless	✓

Table 5.1: Tested Bluetooth Low Energy devices.

the tested devices (all except the JBL Charge 5) exposed the device information service. On the PineTime smartwatch, the service was accessible all the time, without any way to turn it off. Also, both Sennheiser headphones still advertised their presence over Bluetooth Low Energy a few minutes after they have been enclosed in their case.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

5.3 Android Application

Building an Android application is a main part of this work. With this Android application, the found approaches are tested and evaluated. This section is about the capabilities and limitations of the built Android application.

5.3.1 Visual Appearance

A central part of many Android applications is their user interface. In this section, the interface of the application, developed for this thesis, will be presented and explained. The icons used in this application are not included with Android, but are taken from Flaticon [107].

Main Page

The main page is shown on application startup. From there the two main features can be reached: IP-based scanning and Bluetooth scanning. A screenshot of the main page is shown in figure 5.1. The two buttons lead to the respective activities, conducting

Figure 5.1: Android application main page.

the appropriate scans. The back and home buttons work as expected and close the application. Before starting the IP-based scan (WiFi scan), the application checks for an active WiFi connection. It is not possible to start the new activity when no connection is found.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

WiFi Scanning Page

IP-based scanning is the most important part of this thesis. Therefore, the WiFi scanning page is also the most important page of this Android application. Here the scanning results will be displayed in a summarized form, as is shown in figure 5.2. Each card

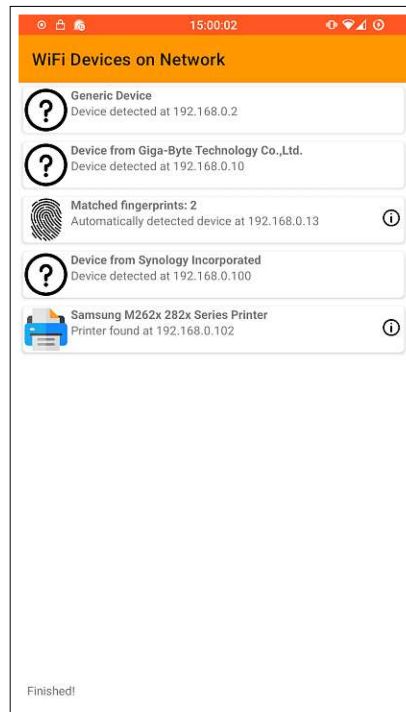


Figure 5.2: Android application IP-based scanning.

displayed represents a scanning result. During the scanning process, the displayed cards can change. The following explanations refer to the results of a fully conducted scan. Below, the possible outcomes will be explained:

Generic Device If the presence of a device has been detected, but no vendor name is known for the detected MAC address, the device will be displayed as "Generic Device". This card corresponds to detection level 0.

Device from ... This card basically has the same meaning as the "Generic Device" card, but a vendor name has been found in the MAC vendor table.

Device type A scanning step found a match for an investigated device. The generic message in the upper line is replaced by the name of the device type. This indicates a detection level greater or equal to 2. Additionally, a custom symbol will be displayed on the left side of the card.

Fingerprint When a device card has a fingerprint symbol on the left side, a match has been found using the automatically generated fingerprints. When only one firmware

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

image matches, the name is shown in the upper line of the card. If multiple images match, the number of matching firmware images is displayed instead.

Information symbol The information symbol on the right side has different meanings, dependent on which type of card it is displayed on. If it is on a "Device type" card, it indicates that additional information about the device can be fetched. In this case, it also implies a detection level of 4. On a "Fingerprint" card, the symbol indicates that multiple firmware images match the device in question. In both cases, the card can be clicked to open the detail page. This page will be described below.

Additionally to the cards, on the lower end of the page, a status bar is displayed. Here, messages indicating the current progress of the scan can be read.

Details Page

As mentioned before, the detail page can provide additional information for single devices, or a list of matching firmware images for a fingerprint match. Examples of these views are shown in figure 5.3 for the detailed information about a single device and in figure 5.4 for the list of matching firmware images. At the top of both pages, the currently selected card is displayed. For the single device additional information view, the page body layout completely depends on custom code written per device. Figure 5.3 is just a single example of what this page might look like. When viewing the details page for an automatically generated fingerprint match, the additional information section will contain the names of the firmware images, that match the analyzed device.

Bluetooth Scan Page

The Bluetooth scanning page is built almost similar to the WiFi scanning page. The same card format is used to present the results, but no additional information page is used here. Also, the status bar at the lower end is again used to indicate the current scanning step. An example of such scanning results is shown in figure 5.5. As before, multiple scanning outcomes are possible:

Unknown device An unknown Bluetooth Classic device will have a question mark as its symbol. This means that no fingerprint matches the detected device.

Known Bluetooth device When a fingerprint matches a Bluetooth Classic device, the corresponding card gets marked with the Bluetooth logo instead. On the first line, the locally stored name from the fingerprint will be displayed, on the second line the transmitted display name is shown.

