Curious Apps: Large-scale Detection of Apps Scanning Your Local Network

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Paul Theodor Hager, BSc
Matrikelnummer 01426941

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Assistant Prof. Dipl.-Ing. Dr.in techn. Martina Lindorfer, BSc
Mitwirkung: Dipl.-Ing. David Schmidt, BSc

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.

Wien, 2. Dezember 2022

Paul Theodor Hager | Martina Lindorfer

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.

Curious Apps: Large-scale Detection of Apps Scanning Your Local Network

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Paul Theodor Hager, BSc
Registration Number 01426941

to the Faculty of Informatics

at the TU Wien

Advisor: Assistant Prof. Dipl.-Ing. Dr.in techn. Martina Lindorfer, BSc
Assistance: Dipl.-Ing. David Schmidt, BSc

Vienna, 2nd December, 2022

Paul Theodor Hager   Martina Lindorfer

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

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

Paul Theodor Hager, 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, 2. Dezember 2022

Paul Theodor Hager

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

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.

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

Acknowledgements

First of all, I would like to thank Dr. Martina Lindorfer and Dipl.-Ing. David Schmidt for supervising and supporting me while writing this thesis. Also, I would like to thank my colleagues and friends for the continued discussion on this thesis and their support by proofreading. A special thanks is going to my parents for making everything possible in the first place.

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.

Kurzfassung

In unserer vernetzten Welt werden Datenlecks und fragwürdige Profiling-Techniken immer relevanter: Nicht nur auf Websites, sondern auch auf Smartphones.
Viele verschiedene Datenpunkte werden verwendet, um Nutzerprofile für beispielweise Werbung zu erstellen. Diese Datenpunkte enthalten oft sensible Informationen und können, wenn sie nicht sorgfältig verwendet und gespeichert werden, publik werden. In unserer Arbeit suchen wir nach Android Anwendungen (Apps), die das lokale Netzwerk (LAN) scannen und diese Datenpunkte möglicherweise für Profiling verwenden.

Wir fassen zusammen, wie Android Apps im LAN nach anderen Geräten suchen können und, wie diese Techniken in einer Android App implementiert werden können. Auf der Grundlage dieser Recherche haben wir über 40 handgefertigte Yara-Regeln entwickelt, um Android-Apps mit LAN-Scan-Fähigkeiten abzustimmen. Basierend auf diesen Yara-Regeln haben wir ein hybrides Analyse-Framework entwickelt, um Android Apps zu finden, die LAN-Scans ohne Benutzerinteraktion durchführen. Das von uns vorgeschlagene Analyse-Framework besteht aus drei Teilen: Zunächst verwenden wir über 40 handgefertigte Yara-Regeln, um Android Apps vorzufiltern, die Daten/Funktionen im Zusammenhang mit LAN-Scanning enthalten. Im zweiten Schritt führen wir die vorgefilterten Android Apps dynamisch auf einem echten Android-Smartphone aus und erfassen den Netzwerkverkehr. Zuletzt analysieren wir die Netzwerk-Dumps auf LAN-Scan Aktivitäten. Wir haben unser hybrides Analyse-Framework mit 3 verschiedenen Android-App Datensätzen (Top 1000 General Purpose, 1.259 IoT/Companion-Apps, 117 Malware-Apps) ausgeführt, insgesamt mit über 2.300 verschiedenen Android Apps. Acht Android Apps führen ohne Benutzerinteraktion ARP-Scans im LAN durch und 29 Android Apps verschicken SSDP-Suchanfragen, ebenfalls ohne Benutzerinteraktion. Auf der Grundlage unserer Ergebnisse erstellen wir Casestudies, um die Ursache der LAN-Scan Aktivitäten zu ergründen. Mit unserem Ansatz finden wir eine Android-App, bei der Profiling den einzigen ersichtlichen Grund für das durchgeführte LAN-Scanning darstellt.

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

Privacy leaks and shady fingerprinting techniques are becoming more and more relevant in our connected world: Not only on websites but also smartphones.
All kinds of different data points are used to create user profiles for fingerprinting and advertisement. These data points often contain sensitive information and, if not carefully used and stored, may be leaked. In our work we search for Android applications (apps) which are scanning the local area network (LAN) and potentially are using these datapoints for fingerprinting or advertisement.

We summarize how LAN scanning in Android apps can be done technique-wise (what kind of LAN scanning techniques exist?) and implementation-wise (how can these LAN scanning techniques be implemented?). Based on this research, we developed over 40 handcrafted Yara rules to match Android apps with LAN scanning capabilities. We use this Yara ruleset in our hybrid analysis framework to find apps that are LAN scanning without user interaction. Our proposed framework consists of three parts: First, we use our Yara ruleset to pre-filter Android apps which contain data/functions related to LAN scanning. Next, we dynamically run the pre-filtered Android apps on a real Android smartphone and capture the network traffic. In the last step, we analyze the network dumps for LAN scanning activities. We run our hybrid analysis framework with 3 different Android app datasets (Top 1,000 General Purpose, 1,259 IoT/companion apps, 117 malware apps), totaling over 2,300 different Android apps. We found 8 Android apps ARP scanning the LAN and 29 Android apps sending SSDP search requests without user interaction. Based on our findings we conduct case studies to research why the found Android apps are doing this. We found at least one Android app where we feel certain the LAN scanning happens for fingerprinting reasons.

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

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.

Contents

Kurzfassung ix

Abstract xi

Contents xiii

1 Introduction 1
1.1 Problem Statement and Motivation . . . 1
1.2 Methodology . . . 2
1.3 Structure . . . 3

2 Related Work 5

3 Background 9
3.1 Android Platform . . . 9
3.2 Network Scanning . . . 11
3.3 Static Analysis . . . 15
3.4 Dynamic Analysis . . . 18

4 Design and Implementation 21
4.1 Overview . . . 21
4.2 App Pre-Filtering . . . 21
4.3 Static Analysis . . . 22
4.4 Dynamic Analysis . . . 23

5 Results 27
5.1 Yara Ruleset . . . 27
5.2 App Dataset . . . 29
5.3 Analysis Results . . . 30
5.4 Case Studies . . . 32
5.5 Traffic Characterization . . . 42
5.6 Discussion of Results . . . 44

6 Discussion 53

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

6.1 Limitations 53
6.2 Future Work 54

7 Conclusion **57**

A Appendix **59**
A.1 Yara Rules 59
A.2 Hashes 66

List of Figures **69**

List of Tables **71**

List of Algorithms **71**

Bibliography **73**

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 Bibliothek Your knowledge hub WIEN

CHAPTER 1

Introduction

1.1 Problem Statement and Motivation

The scanning of networks has always been an indicator of malicious activities. Many events in the past, like the famous Mirai botnet,[1] which compromised thousands of Internet of Things (IoT) devices, or the devastating WannaCry attacks [34], a cryptoworm abusing the EternalBlue vulnerability [32], both scanning for new victims, lead to this judgment. Attackers use network/device scanning mostly to gather more information for future operations. MITRE ATT&CK [60, 61],"a globally-accessible knowledge base of adversary tactics and techniques based on real-world observations" [40], even has its own tactic for scanning: *Reconnaissance - TA0043*, with the technique: *Active Scanning - T1595*. This is of interest because researchers found that everyday websites, which are normally considered harmless, are scanning the local network too: They are tracking, scanning and possibly even attacking the local network [1, 33, 36, 50].
As the usage of smartphones and apps is growing [59] and more apps are added to the app markets (e.g., Google Play Store [58] and Apple App Store [56]) every day, this begs the question, if apps on our smartphones are doing the same. Since iOS 14, apps that want to interact with devices on the local network have to ask for permissions [8]. Recently there were some public discussions about apps that are asking for permissions to do local network scanning.[2] People found out that apps like Signal, PayPal, Spotify and 1Password are requesting the LAN scanning permission.[3] It is not immediately clear why these apps are doing this. For some apps, such activities might be self-explanatory, e.g., apps that are communicating with smart devices (e.g., smart TV, smart light bulb),

[1] https://www.bbc.com/news/technology-42342221, last accessed: 20.10.2022
[2] "I don't know what fresh hell this is but abso-fucking-lutey not, Instagram.", Zack Whittaker, https://twitter.com/zackwhittaker/status/1461545972711018498, last accessed: 20.10.2022
[3] "Some of the more interesting ones on my phone.", damien, https://twitter.com/__damiend__/status/1461552940213325826, last accessed: 20.10.2022

but not for others. For example, based on public reportings,[4] the Among Us![5] iOS app seems to sniff/scan the local area network (LAN) via Universal Plug and Play (UPnP) and Simple Service Discovery Protocol (SSDP) without proper permission from the user. As apps can be used to attack the local network and the IoT devices running there, as shown by Sivaraman et al. [54], this raises the question: Why are they scanning the local network?
Therefore we explore local network scanning activities of smartphone apps. Based on the research of Kuchhal et al. [36], the goal of this thesis is to:

- Survey methods and approaches of LAN scanning from mobile apps.
- Answer the following research questions:
  - RQ1: Find apps that conduct local network scanning activities.
  - RQ2: Investigate why those apps perform local network scans.
  - RQ3: Study if the apps scan without user interaction/consent.

1.2 Methodology

The methodological approach consists of the following parts:

1. **Literature and Approach Research:**
   In the first step we perform a scientific literature research. The literature research covers the current state of the art and related frameworks, tools and approaches in the research community. Further, we consider related sources, like blog posts, to get a better understanding of how a local network scan could be executed. This step helps us to answer the question how LAN scanning can be done from mobile apps.

2. **Designing a (Large-scale) Hybrid Analysis Testbed:**
   Based on the findings of the previous step we design a hybrid (e.g., static and dynamic) analysis testbed for the Android platform. We handcraft a set of Yara rules that detect LAN scanning capabilities by reversing apps which are known to have such capabilities (e.g., Fing [39]). Further, we create a dynamic analysis system to find concrete apps; apps which are conducting LAN scanning activities without user interaction. The resulting testbed is able to detect Android apps which are scanning the local network.

[4] "AmongUs - The Real Imposter, sniffing on the internal network. Apparently #AmongUs on iOS is sniffing on the Lan devices with uPnP / SSDP Making a Joke out of @Apple's #App #Privacy #Policy", Chilik Tamir, https://twitter.com/_coreDump/status/1369273264674111488, last accessed: 20.10.2022

[5] https://apps.apple.com/us/app/among-us/id1351168404, last accessed: 20.10.2022

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

3. **Large-scale Hybrid Analysis of Android Apps:**
   In this step we execute a large-scale hybrid analysis of a set of Android apps. We choose the Android platform in favor of the iOS platform because of the wide spectrum of already established analysis and reversing tools of APK files. This step helps us in answering the question on how many apps are performing LAN scanning activities.

4. **Empirical Evaluation:**
   Based on the results of the large-scale analysis and casestudies of a few selected apps, using both, quantitative and qualitative methods, we evaluate why apps are scanning the local network and if they do it without user consent.

1.3 Structure

The remaining thesis is structured as follows: Chapter 2 gives a brief summary of related work and ongoing work in this field of research. In Chapter 3 we discuss the background and fundamentals needed for this thesis. Chapter 4 shows the details of our approach on how we answer the research questions and how we implemented it. Further, in Chapter 5 we present our findings and answer the research questions. Following, in Chapter 6 we discuss some further miscellaneous findings, limitations, and ideas for future work. Lastly, we summarize and conclude our work in Chapter 7.

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 2

Related Work

In this chapter we give a brief overview of ongoing research on relevant topics: local network scanning and attacks, and dynamic, static, and network analysis of mobile apps.

Local Network Scanning And Attacks. Kuchhal et al. [36] performed a large-scale empirical investigation if and how popular websites are scanning their users' local networks. The authors used the Tranco top 100,000 domains as well as additional 145,000 known malicious sites for their research. In both groups, they found multiple (>100) sites which are scanning the local network. They identified that 40% of the top 100,000 sites were scanning their users' local network for host profiling, purportedly for fraud and bot detection. Additional causes were: native application communication and developer errors. The authors did not uncover explicit user tracking. Malicious websites were not found to attack the local network but rather, the authors inferred that the malicious sites scanning the local network are compromised or cloned phishing websites and that the scanning results from the corresponding original site. Further, they found that many tested sites scanned their users' local networks based on their operating system: they found significantly more local scanning activity for the Windows operating system. Their work establishes basic empirical knowledge on the scanning of localhost and LAN hosts from real-world websites. Our work looks at local network scanning originating from smartphone apps instead of websites. We expect more LAN scanning to originate from mobile devices compared to websites, as there are more legitimate use cases for it (e.g., companion apps for IoT devices).

Sivaraman et al. [54] demonstrated in their paper how the local network could be attacked via forged smartphone apps. Without knowledge of the user such apps would scan the network for vulnerable IoT devices, report them to an external entity, modify the firewall to allow external calls and, ultimately, attack the IoT device. The authors show that home routers are poorly protected against such attacks and the need of increased security for IoT devices. In this thesis, we try to detect such malicious 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.