Bluetooth Low Energy device A card with the downwards pointing arrow indicates a found Bluetooth Low Energy device. Depending on the available information from the device information service, the gained data will be displayed on the card.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

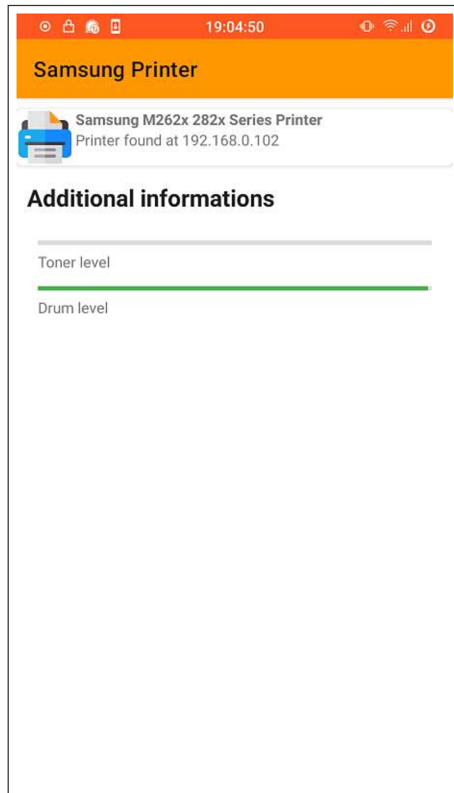


Figure 5.3: Detail page of a Samsung printer.

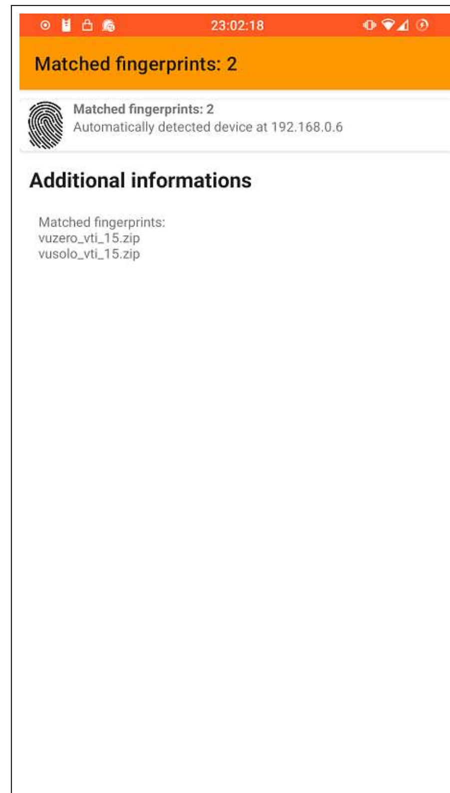


Figure 5.4: Matching firmware image information.

5.3.2 Limitations

Not all state-of-the-art approaches can be recreated using an Android application. This section is about the limitations of scanning for IoT devices using an unmodified Android device. As stated before, machine learning approaches were not evaluated in this work.

Passive Data Collection

In current literature, some researchers classify IoT devices by listening to transiting network traffic. The used properties for classification range from just the destination server, e.g. in Guo and Heidemann's work [57], to full deep packet inspection as seen in the work from Khandait et al. [59]. These approaches do not work on an unmodified Android phone. Capturing packets from the internal WiFi card is not possible with most integrated chipsets [108]. Even when external hardware is used, still a custom kernel and root access are needed. Also, using the internet connection sharing function of modern Android devices does not help. The own traffic can be captured using the Android VPN API [109]. Nevertheless, during my testing, connected devices did not use the established tunnel. Therefore, capturing packets from external devices does not work

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Figure 5.5: Android application Bluetooth scanning.

using unmodified Android only.

Raw Sockets

It is not possible to create raw sockets on unmodified Android [90], as special rights are needed to do so. This impacts various aspects of remote device scanning.

Sending and receiving ICMP messages only works very limited on Android. In section 4.4.2, it is shown that ICMP echo messages can be sent and the answers can be received. Also, when using UDP [92], the port unreachable message can be caught. To the best of my knowledge, no other ICMP interaction is possible with unmodified Android.

Not being able to create raw sockets also affects port scanning [36] and host discovery [29]. Many aspects of this work are inspired by Nmap [28]. But without privileged system access, many scanning approaches do not work. For host discovery, the ICMP ping scan approach can be used utilizing the integrated "ping" program or the Android network API. TCP ports can be scanned using the system's connect function. All other approaches need higher privileges. In section 2.2 and 2.4.1, the different scanning techniques have been explained. Alternative scanning techniques often provide better speed or reduce bandwidth consumption. Also, some scanning methods could be used to circumvent firewalls and therefore improve scanning results.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bluetooth Scanning

In section 2.5.1, it has been explained why it is not possible to listen to surrounding foreign Bluetooth communications with an off-the-shelf Bluetooth adapter. This limitation does not only affect Android, but all devices with similar hardware. Another set of restrictions is introduced by the custom Bluetooth software stack in Android [82]. How this influences the Android application for this work has been discussed in section 4.3.1.

5.4 Firmware Image Analysis Tool

To aid with the Android application's device recognition capabilities, the automatic firmware image analysis tool has been written. Implementation details can be found in section 4.1.2. In this section, the usage and outputs of the developed tool will be described.

Usage Before firmware analysis can be started, the firmware images under investigation need to be collected first. For analysis, all images need to be stored in a single folder, without any subdirectories. The firmware image analysis tool reads the folder path from the first command line argument. At the current state of development, it is not possible to add an already existing fingerprint results file for extension. All images, that should be recognized with the resulting output file have to be present in the input directory.