Farrah et al. [25] show in their paper how various zero conf protocols can be attacked with man in the middle (MitM) attacks. Zero conf protocols (e.g., multicast DNS (MDNS) or DNS-based Service Discovery (DNS-SD)) are used to provide a plug and play method to setup internal networks. The authors ran over 300 experiments to test their implementations and to demonstrate the feasibility and the severity of MITM attacks on zeroconf protocols. Further, they provided Zeek code for detecting these attacks in network traffic. Our work covers the abuse of SSDP (see Section 3.2.1), a zero conf protocol, for LAN scanning purposes.

Acar et al. [1] presented two web-based attacks against local IoT devices that any malicious website or third-party script could perform. In their attack scenario, a victim visits the attackers website, which contains malicious code to communicate with local IoT devices via HTTP. They showed that the malicious code can circumvent the same-origin policy by exploiting error messages on the HTML5 MediaError interface or by carrying out DNS rebinding attacks. Further, the authors concluded that attackers could gather sensitive information from devices, track and profile the owners to serve ads, or control IoT devices. Some potential countermeasures were proposed as well. Our work looks into how prevalent local network scanning attacks/activities are in smartphone apps.

Static Analysis. Zhao et al. [71] contributed an automatic approach to detect hidden behaviors in mobile apps via input validation tracking. Their static approach, INPUTSCOPE, detects the execution context of the input validation and also the content involved in the validation. This enables them to expose secrets of interest. The authors tested their tool with over 150,000 apps and found 12,706 apps to have backdoor secrets (e.g., master password) and 4,028 apps to have blacklist secrets (e.g., censorship keywords). Our work shows an automatic approach to find potential LAN scanning apps.

Wu et al. [68] presented an open-port analysis pipeline. They developed a crowdsourcing app and a server-side analytic engine to continuously monitor open ports on smartphones. The authors published the app on Google Play for 10 months and collected over 40 million port monitoring records from 3,293 users in 136 countries worldwide. With this approach, the authors were able to observe 925 open ports in popular apps and 725 open ports in built-in system apps. For further analysis, the authors developed a static analysis tool: OPTool. They developed their tool specifically for the open-port diagnosis and uses a backward slicing graph (BSG) to simultaneously track multiple parameters. Out of 1,027 apps that are known to have open ports, OPTool flagged 671 apps with potential Java open-port constructions and 98 apps with native open-port constructions. The remaining apps are opening ports via dynamic code loading or are heavily obfuscated which renders them impossible to find for the OPTool. We use a static approach to pre-filter Android apps for LAN scanning activities.

Brengel et al. [18] proposed Yarix, a method to efficiently find files matching arbitrary Yara rules. For scaling purposes, Yarix creates an inverted n-gram index that maps fixed-length byte sequences to files in which they appear. To match the indexed files with Yara rules, Yarix transforms these Yara rules into index lookups to find a set of potentially matching files. Afterward, the Yara rules are applied to the found files to find

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 Bibliothek WIEN Your knowledge hub

hard matches. This approach speeds up the matching on big file datasets: The authors tested their approach on 32M malware samples with 1,404 Yara rules and found that the index requires just 74% of the disk space required for the actual files. Also, they show that their querying system is five orders of magnitude faster than using the standard sequential YARA tool. In our work we use Yara to find potential LAN scanning apps.

Dynamic Analysis and Network Analysis. As we use dynamic and network approaches, we give a brief overview of those topics as well: Wen et al. [66] proposed CANHUNTER, an automatic system for reverse engineering CAN bus commands using just car companion mobile apps. Their system uses backward slicing and dynamic forced execution, and semantic recovery using UI component correlation and function argument association to uncover CAN bus commands. The authors tested CANHUNTER on 236 car companion apps for the Android and iOS platform, showing CANHUNTER can uncover 182,619 unique CAN bus commands of 360 car models of 21 car manufacturers. Further, they recovered the semantics of 86.1 % of those CAN bus commands and validated over 70 % for syntactics and semantics through public resources, cross validation, as well as real car tests. In our work, we use static approaches to find local network scanning activities in smartphone apps.

Further, Ren et al. [48] presented a system that improves visibility and control of personally identifiable information (PII) leaks in network traffic of mobile devices: ReCon. ReCon reveals PII leaks and gives users control over them without requiring any special privileges or custom mobile operating systems. The proposed system uses machine learning to reveal potential PII leaks by monitoring network traffic. By visualizations the system empowers the user with the ability to control these leaks via blocking or substitution of found PII. The authors evaluated ReCon's effectiveness with experiments using leaks from the 100 most popular iOS, Android, and Windows Phone apps and via an institutional review board (IRB) approved user study with 92 participants.

Leveraged by their previous work, Ren et al. [47] provided the first longitudinal study of the privacy impact of using popular Android apps and their updated versions over time. They found the following trends regarding the sharing of PII with other parties: overall privacy tends to worsen across app versions, the types of gathered PII change across versions, HTTPS adoption is slow for apps, third parties not only track users pervasively, but also gather sufficient information to know what apps a user interacts with, when they do so, and where they are located when they do. The authors published a online tool to help users to make informed decisions whether to update an app given a set of changes in a new version. Further, they recommend the usage of ReCon [48], Lumen [44] and AntMonitor [38] to block unwanted privacy leaks from newer app versions. Instead of network monitoring, our work uses a static analysis approach to find local network scanning activities from smartphone apps.

Pradeep et al. [43] collected 424 Android browser apps from various different apps stores and analyzed these apps for personally identifiable information (PII) leaks. They build a novel, open sourced testbed for controlled, repeatable experiments on real webpages that allows to detect browsers modificating webpages and browser exfiltrating PII. Their

results show that a wide range of privacy-enhancing (e.g., blocking known user-tracking services) but also privacy-harming behavior: 32% of the browsers leak at least one type of PII and 81 browsers are sharing the browser history, for 37 of which the authors were not able to find a feature which would require such behavior. The authors argue that 13 of these 37 apps, send unique user identifiers alongside the browser history and thus can be classified as apps that are tracking its users. This behavior is harmful to the users privacy as their browsing history can be linked back. Our works uses a similar testbed to find LAN scanning Android apps.

Reardon et al. [45] searched for Android apps that circumvent the Android permission system. They ran hundreds of thousands of Android apps in their instrumented environment and monitored their runtime and network behavior. The authors looked for evidence of side and covert channels being used to access data that the respective Android app had no permission to access. Upon finding such apps, the authors reversed the apps to see how the apps accessed the unauthorized data. Further, they used software fingerprint to measure the prevalence of the found techniques among other apps. Using the mentioned analysis the authors found several side and covert channels to access sensitive data (e.g., geolocation) which are actively used by many popular apps. Reardon et al. do not cover apps which are LAN scanning as, in contrast to iOS (see Section 3.1.1), there is no permission needed to access the LAN.

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 3

Background

In this chapter we discuss the background and fundamentals needed for this thesis: We give a brief overview of the Android platform, discuss possible network scanning attacks and how they could be implemented on the Android platform, and summarize static and dynamic analysis approaches.

3.1 Android Platform

Android is the most used mobile operating system [57]. It is based on the Linux kernel and is developed as open-source, mostly by Google.[1] A driving factor for the success of the Android mobile operating system is that it is very easy to develop and share/sell Android apps - applications running on the Android platform. Developers can publish their apps via the Google Play Store, from where users can install them. Apps are mostly written in Java or Kotlin, if performance is critical, C/C++ can be used as well and are packaged as "Android Package", or short APK, with the file extension ".APK". APK files are essential ZIP files containing all the necessary files the operating system needs to execute the app. The Java/Koltin code is compiled into Dex bytecode, which is then executed by the Dalvik Virtual Machine ("just-in-time") on an Android smartphone. Since Android 5.0, the Android Runtime (ART) replaces the Dalvik VM and uses the Dex bytecode to further compile ("ahead-of-time" at installation) it into native instructions. Both runtimes are compatible [52]. To make Dex bytecode more human readable, Smali [35] was developed. Smali/baksmali is an assembler/disassembler for the Dex bytecode and is able to approximate the Dex bytecode to the initial source code of the app.
Android uses Linux user-based protection to isolate apps from each other - the Application Sandbox [4]. Each app gets a unique user ID (UID) and is executed in its own process.

[1] https://www.android.com/, last accessed: 20.10.2022

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

Thereby apps can not access files and memory from other apps and are also restricted in accessing system resources (e.g., camera, GPS), see Figure 3.1. As the Application Sandbox is implemented in the kernel, everything above the kernel level runs sandboxed. This allows developers to choose their tooling/programming language as they like, tough Java and Kotlin are advised. Java and Koltin are executed in the Dalvik virtual machine.

3.1.1 Android Permissions

Some Android system APIs (e.g., GPS location, network state and access) are protected via "Android Permissions". Android uses a permission system to protect access to sensitive data and/or sensitive actions. The Android platform knows two groups of permissions: install-time permissions and runtime permissions (also known as dangerous permissions). Pre Android 6.0 there was only the install-time permission system in place. This meant that by installing an app, all the permissions declared in the Android manifest file were granted. To provide a more secure and understandable system, Android 6.0 added the runtime permission system: Runtime permissions give your app the ability to access more restricted/sensitive data and/or actions. See Figure 3.1 for an overview of the Android app-isolation architecture. APIs are only accessible via the operating system and if an app wants to use an restricted API, it has to define the needed capabilities in the "AndroidManifest.xml" file (contained in the .APK file). This file contains all the capabilities of an app. On triggering such an ("dangerous") API via an app the user gets prompted to deny or allow this API access/permission. Once granted, the user is not prompted again to deny or allow specific permissions. For example, if an app would like to use the GPS location of its host smartphone, it needs to define permissions as seen in Listing 3.1 in the "AndroidManifest.xml" file. Therefore apps now have to ask the user during runtime for these specific/sensitive permissions to be granted. For example, consider an app that wants to listen to some kind of IP multicast protocol (e.g., SSDP 3.2.1) and thus has to enter the Wi-Fi multicast mode (to receive multicast packets). Such an app has to declare the *android.permission.CHANGE_WIFI_MULTICAST_STATE* (install-time) permission in its Android manifest file, see Listing 3.1 for an example. Now assume that the mentioned app also wants to access the user's calendar. This means that it also has to declare the *android.permission.READ_CALENDAR* permission in its Android manifest file. But in this case, the *READ_CALENDAR* permission is considered as a "dangerous" permission, as it potentially contains very sensitive data, and thus is treated as runtime permission and the users are asked during execution if they want to grant this permission. In Android 10 the runtime permission system was further enhanced by giving the user the ability to not only allow or disallow some permission during runtime but also to grant permission only while the app is in use. Further, already granted runtime permissions can now be changed in the Android settings [6, 7, 43].

iOS Permissions

Apple iOS has a similar permission system as Android. Compared to the Android permission system however, Apple iOS 14 added new "local network privacy controls" [8]

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 <manifest...>
2     <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"
        />
3     <uses-permission android:name="android.permission.
        CHANGE_WIFI_MULTICAST_STATE" />
4 </manifest>
```

Listing 3.1: Android Manifest declaring permissions for changing WiFi state and to enable multicast mode.

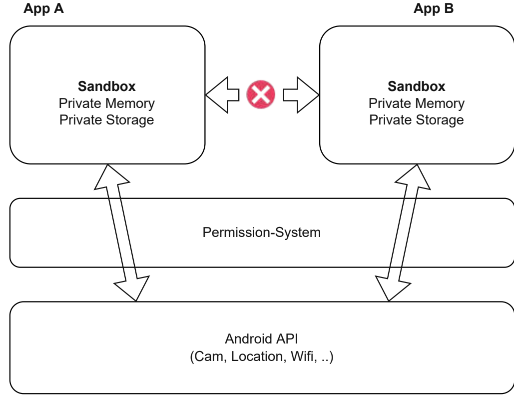

Figure 3.1: Architecture of Android app isolation.

- a permission the user has to give an app during runtime if the app wants to access the local network. The Android system is missing such permission and we think it is needed.

3.2 Network Scanning

Network scanning (network reconnaissance) techniques are used to gain information about networks and their respective devices. This information includes the count of active devices, open services per device, OS information per device, device/service versions, and some more. An adversary may use this information to attack or, more interesting for the scope of this work, track and fingerprint a network or devices connected to a network [17, 36, 51]. In this thesis, we are only interested in network scanning techniques and methods related to LAN environments or localhost.

3.2.1 Scanning Methods

The purpose of this section is to summarize known (active host and port) scanning techniques and methods:

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

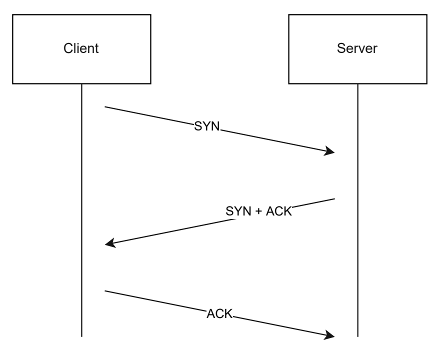

Figure 3.2: TCP Three-Way Handshake.

Internet Control Message Protocol (ICMP). The ICMP Ping scan [17, 51] is one of the simplest scanning methods which identifies active hosts in the network. It sends a ICMP ECHO request to a given IP and waits for a ICMP ECHO reply. This technique works with all major operating systems and an attacker could use it to quickly scan networks for active hosts.

Transmission Control Protocol (TCP). Some could argue that, based on the fact that many application layer protocols (e.g., HTTP, SSH, SMPT) are using the TCP protocol on top of the Internet Protocol (IP), TCP is one of the most used protocols. TCP provides ordered and reliable data transmission/streaming between client and server applications. To establish a connection between client and server the TCP 3-Way Handshake (see Figure 3.2) is used: First, the client sends a TCP packet with the SYN flag set. The server responds with a SYN+ACK TCP packet and the client responds with a TCP ACK packet. After these three packets, data can be transmitted. There are two usual active host/port scanning methods (besides trying to start a normal connection) based on the TCP protocol: TCP SYN scan and TCP SYN/ACK scan.

TCP SYN [17, 51] scan works by sending a TCP SYN packet ("abusing" the TCP 3-way handshake) to a given host-port combination. If the host is up and the used port is open, it replies with TCP SYN-ACK, otherwise, a TCP RST response is sent.

The *TCP SYN/ACK [17, 51]* scan works by sending a TCP SYN/ACK packet. If the host/port combination is up, it responds by sending a TCP RST packet.

Further TCP scanning techniques include:

IDLE Scan. [17, 51] This method needs another device on the network ("zombie") which an attacker could use for initiating the connections in place of the actual scanning host. This technique abuses the IPID field and involves IP spoofing to instrument the "zombie" host.

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

The *XMAS Scan [17, 51]* technique uses the PSH, URG and FIN flags of the TCP header. A closed port responds with a RST packet, open ports do not respond. It is called “XMAS Scan” because in Wireshark[2] such a packet looks like a XMAS-tree.

FIN Scan and NULL Scan [17, 51] are similar to the XMAS scan: The FIN scan only uses the FIN flag and the NULL scan does not set any flag.

User Datagram Protocol (UDP). UDP is, again on top of the IP protocol, one of the most used protocols, as many different application layer protocols are build on it (e.g., DNS) UDP does not provide ordered or reliable data transmission/streaming, rather UDP works like “fire & forget” and thus is much faster than TCP. A basic UDP scan does not need any special flags; it involves a normal UDP connection. Based on the UDP protocol [17, 51] there are various scanning techniques, e.g. SSDP/UPnP scanning, see Section 3.2.1.

Simple Service Discovery Protocol (SSDP). The SSDP networking protocol is a component of the Universal Plug and Play (UPnP) protocol, which allows devices to behave in a “plug and play”-manner. SSDP is used for advertisement and discovery of services in the network and is based on the UDP protocol on top of which HTTP requests are sent. UPnP devices listen on the standard multicast address (IP: 239.255.255.250, UDP Port: 1900) and respond on SSDP discovery (HTTP method “M-SEARCH”) messages.[3] An attacker can use the Simple Service Discovery Protocol (SSDP) to identify UPnP hosts on LAN-level. Compared to other scanning methods, SSDP scanning involves only one packet to be sent (to a special multicast IP address), which makes this method rather attractive. This allows scanning the LAN for UPnP devices [23, 29].

Address Resolution Protocol (ARP). ARP can be used to scan hosts on LAN-level [51]. ARP is used to translate IP addresses to MAC addresses and is needed for communication in LAN networks. It works by broadcasting an ARP packet with the meaning: “If your IP is a.b.c.d, please send me your MAC”. If there is an active host with IP a.b.c.d in the network, it responds to this request. This primitive allows scanning a full network.

3.2.2 Implementation Methods

We found three different main opportunities to implement the methods and techniques described in Section 3.2.1:

Dalvik VM or Android Runtime.
The probably most common way, is to use the default Android app tools: Java/Koltin that gets compiled into the Android app. For example, the open source Android app

[2] https://www.wireshark.org/, last accessed: 20.10.2022

[3] “AmongUs - The Real Imposter, sniffing on your internal network. Apparently #AmongUs on iOS is sniffing on your Lan devices with uPnP / SSDP Making a Joke out of @Apple’s #App #Privacy #Policy”, Chilik Tamir, https://twitter.com/_coreDump/status/1369273264674111488, last accessed: 20.10.2022

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

```
public void m7962d(int i, int i2, int i3) {
    boolean contains;
    for (int i4 = 0; i4 < i3 && m7964b(); i4++) {
        int i5 = (i2 * i3) + i + i4;
        boolean z = true;
        Ip4Address ip4Address = new Ip4Address(new byte[]{(byte) ((i5 >>
            24) & 255), (byte) ((i5 >> 16) & 255), (byte) ((i5 >> 8) &
            255), (byte) (i5 & 255)});
        synchronized (this.f21128f) {
            contains = this.f21127e.contains(ip4Address);
        }
        if (!contains) {
            synchronized (this.f21128f) {
                if (!ip4Address.equals(this.f21124b.m7231c()) && !
                    ip4Address.equals(this.f21124b.m7232b())) {
                    z = false;
                }
            }
            if (!z) {
                try {
                    if (ip4Address.m7237p().isReachable(
                        AGCServerException.f19186OK)) {
                        synchronized (this.f21128f) {
                            this.f21127e.add(ip4Address);
                        }
                    }
                } catch (IOException unused) {
                }
            }
        }
    }
}
```

Listing 3.2: Code of Fing Android app which is responsible for LAN Scanning.

Ning("Network-Scanner for Android") [30] uses this approach: The `isReachable()` function from the `InetAddress` class [5] is used[4] to (ping) scan all IP addresses from the network. Fing [39], the closed source version, does the same. See Listing 3.2 for the decompiled code: The for-loop in line 3 assembles the IP addresses. For each created IP address an, `Ip4Address` Object is instantiated in line 6 and is used in line 18 to check if the IP is reachable.

Native C/C++

This method involves writing the scanning functionality with C/C++. Android provides the Native Development Kit [3] (NDK) which allows the usage of C/C++ in Android apps. Upon compilation, the C/C++ code gets compiled to a native library which

[4] https://github.com/csicar/Ning/blob/master/app/src/main/java/de/csicar/ning/scanner/PingScanner.kt#L26, last accessed: 20.10.2022

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

then can be used through the Java Native Interface (JNI) in your Java/Kotlin code. This allows writing the scanning functionality in C/C++ and use it as a library within Java/Kotlin. C/C++ might be used for obfuscating/hiding functionality.

High Level
The scanning functionality can also be implemented in a high-level manner, e.g., interpreters and similar. An example for this would be JavaScript which gets executed in a browser/webview. We found an app which is written in Apache Cordova [21] where the scanning functionality is implemented in JavaScript, see our case study in Section 5.4.3.

3.2.3 Detection Possibilities

For the scope of this work we decided to detect ARP scanning, as this detection covers various different (all volume-based and IP protocol-based) scanning methods. We take advantage of a "convenient" side-effect of the ARP protocol: If a device inside a LAN network wants to connect (for the first time) to another device (e.g., to a printer), it (the network card to be precise) first has to send a ARP broadcast packet to get the MAC address. The same applies if a device scans the network with any IP based protocol (TCP, UDP, ICMP, ...). For every IP the scanning device wants to check, it has to send a ARP broadcast packet first. This behavior can easily be seen in traffic dumps. By detecting such behavior we catch practically all volume-based and IP protocol-based scanning.
Further, we decided to also detect broadcast-based scanning, like SSDP. These methods are rather easy to detect as it involves finding a single network packet with the respective characteristics.
ARP scanning can be detected by counting the ARP query packets from a single Host for multiple different IP addresses. SSDP scanning can be detected by looking for UDP packets with Dst-Port 1900 and Dst-IP 239.255.255.250 and with a HTTP payload containing the HTTP M-SEARCH * method.

3.3 Static Analysis

In the following section we discuss static analysis approaches and tools which are important for the scope of this work. Static Analysis means examining Android apps without executing them, e.g., just by decompiling and looking at source code or Dex code. First, we discuss a pattern matching approach (Yara) and Apktool and JD-GUI, tools to unpack/decompile APK files. Further, we describe data flow analysis and give a brief summary of tools using this approach.

3.3.1 Static Analysis Approaches

The following approaches are important for our 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.

```
rule example : some_tag {
    meta:
        example_text = "This is just an example"
        example_number = 3
        example_bool = true
    strings:
        $a = "foobar"               // string
        $b = {66 6f 6f 62 61 72}    // hex bytes
        $c = /foobar/               // regex
    condition:
        $a or $b or $c
}
```

Listing 3.3: Yara Rule Example.

Yara

Yara [63], the "The pattern matching swiss knife", is a tool which can be used for identification and classification tasks. Originally Yara was designed for quick malware classification but more use cases have emerged over time. It uses "rules" that are applied to data (file or memory) and tell you if a rule "matches". See Listing 3.3 for an example Yara rule.

Rules consist of three main parts: meta, strings and the condition. In the "meta" section additional information (e.g., a description of the rule) about the rule are stored. The "strings" section lets you define patterns. Those patterns can be in the form of normal strings, hex-bytes or regular expressions. In the last part, "condition", the condition for the rule to match, are defined. The rule above, named "example", would trigger if the given file/memory contains either "foobar", the hex-bytes "66 6f 6f 62 61 72" or the regular expression "foobar".
We use Yara to filter a subset of interesting apps from the initial set. Then we use the filtered subset in the next analysis step. In Section 4.2 we discuss the approach in detail.

Apktool

Apktool [67] is a tool for reverse engineering Android APK files. It allows to unpack Android APKs files and tries to disassemble dex files into Smali code. Further, Apktool is able to repack unpacked and modified Android apps. For the scope of this work, we need the unpack/disassemble features that are helpful for writing Yara rules.

JD-GUI

JD-GUI [24] is able to decompile the Smali bytecode back into Java and was used in combination with Apktool, to unpack/disassemble Android apps and to create Yara rules.

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

3.3.2 Data Flow Analysis

Data flow analysis tries to compute the different paths on which data "flows" through the application. One special version of data flow analysis is "taint analysis". Taint analysis "taints" (e.g., marks) interesting data or variables and tracks their flow path through an application [53]. Code parts that return or generate interesting data are called "sources" and code parts that consume interesting data are called "sinks" [53]. This type of analysis is of interest for us, because it does not need to run the actual mobile apps and thus is rather suitable for a large-scale analysis.

Existing Frameworks

ValueScope and LeakScope [53, 72]. LeakScope is a static analysis tool build on top of dexlib2 [35] and Soot [62] by Zuo et al. [72]. The authors build the tool to automatically detect data leakage vulnerabilities in Android apps that are using cloud APIs. It uses predefined sinks (e.g., methods that use Cloud APIs) to detect points of interest. To reconstruct possible inputs for predefined sinks, LeakScope transforms the Android Dex bytecode into a Call-Graph (CG) with Soot. Based on the CG a data dependency graph (DDG) is built: program instructions are represented as nodes and if there are "definition-usage" dependencies between two nodes, edges are added. Now the predefined sinks are searched in the DDG and, if found, it reconstructs all computation steps which are involved in reaching the sink. Simulating all computation steps results in possible input values for the sink in question. LeakScope is only able to reconstruct string values (e.g., API keys) [53, 72].

The shortcoming of only being able to reconstruct strings was tackled by Schmidt [53]: ValueScope. They extended the reconstruction to the following types: Integer, Strings, InetAddress, InetSocketAddress, URI, URL, okhttp3.HttpUrl and even arrays. Another shortcoming was fixed: To avoid running into an endless loop LeakScope does not analyze cyclic code-blocks, which leads to missing coverage. ValueScope solves this by marking already analyzed code-blocks and only analyzing unmarked code-blocks. Further improvements include: adding separate timeouts for the backtracking and the forward computation, and build related improvements like using Gradle[5] as build tool and dependency cleanup [53].

For the sake of completeness three other static analysis approaches are discussed as well: FlowDroid [14], BackDroid [69] and Amandroid (also known as Argus) [65].

FlowDroid [14]. FlowDroid is a static taint analysis for Android apps that is context, field, object, and flow-sensitive and considers the Android application lifecycle, callbacks, and UI widgets. It is built on top of Soot [62] and Dexpler [15] and uses "Jimple" an intermediate representation of the Android Dalvik bytecode. Jimple is designed for performant static code analysis and is based on a typed three-address intermediate representation. It further provides only a handful of operations that makes it easier to

[5] https://gradle.org/, last accessed: 20.10.2022

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

analyze [62]. For data flow tracking (e.g., taint-analysis), FlowDroid uses the IFDS [49] framework. The general idea of IFDS (inter-procedural, finite, distributive, subset) is that dataflow-analysis problems can be solved precisely in polynomial time by transforming them into a special kind of graph-reachability problem [49]. The authors of FlowDroid show, via a set of experiments, the superior precision and recall of FlowDroid compared to the commercial tools AppScan Source and Fortify SCA [14].

BackDroid [69]. BackDroid is developed by Wu et al. [69] and is able to skip code which is irrelevant for the static analysis and only look at relevant code paths. The authors propose a technique called "on-the-fly bytecode search". This technique enables searching for the caller of a sink (backtracking) just in time and thus only relevant code paths are analyzed. Skipping irrelevant code paths results in better performance, as show by the authors im comparison with Amandroid/Argus: 3,178 popular apps were analyzed for crypto and SSL misconfigurations. BackDroid was 37 times faster while preserving close or even better detection rates.

Argus-SAF/Amandroid [65]. Argus-SAF (also known as Amandroid) is similar to FlowDroid with the main difference being Amandroid/Argus is able to track inter-component communication (ICC) extensively [53, 65].

3.4 Dynamic Analysis

In this section we discuss dynamic analysis approaches and tools needed in the scope of this work. Dynamic Analysis involves running Android apps and uses data which is gained from app execution. In this work we use a real Android smartphone to execute apps and the following approaches.

3.4.1 Android Debug Bridge (ADB)

The Android Debug Bridge (ADB) [26] is a command-line tool which allows to communicate and control an Android smartphone and its Android apps. ADB provides various useful commands, e.g., to install new apps on an Android phone or to debug apps already running. It involves three components (see Figure 3.3):

- *Client* - Command-line tool itself which sends commands to the server.
- *Server* - ADB server which is responsible for the communication (mostly USB or TCP) with the adb deamon running on the smartphone or emulator
- *Daemon (adbd)* - Daemon which runs on the Android smartphone and executes the given commands.

We use ADB to install, uninstall apps and to run commands, like stopping apps and running "Monkey" (see Section 3.4.2).

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.

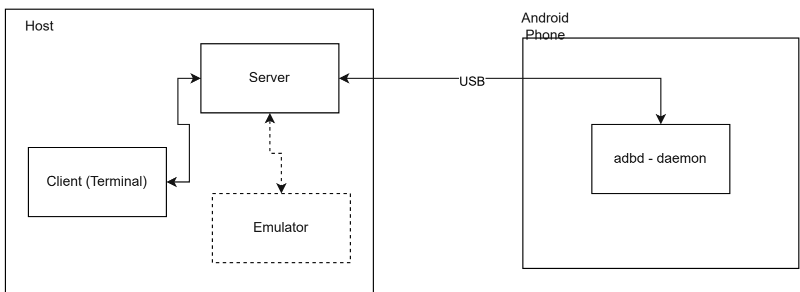

Figure 3.3: Components of the Android Debug Bridge (ADB).

3.4.2 Monkey

Monkey [28] runs on the device or emulator and is able to provide pseudo-random streams of user events (e.g., clicks, touches). Most often it is used for (stress-)testing apps. In our case, we use it to start the 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

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 4

Design and Implementation

In this chapter, we discuss the implementation of our analysis framework to detect and find LAN scanning apps.

4.1 Overview

As shown in Figure 4.1, our approach is split into three parts: In the first part we use Yara to pre-filter the initial set of apps for interesting ones. The second part takes the resulting set and uses a dynamic analysis approach to find apps that are scanning the LAN. In the last step we evaluate the found apps that the following questions in mind:

- RQ1: How many apps are scanning the local network?
- RQ2: Why is the app scanning the local network?
- RQ3: Did the user consent to this behavior?

4.2 App Pre-Filtering

As static data flow and dynamic analysis are rather resource intensive, we choose Yara to pre-filter our initial set of apps. This is done as follows: First, we unpack all apps with the Apktool. We use the following command to unpack apps:

```
apktool d -f -o <app>.out <app.apk>
```

In combination with the main ruleset (see Appendix A.1.1), we use the resulting files (except the files in the “res” folder, which mostly contains design/layout related files) as input for Yara. Then we string-concatenat all matching Yara rules and use them as input for Yara again. For the second Yara execution, we use the “meta-ruleset”: With “meta-rules” (see Appendix A.1.2) we can match apps on the rules matched in the 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

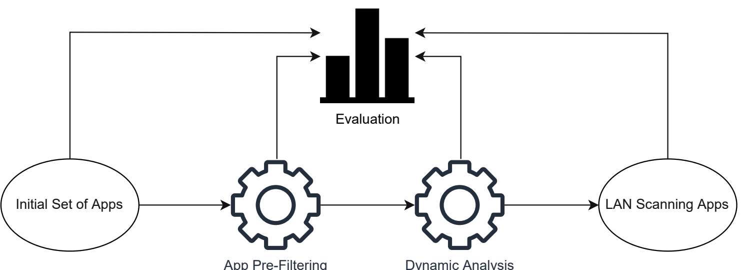

Figure 4.1: Overview of our hybrid (static and dynamic) analysis framework.

```
1  for a in apps:
2      matches = []
3      metamatches = []
4      for f in apktool(a):
5          for r in rules:
6              if Yara(f,r):
7                  matches.append(r)
8      for r in metarules:
9          if Yara(matches.toString(), r):
10             metamatches.append(r)
11     output(a)
12     output(matches)
13     output(metamatches)
```

Listing 4.1: Highlevel pseudocode pre-filtering with Yara.

Yara execution. This allows to easily match apps by various conditions based on their initial main ruleset matches. Listing 4.1 shows high level pseudocode for the pre-filtering with Yara:

We define the main Yara ruleset to match apps that we associated with LAN scanning, see Section 3.2.3. The Yara rules were handcrafted by analyzing apps which are known to have LAN scanning abilities, e.g., Fing [39] and Ning [30]. For example the rule in Listing 4.2 matches apps that are using the `isReachable()` function from the `InetAddress` class [5].

We show the full Yara ruleset in Appendix A.1.1 and discuss it in Section 5.1.

4.3 Static Analysis

Initially we planned to use a full static analysis approach without dynamic analysis, as using a dynamic approach can be rather complex and time consuming. We tried to use data flow analysis to detect “sinks” using data that is related to LAN scanning (e.g., IP addresses in LAN range 10.0.0.0/8). To evaluate for static (data flow) analysis we tried to use the following tools, but we sequentially found them to be impractical.

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.

```
1 rule PING_InetAddress_isReachable {
2     meta:
3         description = "Detects isReachable() function call often used for
              ping scanning"
4     strings:
5         $x1 = "Ljava/net/Inet4Address;->isReachable(" ascii
6         $x2 = "Ljava/net/InetAddress;->isReachable(" ascii
7     condition:
8         1 of ($x*)
9 }
```

Listing 4.2: Yara rule matching Java isReachable (Ping) API functions.

- Flowdroid/LeakScope/ValueScope (Soot-based) and Androguard:
  Have the same issue: Both are not able to "track" dataflow through threads. See Listing 4.3 for an example of code from the popular LAN network scanning app "Fing"[1] causing problems with analysis: Line 9, where the LAN scanning code is called, is wrapped insided a Java thread. The mentioned framework/tools were only able to track the flow to the calling of the thread, resulting in a "cut" of the data flow. Based on the Android/Java architecture, e.g., network operations are not allowed in the main thread,[2] we assume that almost all LAN scanning is done inside its own thread. Thus these frameworks/tools (without added functionality) are not suited for our purpose.

- Argus-SAF/Amandroid:
  We were not able to get Argus-SAF running properly and based on the commits on GitHub [55], we consider it unmaintained.

4.4 Dynamic Analysis

As we found the data flow analysis approaches impractical for our research, we decided to use a dynamic approach. We choose a similar approach to Kuchhal et al. [36], except we tested for Android apps and not websites. However, with this approach, we can not find LAN scanning activities that are triggered by user-interaction. Yet, we think that this approache catches the most interesting apps: Apps that are triggering LAN scanning activities at app startup and without user-interactions (except starting the app). This is of interest because such Android apps might try to hide their tracking/fingerprinting behavior from the user.

[1]https://play.google.com/store/apps/details?id=com.overlook.android.fing, last accessed: 20.10.2022
[2]https://stackoverflow.com/a/6343299, last accessed: 20.10.2022

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

```
1     public void f() {
2         synchronized (this.f14630f) {
3             if (this.f14627c == 1) {
4                 Log.d("fing:inet-finder", "Starting INET address finder
                      ...");
5                 this.f14627c = 2;
6                 Thread thread = new Thread(new Runnable() { // from class
                      : com.overlook.android.fing.engine.j.f.a
7                     @Override // java.lang.Runnable
8                     public final void run() {
9                         c.c(c.this);
10                    }
11                });
12                this.f14628d = thread;
13                thread.start();
14            }
15        }
16    }
```

Listing 4.3: Java code with threads causing problems in tracking the call flow/graph of the app.

The dynamic analysis involves the following steps: First, we connect the used Android smartphone to the testing WIFI where we can dump the network traffic via Tcpdump.[3] Sequential, for every Android app we start a network dump, install the app via ADB, and its main entrypoint (activity) via Monkey. Similar to the work of Kuchhal et al. [36], we just start the app and wait 60 seconds for the app to do its scanning activities. If the app is providing such functionality, we execute/trigger one broadcast and one service each, after these initial 60 seconds. We wait additional 30 seconds for executed broadcasts and services, which sums up to a maximum of 120 seconds. Except from starting the app and the triggering of broadcasts and services (if provided by the app) we do not trigger any additional interactions with the app. This means that any network scanning activity conducted by the respective app is triggered only via the mentioned started activities, broadcasts or services mentioned above.
We also used Monkey for user simulation by instructing it, after starting the app, to trigger 100 random touch events. We used the command from Listing 4.4 for Monkey. 10 random apps from the GP dataset and 10 random apps from the IoT dataset were used in this experiment. However, we found no additional scanning activity compared to running without the Monkey user simulation. Smart user simulations are part of our future work, see 6.1.
Finally, we kill and uninstall the app and stop the network dump. We execute all of the mentioned executed commands (e.g., installing an app using ADB) using the bash

[3] https://www.tcpdump.org/, last accessed: 20.10.2022

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.