Output The resulting fingerprints will be written to a file called "fingerprints.json" in the current working directory. During the program run, information regarding the found properties will be printed on the console. An example of such an output for a single firmware image is shown in listing 5.8.

```
1  ***************************************************
2  Analyzing: netgear-RN4220S-root.tar
3  Found Avahi service at: ./etc/avahi/services/nut.service
4  -Service: _nut._tcp
5  -Port: 3493
6  Found Avahi service at: ./etc/avahi/services/frontview.service
7  -Service: _http._tcp
8  -Port: 80
9  Found smb config in: ./etc/default/config/etc/samba/smb.conf
10 Found hostname file in: ./etc/default/config/etc/hostname
11 -Hostname: readynasos
12 Found telnet banner at: ./etc/issue.net
13 ***************************************************
```

Listing 5.8: Output from firmware analysis tool.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Firmware images that lead to no result during analysis will not be present in the resulting fingerprint file. Any files or archives that do not have a supported file format will be skipped.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 6

Discussion

In this chapter, the approaches and results from this thesis will be discussed. This includes possible interpretations of the results and advantages or disadvantages from different tried approaches. Because of time and resource limitations, not all interesting aspects could be evaluated. Here I also will explain what those aspects were and how they affect this thesis.

6.1 Device Classification and Recognized Devices

Being just a normal network member has some disadvantages for device detection. To recognize a device, it has to expose services suitable for classification. If a device employs a firewall that blocks access from other devices, detection algorithms might not work. This applies to both, blocking of ICMP messages and restricting access to certain ports.

There are many benefits to being able to passively listen to device traffic. In my opinion, the most important one is being able to detect devices that do not expose any services or, as discussed above, block access to these services for certain devices. Another advantage is that passively listening to traffic will not produce any warnings in firewalls or intrusion detection systems. But there are also some downsides to just passively listening. When listening to a network's traffic, also traffic caused by users might be captured. This leads to some legal (data protection laws) and ethical questions. Also, there are some technical aspects to consider. Real-time processing of passing through traffic might need a lot of processing power depending on the traffic volume. Storing this traffic can also be problematic as more and more data gets collected.

Despite the downsides, active scanning from within the network also has some advantages. No traffic needs to be stored and processed in real-time. Therefore, there are fewer privacy considerations. Also, active scanning methods are usually faster, as it is not necessary to wait for the device under investigation to send messages through the network. Services

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

can be queried on demand and therefore might reveal information not obtainable using just passive listening. Furthermore, encryption of communication streams can hinder passive listening approaches.

Considering the advantages and disadvantages of the mentioned approaches, the utilized method has to fit the desired use case. With Android, passive approaches are almost unusable. Also, with default hardware and software, recording traffic is not possible. Additionally, Android devices are usually very resource-constrained in terms of available power. Therefore, active approaches are better suited for this kind of use case.

6.2 Fingerprinting Techniques

Throughout this thesis, three fingerprinting approaches have been described. In this section, these will be discussed regarding their advantages and disadvantages.

6.2.1 Manual Analysis

Manually analyzing devices is the most flexible way of finding identifiable information. Flexibility is the most important advantage of this approach. While automatic tools are limited to defined structures, manual analysis leaves the determining of the structure partially to human intuition. This can be an advantage, but also has some downsides at the same moment. Not adhering to the same procedure every time makes it possible to miss information, usually found using a structured approach. Human error is thereby a factor in the quality of the resulting fingerprints.

A major problem of the manual approach is that the end result is not a fingerprint in a machine-readable format, but the knowledge of how to presumably detect a device. This knowledge needs to be manually transformed into code to be able to use it. This procedure can be error-prone. Also, a considerable amount of time is needed per device for integration into the classification algorithm. This limits the usefulness when trying to classify a high number of devices.

Despite the downsides, manually analyzing devices is - for now - the only way of finding additional information sources, as this is highly device-specific. Also, devices using an uncommon operating system will rarely be supported by a firmware analysis program. Therefore, manual analysis might be necessary from time to time.

6.2.2 Automatic Generation

While trying to solve the scalability problem, an automatic approach has been developed. Properties often used during the manual analysis, are automatically read from a device's firmware image and stored combined in a single fingerprint file. During this analysis, no human interaction is needed. Therefore, it is theoretically possible to analyze a large number of devices, needing a fraction of the time needed when doing manual analysis.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

As mentioned before, at the current state of development, only Linux-based firmware images are supported. This design decision has been made, because Linux-based systems usually adhere to a common folder structure and use standardized and well-known software. These common properties make it easier to recognize patterns and obtain useful information. While other operating systems are also of interest, time constraints prevented me from supporting other systems as well. As a downside, this also limits the subset of compatible devices. Devices, which do not run on a Linux-based operating system, can not be analyzed. Therefore, very small devices, too resource-constrained to run Linux, have to be analyzed manually. This is also true for any other device running another - maybe custom - operating system.

The static nature of the analysis limits the amount of collectible information. No special device properties are taken into account. Nevertheless, depending on the firmware image, device type, concrete model or even the firmware version can be determined using the collected information. Also, the set of analyzed properties is consistent, leading to reliable and reproducible results.

6.2.3 Bluetooth

Herfurt and Mulliner's [60] approach for Bluetooth device fingerprinting can not be used on Android as is. The record handle, a key aspect of their fingerprinting strategy, is not exposed to software running on Android devices. For this thesis, I modified the algorithm to use the service UUIDs instead. Since these are - as described in section 2.5.1 - not always chosen freely, the ability to distinguish devices might suffer. Also, the approach has been published in 2004, when the number of different Bluetooth products was lower than today.

During my testing, the adapted algorithm was often not able to distinguish different devices belonging to the same device family. This can be an indication, that some devices use the same or at least similar firmware. Devices across different generations could usually be distinguished. A new generation of a device usually introduces new features, which need to be advertised using additional service UUIDs. Thereby, analyzing the newer device results in another hash value than the older one.

Another aspect, not covered in the original work from Herfurt and Mulliner, is that device vendors might release devices that are the same model, but have different first 24 bits in their MAC address. This problem has been coped for in my work by not only comparing MAC addresses, but also the registered vendor names. This solves the problem in some cases, but several vendors have multiple address blocks registered with a slightly differently pronounced name. Since the comparison is just based on equality of the two strings, the comparison fails in that case.

Bluetooth Low Energy

For this work, there is no fingerprinting strategy implemented for Bluetooth Low Energy devices. Instead, the wanted information is read directly from the device. This approach

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

does not need any fingerprint database or device analysis before being usable with a device. Therefore, it is not necessary to update the developed application when new devices are released, as they can be identified as long as they expose the desired service.

There are also some downsides to this approach. First, it can not be ensured by me or any other application developer, that the values returned by the devices are correct and useful. Also, not all devices have the device information service implemented and therefore can not be classified using this method.

Another interesting aspect of this method is the lack of data security. There is no password or other form of protection, leaving the data information service and often other services as well open to access from nearby third parties. If hardware vendors react to this threat, it might be possible that the approach used in this thesis does not work for future devices.

6.3 IP-based Scanning Approaches

To make use of the found properties, scanning strategies are needed. For this thesis, two have been implemented in the Android application. This section is about their properties, advantages and disadvantages.

6.3.1 Tree Based Approach

The tree-based approach offers high flexibility through custom-written code for each detection step. Utilizing the different detection levels also partial classifications are possible. For difficult-to-classify devices, determining only the device type might be the only possible option. On the other hand, for some devices much more information can be obtainable. For demonstration purposes, the details page in the Android application has been created for allowing as much flexibility as possible. Depending on the device type various data can be displayed there.

But there are not only advantages to this approach. As with the manual fingerprint generation, a lot of time-consuming manual work is needed to add a step to the detection tree. Maintenance will become more and more difficult as the tree grows. Also, bugs will be difficult to find, as there is more code to verify. Newly released devices will not be recognizable as long as no application update is being rolled out. The advantages and disadvantages of this approach are similar to those discussed above for the manual device analysis. This is partly because these two parts are working together closely for device classification.

Since Android 10 this approach is pretty much unusable, as the MAC address is used as the starting point for device analysis. Because of privacy reasons, the hardware addresses of the nearby network devices are hidden in modern Android versions. For this work, root access has been used to circumvent this restriction. When the application is developed further, the focus should be more on the second classification approach described below, as it fully works without any Android modifications.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

6.3.2 Classification Using Automatically Generated Fingerprints

The number of different IoT devices is growing constantly. To keep up with this development device analysis needs to be automated. These automatically generated fingerprints have a fixed set of possible properties, suitable for automated scanning. This approach has several advantages. The number of steps used for the classification process is known in advance. Also, there is only one procedure for every device. This helps with code maintenance and finding errors. No root access is needed for this approach, therefore it still can be used beyond Android 10.

Nevertheless, there are also some downsides. Partially the approach is limited by the fingerprint generation software. When no fingerprint can be found automatically, it will not be recognized by this method. While fingerprints can also be manually inserted into the database, this contradicts the design goals. Also, it will not be possible to classify all existing IoT devices based on the predefined feature set, that is checked when using this method.

The classification result, produced by the automatic approach, does not always contain only one detected device, but also a set can be returned. The reason is the limited possible feature set, which is predefined by the analyzed properties from the fingerprint creation tool. This design choice has been made to be able to classify more devices, at the cost of sometimes less meaningful results.

6.4 Future Work

Due to time and resource constraints, not all interesting aspects could be paid attention to in this thesis. This section is about possible future research that could extend this work and improve its capabilities.

In my work, machine learning was out of scope. Nevertheless, in the literature, some interesting approaches can be found for device classification using machine learning and artificial intelligence. Since Android devices are usually very constrained in terms of energy and processing power when compared to wall-powered machines, it has to be determined which of these approaches can be ported to Android. Thereby the new machine learning Android APIs [110] can help.

In my opinion, this work would benefit greatly from improvements at automatic fingerprint generation. Improvements in this area would make the classification more reliable and versatile, enabling the Android application to detect a greater number of devices. These improvements can be of various forms. Being able to analyze other operating systems than Linux opens the detection algorithm to a wider variety of devices. Also, some firmware images are encrypted or in an uncommon format. Future research could also deal with these kinds of problems.