```
1 adb shell monkey --throttle 250 -c android.intent.category.LAUNCHER -s
      666 --pct-touch 100 -p <package-name> 100
```

Listing 4.4: Monkey command line flags used to simulation touch events.

Table 4.1: Timeouts of ADB commands.

ADB Command	Duration (sec)
Install	120
Start	10
Triggering Service/Broadcast	120
Stopping App	10
Uninstall	120

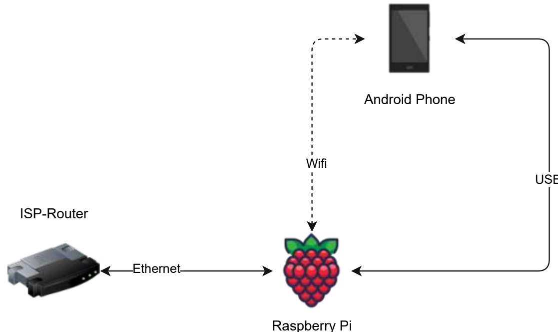

Figure 4.2: Architecture of the dynamic analysis testbed.

"timeout" function, see Table 4.1. If an ADB command runs into a timeout, we stop the dynamic analysis, to limit the execution time.

Figure 4.2 shows the architecture of the testbed. We choose a Raspberry PI[4] as the machine executing the ADB commands and dumping the network traffic. To setup the WiFi network on the Raspberry PI we used raspap[5] which provides a nice and easy way to setup WiFi networks. We hook the Android smartphone, a OnePlus 6T with 8GB RAM running Android 11 with OxygenOS 11.1.2.2, via USB to receive ADB commands and connect it via WIFI to dump the network traffic.

For the analysis of the traffic dump we wrote a tool using Go[6] and the GoPacket [27], a network packet processing library. After the execution of all apps we analyze the network dumps: We look for ARP and SSDP packets to find "volume-based" scanning activities (ARP) or multicast scanning activities, see Section 3.2.3. We detect:

[4]https://www.raspberrypi.com/products/raspberry-pi-4-model-b/, last accessed: 20.10.2022
[5]https://raspap.com/, last accessed: 20.10.2022
[6]https://go.dev/, last accessed: 20.10.2022

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

- *ARP* - If more than 3 ARP packets to different IPs from one single IP (Android smartphone IP) are found we detect it as ARP scan. We choose 3 ARP packets as some ARP packets are expected to be seen as this is normal behavior. Since every operating system has a different implementation of the ARP protocol there are different numbers of expect ARP packets. However there should not be more than 3 ARP packets in normal usage.
- *SSDP* - We detect every SSDP packet as scanning activity.

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 5

Results

In this chapter we present our results and answer the research questions presented in Chapter 1.1.

5.1 Yara Ruleset

Our Yara ruleset can classify Android apps that have the capabilities for LAN scanning (e.g., code/data needed for such functionality). We used apps (e.g., Fing [39] and Ning [30]) that are known to have such functionality for creating this ruleset. It consists of 43 normal Yara rules and one "meta-rule", see Section 4.2 for how they are used and their development process.
We categorize them as follows:

- Permission Rules
  Cover Android permissions.
- IP/MAC Rules
  Cover special IP and MAC addresses related to LAN.
- Keyword Rules
  Cover keywords as e.g., "portscan" in Android app code.
- Unix Path Rules
  Cover special Unix paths (e.g., "/proc/net/arp").
- Java API Rules
  Cover Java API functions which are related to scanning/networking.
- Unix Command Rules
  Cover Unix commands as e.g., "ping <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.

- Native Rules
  Look for functionality in native code (".so" files).

- Misc Rules
  Miscellaneous Yara rules.

Our Yara ruleset can be found in Appendix A.1.1. Each rule describes its purpose in the corresponding metadata field. Table 5.2, Table 5.3 and Table 5.4 show their matching rates on the respective Android app datasets. Further, we created one special Yara rule: the "meta-rule" - "LAN_Scanning_APK" (see Appendix A.1.2). We use this rule to select Android apps for further processing. Based on experiments with our normal Yara ruleset on known LAN scanning Android apps we build this "meta-rule" to catch behavior related to LAN scanning. It checks if an Android app matches at least four of the following Yara rules:

- Hardcoded_local_IPs
- proc_net_arp
- GATEWAY_IP_
- UPnP_Keyword_http
- UPnP_Multicast_Addr
- Scan_Keywords
- NsdManager_discoverServices
- PING_InetAddress_isReachable
- PING_cmd
- WifiInfo_getIpAddress
- Get_Local_IP
- Get_Local_via_WEB
- Get_Local_IP_hashcode
- InetAddress_getHostAddress

We ignored the Yara rules related to Android permissions in the "meta-rule" as previous work (e.g., Reardon et al. [45]) shows that some Android apps try to circumvent the permission system. However, we did not find any app circumventing the permission system. We also tested the app pre-filtering (see Section 4.2) with a weakened Yara meta-rule for the IoT dataset: Instead of *4 of them* we used *3 of them* in the rule condition

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

(see Appendix A.1.2) which results in fewer apps being filtered. With this weakened Yara meta-rule for the IoT dataset, we got 165 additional apps for the dynamic analysis (see Section 4.4) which resulted in one additional LAN scanning (ARP) app. We consider this a reasonable tradeoff between time and precision.

5.2 App Dataset

For conducting our research we used three different Android app datasets: IoT companion apps, top 1000 general purpose (GP) apps and malicious apps. We further manually choose apps for evaluating our analysis approach.

5.2.1 IoT Companion Apps

Companion apps are apps which are used in combination with IoT devices for various different purposes, e.g., initial setup or for updating [53]. Thus, these apps need to somehow find and communicate with local IoT devices.
We expect seeing LAN scanning from companion apps to be more likely compared to general purpose apps, as a respective app has to somehow "find" its corresponding IoT device in the LAN. In this scenario, if the user has given consent, e.g., via UI interaction, LAN scanning is normal and can be classified as intended behavior.
We used an updated app dataset of Wang et al. [64]: In 2022 only 1,259 apps were left for downloading from their initial 2,081 apps. As Renuka Kumar et al. show in their paper [37] this could be because of different reasons, e.g.: geoblocking.

5.2.2 Top 1000 General Purpose

In comparison to the IoT companion apps, where LAN scanning is somehow expected, we expect to see LAN scanning activities in the top 1,000 GP app dataset much less frequent. We used the top 1,000 GP apps from the Austrian Google Play Store.[1]

5.2.3 Malicious Apps

We downloaded 117 random Android APK samples, related to various APT groups and malware families (e.g., Hydra, APT-C-23, Joker, Teabot, ..), from MalwareBazaar and VirusTotal. See Appendix A.1.3 for a full list of md5 hashes.

5.2.4 Manually Choosen Apps

To validate our analysis approach we used the following apps: We used Fing [39] and Ning [30] for creating Yara rules. The two mentioned apps are used for LAN scanning in an administrative scenario and thus a good fit for creating Yara rules, which are looking for LAN scanning apps. For validating our dynamic analysis we used the following

[1]https://play.google.com/store/apps, last accessed: 20.10.2022

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

Table 5.1: Hybrid Analysis Results

Set	Apps	After pre-filter	Dumps with data	Dumps with scanning	ARP-Scan	SSDP-Scan
IoT	1,259	479 (38%)	471 (98.3%)	31 (6.5%)	7 (22.5%)	28 (90.3%)
Top 1000	1,000	323 (32.3%)	319 (98.7%)	2 (0.6%)	1 (50%)	1 (50%)

app: Hot Pot Browser - com.huoguo.browser.[2] Colleagues found this app in related research [43] as it is LAN scanning directly on startup - behavior which we look for.

5.3 Analysis Results

In this section, we present our research findings. First, we discuss the results of our static and dynamic analysis. Second, we showcase four case studies of apps we found to scan the LAN. Last, we give concrete answers to our research questions.

5.3.1 Static Analysis Results

Table 5.1 shows the results of the app filtering. We used 1,259 IoT, Top 1,000 GP apps and 117 known malicious apps for the initial app pre-filtering step using Yara. See Section 4.2 for our discussion of this static analysis approach. After the pre-filtering 479 IoT (38%), 323 GP (32.3%) and 0 malicious apps were left. We compared the matching rules of both app datasets and found the following relations: As we can see in Table 5.2, IoT apps seem to be more likely to use the Android permissions related to changing networking and WIFI state; based on the matching of the following Yara rules:

- AndroidManifest_Permission_CHANGE_WIFI_STATE
- AndroidManifest_Permission_CHANGE_WIFI_MULTICAST_STATE

Also, IoT apps are more likely to use/access *proc/net/arp* (see rule *proc_net_arp*), which is, as IoT/companion apps have to known about related LAN devices, somehow expected for IoT apps. *UPnP_Multicast_Addr* and *UPnP_Keyword_http* are more prevalent in the IoT dataset which is also expected as IoT apps are more likely to use UPnP. *Hardcoded_local_IPs* on the other hand, is quite common in both datasets. This is most likely because local IPs are used to detect rooted devices or emulators. For example, Schmidt et al. [53] found the IP "10.0.2.2" in a large number of apps. They found this IP in a ""React Native" library by Facebook, which uses this IP upon emulator detection. Based on the rule *NsdManager_discoverServices* we can see that service discovery activities are more prevalent in the IoT app dataset. Table 5.1 supports this, as the SSDP matches are much higher in the IoT app dataset as well.
Interestingly, the *Scan_Keywords* rule has more matches on the GP app dataset than the

[2]https://www.virustotal.com/gui/file/00c9f5646d2c93c920b5307990ee5e00aef3198f5e34a3a80811b1b0d130441c, last accessed: 20.10.2022

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

IoT app dataset, which is unexpected. Also rule *proc_x_stat* is interesting: it matches more often on the GP app dataset. We assume that app developers do this, just to get information about the current app process.

To further analyze the matching Yara rules, we combined the two Android app datasets (GP dataset and IoT dataset) into one dataset and created three new datasets for ARP and SSDP scanning Android app: the ignored app dataset (apps which did not survive the pre-filtering; 1,457 apps), the filtered app dataset (apps which did survive the pre-filtering; 794 / 773 apps), scanning app dataset (apps which were found ARP/SSDP scanning; 8 / 29 apps). The results of this analysis can be seen in Table 5.3 for ARP scanning and Table 5.4 for SSDP scanning.
What we can see from Table 5.3 and Table 5.4 is that the results for the "Ignored App Dataset" are always smaller than the results for the "Filtered App Dataset". This shows that the Yara "meta-rule" (see Section 4.2) indeed covers network-related (LAN scanning) capabilities pretty well.
Looking at the "ARP Scanning App Dataset" results in Table 5.3 we can see that a few of the Yara rules match on all of the Android apps:

- AndroidManifest_Permission_ACCESS_WIFI_STATE
- Get_Local_IP
- InetAddress_getHostAddress
- NetworkInterface_getInetAddresses
- NetworkInterface_getNetworkInterfaces

Further, looking at the "SSDP Scanning App Dataset" results in Table 5.4 we can see that the following few Yara rules match on all of the Android apps:

- AndroidManifest_Permission_ACCESS_WIFI_STATE
- UPnP_Multicast_Addr
- UPnP_Keyword_http
- InetAddress_getHostAddress

For this datasets a modified Yara "meta-rule" might have resulted in similar results, however, our Yara "meta-rule" catches a wide spectrum of capabilities needed for LAN scanning. For example the Yara rule "Get_Local_IP_hashcode" (which has 0 matches in the "ARP Scanning App Dataset") looks for code which (ab)uses (e.g., for obfuscation) the "hashCode()" function of "java.net.InetAddress" to load the current IP.
The results of the Yara rule "Scan_Keywords" are also interesting: It has 0 matches in the "ARP Scanning App Dataset" and 1 (3.45%) match in the "SSDP Scanning App

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

Dataset", whereas the respective matches in the "Filtered App Dataset" are 93 (11.71%) and 92 (11.90%). As this rule tries to find strings inside Android apps that clearly can be related to scanning activities (e.g., "portscan"), this could be an indicator that the Android apps we found LAN scanning might try to hide this functionality.
Also only 4 of the 8 Android apps in the "ARP Scanning App Dataset" are using the Unix command "ping" (see matches for Yara rule "PING_cmd" in Table 5.3).
Interestingly, the matches for the three rules in the "Native Rules" category (C_socket_usage, C_ICMP, C_http_request_parts) in the "SSDP Scanning App Dataset" are quite a bit higher compared to the "Native Rules" in the "ARP Scanning App Dataset". This indicates that the SSDP scanning Android apps are more likely to use native libraries.
The Yara rule "PING_InetAddress_isReachable" in the category "Java API Rules" in the "SSDP Scanning App Dataset" has 16 (55.17%) matches, whereas the same Yara rule has 0 matches in the "ARP Scanning App Dataset". This could indicate valid SSDP/UPnP behavior where the devices check if corresponding devices are (still) responsive.

5.3.2 Dynamic Analysis Results

Table 5.1 also shows ARP scanning results of the dynamic analysis: Based on the 479 IoT and 323 GP apps from the static analysis, we got for the IoT apps 471 (98.3%) and for the GP apps 319 (98.7%) traffic dumps with data. In the IoT traffic dumps we found 31 apps to conduct scanning activities (ARP and/or SSDP): in 7 we found ARP scanning activity and in 28 we found SSDP scanning activity. Whereas for the GP apps we only found two apps: for one app (*AliExpress*, see Section 5.4.1 for details) we only found ARP scanning activity, and a different app showed SSDP scanning activity. These results support our assumption that IoT apps are more likely to conduct LAN scanning. We performed some further general evaluations and analyses, not directly related to LAN scanning, on the 690 traffic dumps, which we discuss in Section 5.5.

5.4 Case Studies

In this subsection we present four apps which we found doing LAN scanning activities. For each one of the following four apps we conducted a case study: AliExpress (com.alibaba.aliexpresshd), Magic Home Pro (com.zengge.wifi), ENSPIRE Controller (com.yamaha.dkv.enspire) and Hot Pot Browser (com.huoguo.browser).

5.4.1 AliExpress - com.alibaba.aliexpresshd

Name: AliExpress
Package: com.alibaba.aliexpresshd[3]
Last Updated: 25.07.2022
Released on: 27.09.2012
Downloads: 500M+

[3]https://play.google.com/store/apps/details?id=com.alibaba.aliexpresshd, last accessed: 20.10.2022

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

App Version Code/Name: 406 / 8.40.0
Rating: 4.5/5 with 13.3 million ratings
Developer: Alibaba Mobile[4]
Date of Access: 20.10.2022
Purpose:
AliExpress, a online retail service based in China, is the chinese counterpart to Amazon. It launched in 2010 and their app can be considered a "online shopping" app: "AliExpress is the go-to online shopping app to find everything you need at your fingertips!". [2]
Data Safety:
Based on their Google Play Store site [2, 9], see Figure 5.1, AliExpress collects the following data: Personal info (name, email, user IDs, address and phone number), financial info (purchase history), app activity (app interactions and in-app search history), web browsing history, app info and performance (crash logs and diagnostics) and device or other IDs. Further, they state that the following data is shared with other companies or organizations: Personal info (name, email, address and phone number) and financial info (purchase history).
Privacy Policy:
In their privacy policy, [10, 41] in the section "Information That We Collect Automatically" we can see what is collected automatically. They state that they are collecting "Details of Platform buying and browsing activities", and that for this purpose they are collecting "IP addresses, date and time of access to the Platform, device type, unique device identification numbers, browser type, broad geographic location (e.g. country or city-level location), browsing patterns and details of how you have interacted with our Platform and the goods and services available on it". As we observed the LAN scanning activities at startup without any user interactions, we expected to find remarks in their privacy policy - we did not find any remarks towards LAN scanning activities.
Manual Analysis:
We manually installed and started the app, and the app greeted us with "classic" shopping-app-style content. Also, we found quite some features, that are not directly related to a shopping app: chatting, games, online wallet - it felt a little like a social media app. However, we did not find any features related to LAN scanning activities but we found a Reddit post (Mar 10, 2021) [31] where users are discussing why the AliExpress iOS app is asking for LAN access permissions. The users from the Reddit post speculate that the app is doing it for fingerprinting/tracking purposes but no proof is given. A different reason might be, that these information are used to display recommendations or to calculate prices based on the devices in the LAN. We tried to look into the code of the app and look for the code that is related to the scanning activities, but unfortunately we were not able to pinpoint it exactly, as there is quite a lot of code with various different functionality. However, we found the usage of "Alibaba MNN" - "MNN is a blazing fast, lightweight deep learning framework, battle-tested by business-critical use cases in Alibaba"[5] as native lib packed into the app.

[4] https://play.google.com/store/apps/developer?id=Alibaba+Mobile, last accessed: 20.10.2022
[5] https://github.com/alibaba/MNN, last accessed: 20.10.2022

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 POST /amdc/mobileDispatch?appkey=21371601&deviceId=
      Ysvr5OQb244DAF498o3fumJ9&platform=android&v=4.0 HTTP/1.1
2 Connection: close
3 Accept-Encoding: gzip
4 Content-Type: application/x-www-form-urlencoded
5 User-Agent: Dalvik/2.1.0 (Linux; U; Android 11; ONEPLUS A6013 Build/RKQ1
      .201217.002)
6 Host: amdc.aliexpress.com
7 Content-Length: 280
8
9 appVersion=8.40.0&mnc=wifi&lng=0.0&netType=WIFI&bssid=02%3A00%3A00%3A00%3
      A00%3A00&appName=Aliexpress_Android&channel=channel_name&sign=
      b8630549528c59359e2ba0faf742c625076bdc49&carrier=wifi&cv=0&t
      =1657531369063&platformVersion=11&domain=msg-global.m.taobao.com&
      signType=sec&lat=0.0
```