Another part of this thesis, suffering from many restrictions, is Bluetooth scanning. In my testing, the fingerprinting algorithm for Bluetooth Classic did not always work reliably.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

This is one part where future research could take over. But also for Bluetooth Low Energy, some questions remain unanswered. Future research for Bluetooth Low Energy could focus on creating fingerprints when the device information service is not accessible. Also, Android's support for Low Energy devices has some severe issues. Improvements there could also be beneficial to many applications.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 7

Conclusion

In this work, I have researched how a specialized Android application can detect IoT devices in the nearby environment. The newly developed application has two main parts: IP-based network and Bluetooth scanning, while Bluetooth can be divided into the Classic and Low Energy variants. As an addition, a firmware analysis tool has been developed, capable of automatically scanning Linux-based firmware images.

IP-based scanning is mostly restricted to active methods. This makes it possible to detect and classify devices that have network services running. Limitations occur mostly due to software reasons, as Android only allows TCP and UDP connections. In many cases, this is still enough to accurately classify a remote device. Also, active scanning methods quickly lead to results. Further improvements would be most promising at the automatic fingerprint generation, where an improvement would benefit the accuracy and total count of detectable devices.

On the other hand, Bluetooth scanning is severely limited by hardware restrictions, as the built-in hardware of modern Android devices is not capable of listening to foreign surrounding traffic. Despite the advancements in research about this topic, capturing foreign traffic still requires special hardware. Nevertheless, device fingerprinting makes it possible to distinguish devices or device categories, without looking at their display names, as long as they are in discoverable mode. Future work might include improving that fingerprinting algorithm to be able to distinguish similar devices more reliably.

As Bluetooth Low Energy has a device information service standardized, analyzing LE devices often leads to good results for this work. As long as the GATT server on the device in question implements the service, classification works accurately and reliably. This mentioned service also exposes information about the currently installed firmware version of a device. One major advantage of this approach is that no fingerprint is needed, therefore also new devices work without any change to the application.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

2.1 OSI 7 Layer Model [6]. 4
2.2 MAC Structure [10]. 5
2.3 Network Classes [18]. 7
2.4 TCP Header [22]. 8
2.5 TCP Three-Way-Handshake [23]. 9
2.6 TCP Finishing Connection [24]. 10
2.7 UDP Header [22]. 10
2.8 Pinging host in local network on Linux. 12
2.9 SSDP Service Discovery Architecture [31]. 14
2.10 Full UPnP Search [32]. 15
2.11 Frequency Hopping over Time [44]. 21
2.12 Stored Data for each Service [45]. 22

4.1 Network Information from a PS4. 33
4.2 Network Information from Home Router. 34
4.3 Avahi Discovery in Home Network. 37
4.4 Wireshark Displaying Captured SSDP Packets. 39
4.5 Supplies Displayed in Printer's Web Interface. 41
4.6 Firefox Developer Tools. 42
4.7 Tree Based Scanning. 48
4.8 Blueprinting checksum calculation[60]. 51

5.1 Android application main page. 76
5.2 Android application IP-based scanning. 77
5.3 Detail page of a Samsung printer. 79
5.4 Matching firmware image information. 79
5.5 Android application Bluetooth scanning. 80

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Tables

4.1 Detection Levels 48
5.1 Tested Bluetooth Low Energy devices. 75

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

[1] *State of iot 2021: Number of connected iot devices growing 9% to 12.3 billion globally, cellular iot now surpassing 2 billion*, https://iot-analytics.com/number-connected-iot-devices/, Last accessed: 2022-03-16.

[2] *Chapter 15. nmap reference guide*, https://nmap.org/book/man.html, Last accessed: 2022-03-16.

[3] M. Antonakakis, T. April, M. Bailey, *et al.*, "Understanding the mirai botnet", in *26th USENIX Security Symposium (USENIX Security 17)*, Vancouver, BC: USENIX Association, Aug. 2017, pp. 1093–1110, ISBN: 978-1-931971-40-9. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis.

[4] L. L. Peterson and B. S. Davie, *Computer networks: a systems approach*. Elsevier, 2007.

[5] *Layers of osi model*, https://www.geeksforgeeks.org/layers-of-osi-model/, Last accessed: 2022-11-09.

[6] *What is the osi model*, https://www.cloudflare.com/learning/ddos/glossary/open-systems-interconnection-model-osi/, Last accessed: 2022-11-09.

[7] J. Day and H. Zimmermann, "The osi reference model", *Proceedings of the IEEE*, vol. 71, no. 12, pp. 1334–1340, 1983. DOI: 10.1109/PROC.1983.12775.

[8] *What is layer 4 of the osi model?*, https://www.a10networks.com/glossary/what-is-layer-4-of-the-osi-model/, Last accessed: 2022-11-09.

[9] *What is a mac address*, https://www.guru99.com/what-is-mac-address.html, Last accessed: 2022-11-10.

[10] *Introduction of mac address in computer network*, https://www.geeksforgeeks.org/introduction-of-mac-address-in-computer-network/, Last accessed: 2022-11-10.

[11] *What is a mac address*, https://whatismyipaddress.com/mac-address, Last accessed: 2022-11-10.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[12] *Organizationaly unique identifiers*, https://standards-oui.ieee.org/oui/oui.txt, Last accessed: 2022-11-10.

[13] *Understand tcp/ip addressing and subnetting basics*, https://learn.microsoft.com/en-us/troubleshoot/windows-client/networking/tcpip-addressing-and-subnetting, Last accessed: 2022-11-10.

[14] *Structure and types of ip address*, https://www.geeksforgeeks.org/structure-and-types-of-ip-address/, Last accessed: 2022-11-10.

[15] *Understanding cidr subnet mask notation*, https://docs.netgate.com/pfsense/en/latest/network/cidr.html, Last accessed: 2022-11-10.

[16] *Cidr - classless inter-domain routing*, https://www.elektronik-kompendium.de/sites/net/2011231.htm, Last accessed: 2022-11-11.

[17] *Ipv4-netzklassen*, https://www.elektronik-kompendium.de/sites/net/2011221.htm, Last accessed: 2022-11-11.

[18] *Introduction of classful ip addressing*, https://www.geeksforgeeks.org/introduction-of-classful-ip-addressing/, Last accessed: 2022-11-10.

[19] *Private address ranges*, https://www.ibm.com/docs/en/networkmanager/4.2.0?topic=translation-private-address-ranges, Last accessed: 2022-11-11.

[20] *What is a loopback address*, https://www.geeksforgeeks.org/what-is-a-loopback-address/, Last accessed: 2022-11-11.

[21] W. Eddy, "Rfc 9293 transmission control protocol (tcp)", 2022.

[22] *Tcp vs udp – what is the difference between tcp and udp*, https://www.softwaretestinghelp.com/tcp-vs-udp/, Last accessed: 2022-11-15.

[23] *Tcp flags*, https://www.keycdn.com/support/tcp-flags, Last accessed: 2022-11-16.

[24] *Tcp connection termination*, https://www.geeksforgeeks.org/tcp-connection-termination/, Last accessed: 2022-11-17.

[25] J. Postel, "User datagram protocol", Tech. Rep., 1980.

[26] J. Postel, "Internet control message protocol", Tech. Rep., 1981.

[27] *Udp scan*, https://nmap.org/book/scan-methods-udp-scan.html, Last accessed: 2022-09-15.

[28] *Nmap: The network mapper*, https://nmap.org/, Last accessed: 2022-09-15.

[29] *Host discovery techniques*, https://nmap.org/book/host-discovery-techniques.html, Last accessed: 2022-11-23.

[30] R. Droms, *Rfc2131: Dynamic host configuration protocol*, 1997.

[31] *Upnp standards & architecture*, https://openconnectivity.org/developer/specifications/upnp-resources/upnp/, Last accessed: 2022-11-24.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[32] *Exploring upnp with python*, https://www.electricmonk.nl/log/2016/07/05/exploring-upnp-with-python/, Last accessed: 2022-11-24.

[33] *Dns service discovery (dns-sd)*, http://www.dns-sd.org/, Last accessed: 2022-10-09.

[34] A. Gulbrandsen, P. Vixie, and L. Esibov, "A dns rr for specifying the location of services (dns srv)", Tech. Rep., 2000.

[35] S. Cheshire and M. Krochmal, "Rfc 6762: Multicast dns", *Internet Engineering Task Force (IETF)*, 2013.

[36] *Port scanning techniques*, https://nmap.org/book/man-port-scanning-techniques.html, Last accessed: 2022-09-28.

[37] *Tcp syn (stealth) scan (-ss)*, https://nmap.org/book/synscan.html, Last accessed: 2022-12-01.

[38] *Tcp connect scan (-st)*, https://nmap.org/book/scan-methods-connect-scan.html, Last accessed: 2022-12-01.

[39] *What is banner grabbing*, https://securitytrails.com/blog/banner-grabbing, Last accessed: 2022-12-02.

[40] *Issue.net(5) - linux man page*, https://linux.die.net/man/5/issue.net, Last accessed: 2022-09-15.

[41] *The difference between classic bluetooth and bluetooth low energy*, https://blog.nordicsemi.com/getconnected/the-difference-between-classic-bluetooth-and-bluetooth-low-energy, Last accessed: 2022-10-27.

[42] M. Cominelli, F. Gringoli, P. Patras, M. Lind, and G. Noubir, "Even black cats cannot stay hidden in the dark: Full-band de-anonymization of bluetooth classic devices", in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 534–548. DOI: 10.1109/SP40000.2020.00091.

[43] M. Chernyshev, C. Valli, and M. Johnstone, "Revisiting urban war nibbling: Mobile passive discovery of classic bluetooth devices using ubertooth one", *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 7, pp. 1625–1636, 2017. DOI: 10.1109/TIFS.2017.2678463.

[44] D. Spill and A. Bittau, "Bluesniff: Eve meets alice and bluetooth.", *WooT*, vol. 7, pp. 1–10, 2007.

[45] S. Bluetooth, "Bluetooth core specification v5.3", *Specification of the Bluetooth System*, p. 3085, 2021.

[46] *Find ble devices - android developers*, https://developer.android.com/guide/topics/connectivity/bluetooth/find-ble-devices, Last accessed: 2022-10-28.

[47] *Gatt*, https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt, Last accessed: 2022-12-11.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[48] *Specifications - assigned numbers*, https://www.bluetooth.com/specifications/assigned-numbers/, Last accessed: 2022-10-28.

[49] A. Sivanathan, H. H. Gharakheili, and V. Sivaraman, "Can we classify an iot device using tcp port scan?", in *2018 IEEE International Conference on Information and Automation for Sustainability (ICIAfS)*, 2018, pp. 1–4. DOI: 10.1109/ICIAFS.2018.8913346.