Listing 5.1: HTTP request from AliExpress leaking BSSID in the request body.

Further, upon manually analyzing the traffic dump, we found 4 HTTP request/response pairs. 3 HTTP request/response pairs go to amdc.aliexpress.com (47.246.136.167) and contain some sensible data as BSSID, and most likely longitude and latitude, see Listing 5.1 for one of the three HTTP requests. Additionally, we found one single HTTP request/response pair which seems to ask acs.m.taobao.com (198.11.189.91) for the current timestamp, see Listing 5.2.

Our Remarks:
As the main use of this app is to provide a shopping app to their customers, we can not think of any reason, which is related to shopping, to conduct LAN scanning activities. One none-shopping-related reason might be: fingerprinting the users and their LAN, as Kuchhal et al. [36] mentioned and the users of Reddit assume. Further, we are surprised that AliExpress still uses unencrypted HTTP for some of its communication.
Yara Rule Matches:
This app matched on 14/43 of our rules, see Listing 5.3.

5.4.2 Magic Home Pro - com.zengge.wifi

Name: Magic Home Pro
Package: com.zengge.wifi[6]
Last Updated: 11.07.2022
Released on: 18.12.2016
Downloads: 1M+
App Version Code/Name: 180 / 1.8.2
Rating: 3.1/5 with 8.14 thousand ratings

[6]https://play.google.com/store/apps/details?id=com.zengge.wifi, last accessed: 20.10.2022

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

```
GET /gw/mtop.common.getTimestamp/* HTTP/1.1
Connection: close
User-Agent: Dalvik/2.1.0 (Linux; U; Android 11; ONEPLUS A6013 Build/RKQ1
    .201217.002)
Host: acs.m.taobao.com
Accept-Encoding: gzip

HTTP/1.1 200 OK
Date: Mon, 11 Jul 2022 09:22:54 GMT
Content-Type: application/json;charset=UTF-8
Connection: close
ufe-result: A6
x-paramkey: mtop.common.getTimestamp
Content-length: 109
Server: Tengine/Aserver
s-rt: 2
EagleEye-TraceId: 212c89a316575313749343846e90f2

{"api":"mtop.common.getTimestamp","v":"*","ret":["SUCCESS
    ::..................."],"data":{"t":"1657531374936"}}
```

Listing 5.2: HTTP request from AliExpress asking for current timestamp.

```
GATEWAY_IP_2
Get_Local_IP
Hardcoded_local_IPs
InetAddress_getHostAddress
Interesting_Keywords
NetworkInterface_getInetAddresses
NetworkInterface_getNetworkInterfaces
Network_Interface
Special_MAC
TCP_Socket
UDP_DatagramSocket_send
proc_self_fd
proc_x_cmdline
proc_x_stat
```

Listing 5.3: Matching Yara rules for AliExpress app.

Developer: LED Controller[7]
Date of Access: 20.10.2022
Purpose:
This app provides controlling functionality for smart LED lights. The purpose of this app already hints scanning activities: findings corresponding devices.

[7]https://play.google.com/store/apps/developer?id=LED+Controller, last accessed: 20.10.2022

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

Data safety →

Safety starts with understanding how developers collect and share your data. Data privacy and security practices may vary based on your use, region, and age. The developer provided this information and may update it over time.

This app may share these data types with third parties
Personal info and Financial info

This app may collect these data types
Personal info, Financial info and 4 others

Data is encrypted in transit

You can request that data be deleted

See details

Figure 5.1: AliExpress Data Safety statement summary.

Data Safety:
As stated by the app developers, see Figure 5.2, the following data is collected [12, 19]: Personal info (email), app activity, app info and performance (crash logs and diagnostics), and device or other IDs. Further, they state that they do not share data with third parties [19]. However, based on their "Security practices" [19], they do not encrypt data ("Data isn't encrypted - Your data isn't transferred over a secure connection") and they do not provide a way for you to request the deletion of your data. This circumstance makes this app most likely not meeting the General Data Protection Regulation (GDPR) requirements.
Privacy Policy:
The privacy policy [13, 20] states that they are, not only, collecting "location information": "To provide users with more stringent data protection, starting with Android 6.0, Android has removed programming access to the device's local hardware identifier for applications using WIFI and Bluetooth. Now, to access the hardware identifiers of nearby external devices via Bluetooth and WIFI scanning, your app must have location permissions." [20] This paragraph mentions the LAN scanning activity but does not explain why they are doing it and what happens with the collected data. Some could argue that the purpose of the app implies this kind of behavior, but nevertheless, it is not clear if the data is used otherwise. Interestingly, their privacy policy somehow contradicts their "Data Safety" statement [19]: The privacy policy states that they are sharing your data "with service providers and other third parties who perform services on our behalf, such as analytics and marketing services" [20]. Their "Data Safety" statement [19], though, states that they do not share data with third parties.
Manual Analysis:
We manually looked into the app and found the code which is responsible for LAN scanning, see Listing 5.5. As expected, the code is looking for devices the app is able to controll: First the apps reads the IP address via *android.net.wifi.WifiInfo - getIpAddress()*.

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  AndroidManifest_Permission_ACCESS_NETWORK_STATE
2  AndroidManifest_Permission_ACCESS_WIFI_STATE
3  AndroidManifest_Permission_CHANGE_WIFI_STATE
4  GATEWAY_IP_1
5  Get_Local_IP
6  Hardcoded_local_IPs
7  InetAddress_getHostAddress
8  NetworkInterface_getInetAddresses
9  NetworkInterface_getNetworkInterfaces
10 PING_cmd
11 TCP_Socket
12 UDP_DatagramSocket_send
13 WifiInfo_getIpAddress
14 proc_self_fd
15 proc_x_stat
```

Listing 5.4: Matching Yara rules for Magic Home Pro app.

```
1  C5598h.m9230b("DeviceDiscover InDepth:" + C5546p0.this.f21157a);
2  for (int i = 1; i <= 254 && !C5546p0.this.f21166j; i++) {
3      String str2 = b + String.valueOf(i);
4      if (!str2.equalsIgnoreCase(c)) {
5          try {
6              Thread.sleep(10L);
7              C5546p0.this.f21158b.m30c("HF-A11ASSISTHREAD", str2, 48899);
8          } catch (Exception e3) {
9              e3.printStackTrace();
10         }
11     }
12 }
```

Listing 5.5: Code of Magic Home Pro which is responsible for LAN Scanning.

Then it generates the IP addresses to scan the /24 subnet. The *for*-loop in line 2-12 is responsible for this task. For every IP (except the own one) a UDP packet with the content “HF-A11ASSISTHREAD” is sent. We can see this behavior as this generated a massive amount of ARP request packets in our traffic dump, see Figure 5.3. The packet hex data from Figure 5.3 correspond to “HF-A11ASSISTHREAD”.

Our Remarks:

As the purpose of this app is to communicate with (IoT) devices in your LAN, we expected the scanning behavior. However, we expected this behavior after some kind of user interaction. Also, it would be nice to see some more information about this behavior in their privacy policy [20] or “Data Safety” statement. [19]

Yara Rule Matches:

This app matched on 15/43 of our rules, see Listing 5.4

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

Data safety →

Safety starts with understanding how developers collect and share your data. Data privacy and security practices may vary based on your use, region, and age. The developer provided this information and may update it over time.

No data shared with third parties
Learn more about how developers declare sharing

This app may collect these data types
Personal info, App activity and 2 others

Data isn't encrypted

Data can't be deleted

See details

Figure 5.2: Magic Home Pro Data Safety statement summary.

No.	Time	Source	Destination	Protocol	Length	Info
46	3.263145	10.3.141.126	10.3.141.255	UDP	59	49170 → 48899 Len=17
63	4.287873	10.3.141.126	10.3.141.255	UDP	59	49170 → 48899 Len=17
64	4.815225	10.3.141.126	10.3.141.1	UDP	59	49170 → 48899 Len=17
66	4.821975	3a:cf:08:09:5a:a0	Broadcast	ARP	42	Who has 10.3.141.2? Tell 10.3.141.126
67	4.843674	3a:cf:08:09:5a:a0	Broadcast	ARP	42	Who has 10.3.141.3? Tell 10.3.141.126
68	4.866076	3a:cf:08:09:5a:a0	Broadcast	ARP	42	Who has 10.3.141.4? Tell 10.3.141.126
69	4.880570	3a:cf:08:09:5a:a0	Broadcast	ARP	42	Who has 10.3.141.5? Tell 10.3.141.126
70	4.902530	3a:cf:08:09:5a:a0	Broadcast	ARP	42	Who has 10.3.141.6? Tell 10.3.141.126
71	4.924637	3a:cf:08:09:5a:a0	Broadcast	ARP	42	Who has 10.3.141.7? Tell 10.3.141.126
72	4.946129	3a:cf:08:09:5a:a0	Broadcast	ARP	42	Who has 10.3.141.8? Tell 10.3.141.126
73	4.963800	3a:cf:08:09:5a:a0	Broadcast	ARP	42	Who has 10.3.141.9? Tell 10.3.141.126
74	4.984623	3a:cf:08:09:5a:a0	Broadcast	ARP	42	Who has 10.3.141.10? Tell 10.3.141.126
75	4.999195	3a:cf:08:09:5a:a0	Broadcast	ARP	42	Who has 10.3.141.11? Tell 10.3.141.126
76	5.020686	3a:cf:08:09:5a:a0	Broadcast	ARP	42	Who has 10.3.141.12? Tell 10.3.141.126
77	5.042968	3a:cf:08:09:5a:a0	Broadcast	ARP	42	Who has 10.3.141.13? Tell 10.3.141.126
78	5.066098	3a:cf:08:09:5a:a0	Broadcast	ARP	42	Who has 10.3.141.14? Tell 10.3.141.126
79	5.086813	3a:cf:08:09:5a:a0	Broadcast	ARP	42	Who has 10.3.141.15? Tell 10.3.141.126
80	5.110133	3a:cf:08:09:5a:a0	Broadcast	ARP	42	Who has 10.3.141.16? Tell 10.3.141.126
81	5.131735	3a:cf:08:09:5a:a0	Broadcast	ARP	42	Who has 10.3.141.17? Tell 10.3.141.126
82	5.152553	3a:cf:08:09:5a:a0	Broadcast	ARP	42	Who has 10.3.141.18? Tell 10.3.141.126
83	5.175239	3a:cf:08:09:5a:a0	Broadcast	ARP	42	Who has 10.3.141.19? Tell 10.3.141.126
84	5.197730	3a:cf:08:09:5a:a0	Broadcast	ARP	42	Who has 10.3.141.20? Tell 10.3.141.126
85	5.212605	3a:cf:08:09:5a:a0	Broadcast	ARP	42	Who has 10.3.141.21? Tell 10.3.141.126
86	5.229319	3a:cf:08:09:5a:a0	Broadcast	ARP	42	Who has 10.3.141.22? Tell 10.3.141.126
87	5.252042	3a:cf:08:09:5a:a0	Broadcast	ARP	42	Who has 10.3.141.23? Tell 10.3.141.126

```
› Frame 46: 59 bytes on wire (472 bits), 59 bytes captured (472 bits)
› Ethernet II, Src: 3a:cf:08:09:5a:a0 (3a:cf:08:09:5a:a0), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
› Internet Protocol Version 4, Src: 10.3.141.126, Dst: 10.3.141.255
› User Datagram Protocol, Src Port: 49170, Dst Port: 48899
⌄ Data (17 bytes)
    Data: 48462d413131415353495354485245414 4
    [Length: 17]
```

Figure 5.3: LAN scanning activity of Magic Home Pro.

5.4.3 ENSPIRE Controller - com.yamaha.dkv.enspire

Name: ENSPIRE Controller
Package: com.yamaha.dkv.enspire[8]
Last Updated: Feb 24, 2021

[8]https://play.google.com/store/apps/details?id=com.yamaha.dkv.enspire, last accessed: 20.10.2022

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

Released on: May 31, 2016
Downloads: 10K+
App Version Code/Name: 10000017 / 1.2.2
Rating: 2.6/5 with 54 ratings
Developer: Yamaha Corporation[9]
Date of Access: 20.10.2022
Purpose:
The purpose of this app is to controlle the Yamaha Disklavier ENSPIRE (music instrument).
Data Safety:
Their Google Play Store site [11, 22] does not show any "Data Safety" statement, see Figure 5.5.
Privacy Policy:
We were not able to find any privacy policy which violates the Google Play Store regulations.
Manual Analysis:
On looking into the app we found that the app was developed with Apache Cordova [21]. This is interesting, because Abhinav M. et al. [42] published their research involving IoT companion apps written in "hybrid app" frameworks, like Apache Cordova. They developed HybriDiagnostics, a vulnerability-assessment framework that is able to detect nine different types of security issues in such hybrid app. Apache Cordova allows developers to create their apps with HTML, CSS, and JavaScript for practically every mobile platform. Because of this, we found the code which is responsible for the scanning activities written in JavaScript, see Listing 5.8 and 5.7. In the first Listing (5.8) we can see the code which is responsible for the huge amount of ARP packets seen in the traffic dump. The code from the second Listing (5.7) is responsible for the SSDP scanning activities found in the traffic dump, see Figure 5.4.
Interestingly this app is not requesting the "CHANGE_WIFI_MULTICAST_STATE" Android permission. This permission is needed to listen for multicast packets. In the case of this app, this permission is not needed (as this app is only sending SSDP requests and not listening to them), but many apps with similar functionality still declare this permission. We think that there is some confusion about when this permission is needed and when it is not needed.
Our Remarks:
Although the purpose of this app implies scanning activities, we would have expected to find some information about this behavior in their privacy policy or "Data Safety" statement. Their privacy policy or "Data Safety" statement does not even exist.
Yara Rule Matches:
This app matched on 11/43 of our rules, see Listing 5.6

[9] https://play.google.com/store/apps/developer?id=Yamaha+Corporation, last accessed: 20.10.2022

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  AndroidManifest_Permission_ACCESS_NETWORK_STATE
2  AndroidManifest_Permission_ACCESS_WIFI_STATE
3  Get_Local_IP
4  Hardcoded_local_IPs
5  InetAddress_getHostAddress
6  NetworkInterface_getInetAddresses
7  NetworkInterface_getNetworkInterfaces
8  Network_Interface
9  UPnP_Keyword_http
10 UPnP_Multicast_Addr
11 WifiInfo_getIpAddress
```

Listing 5.6: Matching Yara rules for ENSPIRE Controller app.

No.	Time	Source	Destination	Protocol	Length	Info
48	2.938065	10.3.141.126	239.255.255.250	SSDP	168	M-SEARCH * HTTP/1.1
79	3.748001	10.3.141.126	239.255.255.250	SSDP	168	M-SEARCH * HTTP/1.1
160	4.720809	10.3.141.126	239.255.255.250	SSDP	168	M-SEARCH * HTTP/1.1

```
Frame 160: 168 bytes on wire (1344 bits), 168 bytes captured (1344 bits)
Ethernet II, Src: 3a:cf:08:09:5a:a0 (3a:cf:08:09:5a:a0), Dst: IPv4mcast_7f:ff:fa (01:00:5e:7f:ff:fa)
Internet Protocol Version 4, Src: 10.3.141.126, Dst: 239.255.255.250
User Datagram Protocol, Src Port: 43347, Dst Port: 1900
Simple Service Discovery Protocol
  M-SEARCH * HTTP/1.1\r\n
  HOST: 239.255.255.250:1900\r\n
  MAN: "ssdp:discover"\r\n
  MX: 3\r\n
  ST: urn:schemas-upnp-org:device:Disklavier:1\r\n
  \r\n
  [Full request URI: http://239.255.255.250:1900*]
  [HTTP request 3/3]
  [Prev request in frame: 79]
```

Figure 5.4: SSDP traffic dump of ENSPIRE Controller.

5.4.4 Hot Pot Browser - com.huoguo.browser

Name: *Hot Pot Browser*
Package: com.huoguo.browser[10]
App Version Code/Name: 127000 / 1.2.7.0
Developer: *Unknown*
Purpose:
Android browser app.
Manual Analysis:
While research this app we found various interesting things: First, it scans the LAN

[10]http://www.downyi.com/downinfo/119264.html, last accessed: 20.10.2022

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

```
1  var next = function (addr) {
2      var url = 'http://' + addr + '/ctrl/getDescription.php';
3      var httpDone = function () {
4          $worker -= 1;
5          if ($cancel == null) {
6              if (nextAddr() == true) {
7                  next(long2ip($current));
8                  return;
9              }
10         }
11         if ($worker == 0) {
12             if ($cancel) $cancel(false, 'cancel.');
13             if ($callback) $callback(true, 'success.');
14         }
15     };
16     $worker += 1;
17         //appDebug.print('tcp: ' + url);
18     appHttpClient.get(url, { connect_timeout: appValue.TCP_TIMEOUT })
19         .then(function (response) {
20             appPianoList.add(response.data, 'tcp');
21             httpDone();
22         }, function (response) {
23             httpDone();
24         });
25 };
```

Listing 5.7: Code of ENSPIRE Controller which is responsible for LAN (ARP) Scanning.

Figure 5.5: ENSPIRE Controller Data Safety statement summary.

directly at startup and even some anti-virus engines on VirusTotal are flagging it.[11] Based on the VirusTotal report (Symantec Mobile Insight "AdLibrary:Igexin") the mentioned app uses the Igexin SDK, which is known to spy on its user [70]. As reported in a blog post by www.lookout.com [70], we were able to find the "plugin" framework code (see Listing 5.10) which allows the app to load and execute arbitrary code.
Secondly, we also found that the app is using the Jiguang SDK which is known to spy on its users as well [46]. In the Table 5.7 we can see the domain "jpush.cn" being quite

[11] https://www.virustotal.com/gui/file/00c9f5646d2c93c920b5307990ee5e00aef3198f5e34a3a80811b1b0d130441c, last accessed: 20.10.2022

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

```
1  var joinGroup = function (socketId) {
2      chrome.sockets.udp.joinGroup(socketId, $address, function (result) {
3          if (result < 0) {
4              chrome.sockets.udp.close(socketId);
5              done(false, 'joinGroup failure.');
6          } else {
7              $socketId = socketId;
8              done(true, 'success.');
9          }
10     });
11 };
12 var bind = function (socketId) {
13     chrome.sockets.udp.bind(socketId, '0.0.0.0', 0, function (result) {
14         if (result < 0) {
15             chrome.sockets.udp.close(socketId);
16             done(false, 'bind failure.');
17         } else {
18             joinGroup(socketId);
19         }
20     });
21 };
```

Listing 5.8: Code of ENSPIRE Controller which is responsible for LAN (SSDP) Scanning.

prominent in the IoT dataset - it is one of the domains used by the Jiguang SDK to exfiltrate PII.
Unfortunately, we were not able to exactly find the code which is responsible for the LAN scanning behavior. We found some potential code parts which could be responsible but as the code base is rather big with many (native) libraries, we can not pinpoint it exactly.
Our Remarks:
Except for classic browser functionality, it has some malicious "side features".
Yara Rule Matches:
This app matched on 18/43 of our rules, see Listing 5.9

5.5 Traffic Characterization

A by-product of our approach to find LAN scanning Android apps is a rather big number of traffic dumps. These traffic dumps contain various interesting statistics.
Table 5.5 shows some general statistics: We got 323 GP and 479 IoT traffic dumps and we found the avg. GP packet amount (2,046) to be about four times bigger than the avg. IoT packet amount (517). The avg. packet size in byte is rather similar for both datasets (GP 980 vs IoT 739), whereas the avg traffic dump size in kb is quite different: 2,036 kb for the GP dataset against 390 kb for the IoT dataset which roughly correlates with the avg packet amount per dump.

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

```
1  AndroidManifest_Permission_ACCESS_NETWORK_STATE
2  AndroidManifest_Permission_ACCESS_WIFI_STATE
3  AndroidManifest_Permission_CHANGE_WIFI_STATE
4  C_http_request_parts
5  C_socket_usage
6  Get_Local_IP
7  Hardcoded_local_IPs
8  InetAddress_getHostAddress
9  NetworkInterface_getInetAddresses
10 NetworkInterface_getNetworkInterfaces
11 Special_MAC
12 TCP_Socket
13 UDP_DatagramSocket_send
14 WifiInfo_getIpAddress
15 proc_net_arp
16 proc_self_fd
17 proc_x_cmdline
18 proc_x_stat
```

Listing 5.9: Matching Yara rules for com.huoguo.browser app.

We also collected some protocol-level related statistics for HTTP, HTTPS, TCP, UDP, ARP and DNS: see Table 5.6. Although unencrypted HTTP is considered bad practice we still found some usage inside the traffic dumps: Of 323 GP traffic dumps 27 (8.3%) contain unencrypted HTTP traffic, whereas of 479 (13.5%) IoT traffic dumps 65 contain unencrypted HTTP traffic. The GP dataset got an avg. of 0.36 HTTP packets per dump, whereas the IoT dataset has an avg. of 0.49 HTTP packets per dump. In comparison, we found an avg. of 518.92 HTTPS packets in the GP traffic dumps and an avg. of 166.09 HTTPS packets in the IoT traffic dumps. Correlating with the HTTPS statistics, we found an avg. of 540.33 TCP packets in the GP dumps and an avg. of 177.49 TCP packets for the IoT dumps. UDP usage is rather balanced: The GP dumps contain an avg. of 33.04 UDP packets and the IoT dumps an avg. of 23.34 UDP packets. In accordance with our research findings (we found IoT apps to be more likely to conduct LAN scannning activities), we see about three times as many ARP packets in the IoT dumps as in the GP dumps: an avg. of 5.85 ARP packets for the GP dumps and an avg. of 16.07 ARP packets for the IoT dumps. Further, we found an avg. of 8.06 DNS packets in the GP dumps and an avg. of 3.48 DNS packets in the IoT dumps.

Based on the collected DNS packets we looked for the top used domains in the respective traffic dumps, see Table 5.7. As expected, Google-related domains (e.g., googleapis.com) are quite prominent in both traffic dump datasets. Google is highly integrated into the Android landscape and thus this finding is implicit. One difference however can be seen in the usage of Chinese domains in the IoT traffic dumps in comparison to the GP traffic dumps, e.g., jpush.cn, umeng.com, taobao.com. This behavior can also be seen in Table 5.8, based on the Top-Level-Domains (TLD) of the found domains: In the IoT

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

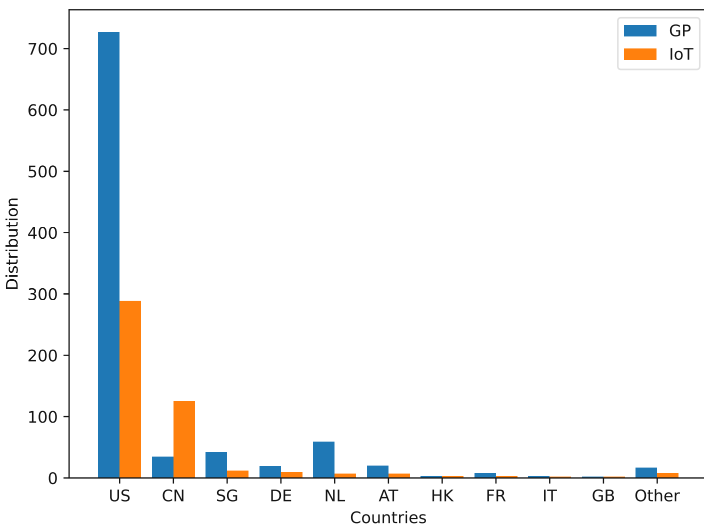

Figure 5.6: Country distribution of IPs from the GP and IoT app dataset.

section the "cn" domain is more prominent than in the GP section. We assume that the reason for the prominent usage of Chinese domains in the IoT traffic dumps is, that IoT devices and their companion apps are in many cases produced in China. We also looked at the destination (DST) IP addresses in the traffic dumps. Based on the DST IP addresses we created a bar chart that display the country distribution of the DST IPs of the respective traffic dumps, see Figure 5.6. As the USA holds many big and important hosting companies (e.g., Google, Microsoft, Amazon) it is expected that we see the US being the most used location. Again, the clear difference between the two bars is that China (CN) is very prominent in the IoT bar whereas CN is only fourth highest in the GP bar. At first glance the rather high occurenc of AT might be unexpected but as we conducted our tests from within Austria this is expected.