[50] D. Kumar, K. Shen, B. Case, *et al.*, "All things considered: An analysis of {iot} devices on home networks", in *28th USENIX security symposium (USENIX Security 19)*, 2019, pp. 1169–1185.

[51] P. Bajpai, A. K. Sood, and R. J. Enbody, "The art of mapping iot devices in networks", *Network Security*, vol. 2018, no. 4, pp. 8–15, 2018, ISSN: 1353-4858. DOI: https://doi.org/10.1016/S1353-4858(18)30033-3. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1353485818300333.

[52] S. Agarwal, P. Oser, and S. Lueders, "Detecting iot devices and how they put large heterogeneous networks at security risk", *Sensors*, vol. 19, no. 19, p. 4107, Sep. 2019, ISSN: 1424-8220. DOI: 10.3390/s19194107. [Online]. Available: http://dx.doi.org/10.3390/s19194107.

[53] K. K. Knabl, *Design, implementation and evaluation of a mobile security scanner app for smart home users*, eng, 2020. [Online]. Available: https://resolver.obvsg.at/urn:nbn:at:at-ubl:1-35509.

[54] A. Amro, "Iot vulnerability scanning: A state of the art", in *Computer Security*, S. Katsikas, F. Cuppens, N. Cuppens, *et al.*, Eds., Cham: Springer International Publishing, 2020, pp. 84–99, ISBN: 978-3-030-64330-0.

[55] *Shodan search engine*, https://www.shodan.io/, Last accessed: 2022-12-16.

[56] H. Al-Alami, A. Hadi, and H. Al-Bahadili, "Vulnerability scanning of iot devices in jordan using shodan", in *2017 2nd International Conference on the Applications of Information Technology in Developing Renewable Energy Processes & Systems (IT-DREPS)*, 2017, pp. 1–6. DOI: 10.1109/IT-DREPS.2017.8277814.

[57] H. Guo and J. Heidemann, "Ip-based iot device detection", in *Proceedings of the 2018 Workshop on IoT Security and Privacy*, Budapest, Hungary: Association for Computing Machinery, 2018, pp. 36–42, ISBN: 9781450359054. DOI: 10.1145/3229565.3229572. [Online]. Available: https://doi.org/10.1145/3229565.3229572.

[58] F. Le, S. Calo, and D. Verma, "Risks and challenges of training classifiers for iot", in *Internet of Things – ICIOT 2021*, B. Tekinerdogan, Y. Wang, and L.-J. Zhang, Eds., Cham: Springer International Publishing, 2022, pp. 15–28.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[59] P. Khandait, N. Hubballi, and B. Mazumdar, “Iothunter: Iot network traffic classification using device specific keywords”, *IET Networks*, vol. 10, no. 2, pp. 59–75, 2021. DOI: https://doi.org/10.1049/ntw2.12007. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/ntw2.12007. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/ntw2.12007.

[60] M. Herfurt and C. Mulliner, “Remote device identification based on bluetooth fingerprinting techniques”, *Trifinite Group, White Paper*, 2004.

[61] J. Dunning, “Taming the blue beast: A survey of bluetooth based threats”, *IEEE Security & Privacy*, vol. 8, no. 2, pp. 20–27, 2010. DOI: 10.1109/MSP.2010.3.

[62] G. Celosia and M. Cunche, “Fingerprinting bluetooth-low-energy devices based on the generic attribute profile”, London, United Kingdom: Association for Computing Machinery, 2019, pp. 24–31, ISBN: 9781450368384. DOI: 10.1145/3338507.3358617. [Online]. Available: https://doi.org/10.1145/3338507.3358617.

[63] G. Celosia and M. Cunche, “Saving private addresses: An analysis of privacy issues in the bluetooth-low-energy advertising mechanism”, in *Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, ser. MobiQuitous ’19, Houston, Texas, USA: Association for Computing Machinery, 2020, pp. 444–453, ISBN: 9781450372831. DOI: 10.1145/3360774.3360777. [Online]. Available: https://doi.org/10.1145/3360774.3360777.

[64] A. Barua, M. A. Al Alamin, M. S. Hossain, and E. Hossain, “Security and privacy threats for bluetooth low energy in iot and wearable devices: A comprehensive survey”, *IEEE Open Journal of the Communications Society*, vol. 3, pp. 251–281, 2022. DOI: 10.1109/OJCOMS.2022.3149732.

[65] S. Alexander, R. Droms, D. Options, and B. V. Extensions, *Ietf rfc 2132*, 1997.

[66] *Service name and transport protocol port number registry*, https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml, Last accessed: 2022-12-26.

[67] *Avahi-browse(1) - linux man page*, https://linux.die.net/man/1/avahi-browse, Last accessed: 2022-12-27.

[68] *Welcome to avahi*, https://www.avahi.org, Last accessed: 2022-09-15.

[69] *Gupnp - gnome wiki*, https://wiki.gnome.org/Projects/GUPnP, Last accessed: 2022-12-29.

[70] *Wireshark - go deep.* https://www.wireshark.org/, Last accessed: 2022-12-31.