5.6 Discussion of Results

RQ1: Find apps that are conducting local network scanning activities.

We used a two-step approach (see Section 4.1) to find LAN scanning apps: First, we used a static approach (Yara) to find apps that might conduct LAN scanning. Table 5.1 shows that, based on the Yara rule matching, we found 479 Iot and 323 GP apps. In the second step, we used a dynamic approach (e.g., executing the apps) to find apps that are performing LAN scans without any user interaction. We found 7 ARP and 28 SSDP scanning apps in the IoT app dataset and 1 ARP and 1 SSDP scanning app in the GP dataset.

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

RQ2: Investigate why those apps are performing local network scans.

To answer this research question, we conducted case studies for four apps, see 5.4. We found the LAN scanning reasons for two (IoT: *Magic Home Pro* and *ENSPIRE Controller*) of these four apps to be valid: The apps were scanning the LAN because the app functionality requires them to do so. However, they do it without clear user consent or user interaction and their privacy policy lacks clear information about LAN scanning activities.
For the *AliExpress* app of the GP app dataset, we were not able to find valid reasons for LAN scanning activities. Further, their privacy policy also lacks information about such activities. Based on Kuchhal et al. [36] and a Reddit post (Mar 10, 2021) [31], it is possible that this is done for fingerprinting reasons.
The fourth app *com.huoguo.browser* is known to spy on its user [16], which we can confirm, and thus, we consider this app is scanning the LAN for malicious reasons.

RQ3: Study if the apps are scanning without user interaction/consent.

All of the apps we found scanning the LAN are doing it without any user interaction, except installing and starting the app. Further, we do not know of any privacy policy from any app that explicitly mentions LAN scanning and its reasons. Thus we conclude that these activities happen without user consent.

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

Table 5.2: Categorized Yara rules and their matches on the app datasets

Rule (Category)	GP Matches	IoT Matches
Permission Rules		
AndroidManifest_Permission_ACCESS_NETWORK_STATE	990 (99.00%)	1,055 (83.80%)
AndroidManifest_Permission_ACCESS_WIFI_STATE	745 (74.50%)	846 (67.20%)
AndroidManifest_Permission_CHANGE_WIFI_STATE	146 (14.60%)	569 (45.19%)
AndroidManifest_Permission_CHANGE_WIFI_MULTICAST_STATE	62 (6.20%)	389 (30.90%)
IP/MAC Rules		
Hardcoded_local_IPs	762 (76.20%)	845 (67.12%)
Special_MAC	148 (14.80%)	180 (14.30%)
UPnP_Multicast_Addr	38 (3.80%)	274 (21.76%)
Keyword Rules		
Scan_Keywords	76 (7.60%)	53 (4.21%)
ICMP_Keywords	3 (0.30%)	8 (0.64%)
Interesting_Keywords	710 (71.00%)	412 (32.72%)
Network_Interface	682 (68.20%)	577 (45.83%)
UPnP_Keyword_http	24 (2.40%)	186 (14.77%)
Unix Path Rules		
proc_net_tcp	10 (1.00%)	1 (0.08%)
proc_net_udp	5 (0.50%)	0 (0%)
proc_x_cmdline	166 (16.60%)	177 (14.06%)
proc_x_stat	638 (63.80%)	218 (17.32%)
proc_self_fd	958 (95.80%)	856 (67.99%)
proc_net_arp	11 (1.10%)	94 (7.47%)
Java API Rules		
WifiInfo_getIpAddress	413 (41.30%)	473 (37.57%)
Get_Local_IP_hashcode	1 (0.10%)	0 (0%)
Get_Local_IP	334 (33.40%)	397 (31.53%)
InetAddress_getHostAddress	939 (93.90%)	942 (74.82%)
NetworkInterface_getInetAddresses	603 (60.30%)	477 (37.89%)
NetworkInterface_getNetworkInterfaces	695 (69.50%)	549 (43.61%)
NsdManager_discoverServices	10 (1.00%)	73 (5.80%)
PING_InetAddress_isReachable	19 (1.90%)	107 (8.50%)
UDP_DatagramSocket_send	446 (44.60%)	489 (38.84%)
UPnP_MulticastSocket_send	30 (3.00%)	229 (18.19%)
TCP_Socket	857 (85.70%)	773 (61.40%)
GATEWAY_IP_1	77 (7.70%)	219 (17.39%)
GATEWAY_IP_2	29 (2.90%)	175 (13.90%)
Unix Command Rules		
ifconfig_eth0	0 (0%)	1 (0.08%)
ip_neigh	3 (0.30%)	5 (0.40%)
PING_cmd	136 (13.60%)	125 (9.93%)
Native Rules		
C_socket_usage	185 (18.50%)	428 (34.00%)
C_ICMP	43 (4.30%)	79 (6.27%)
C_http_request_parts	162 (16.20%)	323 (25.66%)
Misc Rules		
NSD_ServiceTypes	2 (0.20%)	0 (0%)
SOAP_xml	335 (33.50%)	203 (16.12%)
Get_Local_via_WEB	6 (0.60%)	6 (0.48%)

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

Table 5.3: Categorized Yara rules and their matches on: ignored app dataset (apps which did not survive the pre-filtering; 1,457 apps), filtered app dataset (apps which did survive the pre-filtering; 794 apps), ARP scanning app dataset (apps which were found ARP scanning; 8 apps).

Rule (Category)	Ignored App Dataset	Filtered App Dataset	ARP Scanning App Dataset
Permission Rules			
AndroidManifest_Permission_ACCESS_NETWORK_STATE	1,256 (86.20%)	782 (98.49%)	7 (87.50%)
AndroidManifest_Permission_ACCESS_WIFI_STATE	818 (56.14%)	765 (96.35%)	8 (100.00%)
AndroidManifest_Permission_CHANGE_WIFI_STATE	296 (20.32%)	415 (52.27%)	4 (50.00%)
AndroidManifest_Permission_CHANGE_WIFI_MULTICAST_STATE	98 (6.73%)	351 (44.21%)	2 (25.00%)
IP/MAC Rules			
Hardcoded_local_IPs	845 (58.00%)	755 (95.09%)	7 (87.50%)
Special_MAC	95 (6.52%)	232 (29.22%)	1 (12.50%)
UPnP_Multicast_Addr	27 (1.85%)	281 (35.39%)	4 (50.00%)
Keyword Rules			
Scan_Keywords	36 (2.47%)	93 (11.71%)	0 (0.00%)
ICMP_Keywords	1 (0.07%)	10 (1.26%)	0 (0.00%)
Interesting_Keywords	643 (44.13%)	476 (59.95%)	3 (37.50%)
Network_Interface	674 (46.26%)	582 (73.30%)	3 (37.50%)
UPnP_Keyword_http	7 (0.48%)	200 (25.19%)	3 (37.50%)
Unix Path Rules			
proc_net_tcp	3 (0.21%)	8 (1.01%)	0 (0.00%)
proc_net_udp	0 (0.00%)	5 (0.63%)	0 (0.00%)
proc_x_cmdline	162 (11.12%)	180 (22.67%)	1 (12.50%)
proc_x_stat	477 (32.74%)	374 (47.10%)	5 (62.50%)
proc_self_fd	1,103 (75.70%)	705 (88.79%)	6 (75.00%)
proc_net_arp	0 (0.00%)	105 (13.22%)	0 (0.00%)
Java API Rules			
WifiInfo_getIpAddress	235 (16.13%)	646 (81.36%)	5 (62.50%)
Get_Local_IP_hashcode	0 (0.00%)	1 (0.13%)	0 (0.00%)
Get_Local_IP	145 (9.95%)	578 (72.80%)	8 (100.00%)
InetAddress_getHostAddress	1,086 (74.54%)	787 (99.12%)	8 (100.00%)
NetworkInterface_getInetAddresses	400 (27.45%)	672 (84.63%)	8 (100.00%)
NetworkInterface_getNetworkInterfaces	525 (36.03%)	711 (89.55%)	8 (100.00%)
NsdManager_discoverServices	5 (0.34%)	76 (9.57%)	2 (25.00%)
PING_InetAddress_isReachable	13 (0.89%)	113 (14.23%)	0 (0.00%)
UDP_DatagramSocket_send	379 (26.01%)	550 (69.27%)	6 (75.00%)
UPnP_MulticastSocket_send	28 (1.92%)	228 (28.72%)	3 (37.50%)
TCP_Socket	906 (62.18%)	718 (90.43%)	6 (75.00%)
GATEWAY_IP_1	34 (2.33%)	259 (32.62%)	3 (37.50%)
GATEWAY_IP_2	14 (0.96%)	188 (23.68%)	2 (25.00%)
Unix Command Rules			
ifconfig_eth0	0 (0.00%)	1 (0.13%)	0 (0.00%)
ip_neigh	0 (0.00%)	8 (1.01%)	0 (0.00%)
PING_cmd	54 (3.71%)	203 (25.57%)	4 (50.00%)
Native Rules			
C_socket_usage	280 (19.22%)	331 (41.69%)	2 (25.00%)
C_ICMP	26 (1.78%)	96 (12.09%)	0 (0.00%)
C_http_request_parts	208 (14.28%)	276 (34.76%)	1 (12.50%)
Misc Rules			
NSD_ServiceTypes	0 (0.00%)	2 (0.25%)	0 (0.00%)
SOAP_xml	220 (15.10%)	317 (39.92%)	1 (12.50%)
Get_Local_via_WEB	1 (0.07%)	11 (1.39%)	0 (0.00%)

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

Table 5.4: Categorized Yara rules and their matches on: ignored app dataset (apps which did not survive the pre-filtering; 1,457 apps), filtered app dataset (apps which did survive the pre-filtering; 773 apps), SSDP scanning app dataset (apps which were found SSDP scanning; 29 apps).

Rule (Category)	Ignored App Dataset	Filtered App Dataset	SSDP Scanning App Dataset
Permission Rules			
AndroidManifest_Permission_ACCESS_NETWORK_STATE	1,256 (86.20%)	761 (98.45%)	28 (96.55%)
AndroidManifest_Permission_ACCESS_WIFI_STATE	818 (56.14%)	744 (96.25%)	29 (100.00%)
AndroidManifest_Permission_CHANGE_WIFI_STATE	296 (20.32%)	403 (52.13%)	16 (55.17%)
AndroidManifest_Permission_CHANGE_WIFI_MULTICAST_STATE	98 (6.73%)	328 (42.43%)	25 (86.21%)
IP/MAC Rules			
Hardcoded_local_IPs	845 (58.00%)	736 (95.21%)	26 (89.66%)
Special_MAC	95 (6.52%)	225 (29.11%)	8 (27.59%)
UPnP_Multicast_Addr	27 (1.85%)	256 (33.12%)	29 (100.00%)
Keyword Rules			
Scan_Keywords	36 (2.47%)	92 (11.90%)	1 (3.45%)
ICMP_Keywords	1 (0.07%)	9 (1.16%)	1 (3.45%)
Interesting_Keywords	643 (44.13%)	473 (61.19%)	6 (20.69%)
Network_Interface	674 (46.26%)	564 (72.96%)	21 (72.41%)
UPnP_Keyword_http	7 (0.48%)	174 (22.51%)	29 (100.00%)
Unix Path Rules			
proc_net_tcp	3 (0.21%)	8 (1.03%)	0 (0.00%)
proc_net_udp	0 (0.00%)	5 (0.65%)	0 (0.00%)
proc_x_cmdline	162 (11.12%)	177 (22.90%)	4 (13.79%)
proc_x_stat	477 (32.74%)	374 (48.38%)	5 (17.24%)
proc_self_fd	1,103 (75.70%)	686 (88.75%)	25 (86.21%)
proc_net_arp	0 (0.00%)	97 (12.55%)	8 (27.59%)
Java API Rules			
WifiInfo_getIpAddress	235 (16.13%)	629 (81.37%)	22 (75.86%)
Get_Local_IP_hashcode	0 (0.00%)	1 (0.13%)	0 (0.00%)
Get_Local_IP	145 (9.95%)	561 (72.57%)	25 (86.21%)
InetAddress_getHostAddress	1,086 (74.54%)	766 (99.09%)	29 (100.00%)
NetworkInterface_getInetAddresses	400 (27.45%)	654 (84.61%)	26 (89.66%)
NetworkInterface_getNetworkInterfaces	525 (36.03%)	692 (89.52%)	27 (93.10%)
NsdManager_discoverServices	5 (0.34%)	73 (9.44