[71] *Universal media server*, https://www.universalmediaserver.com/, Last accessed: 2022-12-31.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[72] *Netbios - network basic input/output system*, https://www.elektronik-kompendium.de/sites/net/0907221.htm, Last accessed: 2023-01-03.

[73] N. W. Group *et al.*, "Protocol standard for a netbios service on a tcp/udp transport: Concepts and methods", *IETF RFC 1001*, 1987.

[74] *Nbtstat*, https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/nbtstat, Last accessed: 2022-09-19.

[75] *Smb.conf*, https://www.samba.org/samba/docs/current/man-html/smb.conf.5.html, Last accessed: 2023-01-03.

[76] *Samba - opening windows to a wider world*, https://www.samba.org/, Last accessed: 2022-09-06.

[77] *Python-libarchive*, https://pypi.org/project/python-libarchive/, Last accessed: 2022-09-19.

[78] *Filesystem hierarchy standard*, https://refspecs.linuxfoundation.org/FHS_3.0/fhs/index.html, Last accessed: 2022-09-19.

[79] *Dhcp-options(5) - linux man page*, https://linux.die.net/man/5/dhcp-options, Last accessed: 2022-09-06.

[80] *What is reverse dns and how does it work?*, https://phoenixnap.com/kb/reverse-dns-lookup, Last accessed: 2022-09-06.

[81] *Upnp technical basics: Upnp device architecture (uda)*, http://upnp.org/resources/documents/UPnP_UDA_tutorial_July2014.pdf, Last accessed: 2022-09-06.

[82] *Bluetoothdevice - android developers*, https://developer.android.com/reference/android/bluetooth/BluetoothDevice, Last accessed: 2022-10-27.

[83] *Einführung in bluetooth generisches attributprofil (gatt)*, https://www.bluetooth.com/de/bluetooth-resources/intro-to-bluetooth-gap-gatt/, Last accessed: 2022-10-28.

[84] *Device information service 1.1*, https://www.bluetooth.com/specifications/specs/device-information-service-1-1/, Last accessed: 2022-10-28.

[85] *Oneplus 5*, https://www.oneplus.com/global/5, Last accessed: 2023-01-10.

[86] *Lineageos*, https://lineageos.org/, Last accessed: 2023-01-10.

[87] *Apache commons net*, https://commons.apache.org/proper/commons-net/, Last accessed: 2023-01-10.

[88] *Inetaddress*, https://developer.android.com/reference/java/net/InetAddress, Last accessed: 2023-01-11.

[89] *Restriction on access to /proc/net filesystem*, https://developer.android.com/about/versions/10/privacy/changes#proc-net-filesystem, Last accessed: 2022-09-21.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[90] *Raw(7) — linux manual page*, https://man7.org/linux/man-pages/man7/raw.7.html, Last accessed: 2022-09-28.

[91] *Socket*, https://developer.android.com/reference/java/net/Socket, Last accessed: 2023-01-12.

[92] *Datagramsocket*, https://developer.android.com/reference/java/net/DatagramSocket, Last accessed: 2023-01-14.

[93] *Datagrampacket*, https://developer.android.com/reference/java/net/DatagramPacket, Last accessed: 2023-01-14.

[94] *Connect to the network*, https://developer.android.com/training/basics/network-ops/connecting, Last accessed: 2022-09-28.

[95] *Retrofit - a type-safe http client for android and java*, https://square.github.io/retrofit/, Last accessed: 2023-01-15.

[96] *Ssdp-client*, https://github.com/resourcepool/ssdp-client, Last accessed: 2022-10-08.

[97] *Use network service discovery*, https://developer.android.com/training/connect-devices-wirelessly/nsd, Last accessed: 2022-10-09.

[98] *Bluetooth - android open source project*, https://source.android.com/docs/core/connect/bluetooth, Last accessed: 2023-01-17.

[99] *Bluez - official linux bluetooth protocol stack*, http://www.bluez.org/, Last accessed: 2023-01-17.

[100] *Blegattcoroutines*, https://github.com/Beepiz/BleGattCoroutines, Last accessed: 2023-01-18.

[101] *Overview - okhttp*, https://square.github.io/okhttp/, Last accessed: 2023-01-22.

[102] *Github - google/gson*, https://github.com/google/gson, Last accessed: 2023-01-22.

[103] *Home - vuplus*, https://www.vuplus.de/, Last accessed: 2023-01-23.

[104] *Openwebif api documentation*, https://github.com/E2OpenPlugins/e2openplugin-OpenWebif/wiki/OpenWebif-API-documentation, Last accessed: 2023-01-23.

[105] *Picasso*, https://square.github.io/picasso/, Last accessed: 2023-01-23.

[106] *So aktualisierst du die systemsoftware auf einer ps4-konsole*, https://www.playstation.com/de-de/support/hardware/ps4/system-software/, Last accessed: 2023-01-22.

[107] *Vector icons and stickers - png, svg, eps, psd and css*, https://www.flaticon.com/, Last accessed: 2023-01-25.

[108] *Wireless cards and nethunter*, https://www.kali.org/docs/nethunter/wireless-cards/, Last accessed: 2023-01-26.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[109] *Vpn - android developers*, https://developer.android.com/guide/topics/connectivity/vpn, Last accessed: 2023-01-26.

[110] *Machine learning - android developers*, https://developer.android.com/ml, Last accessed: 2023-01-30.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub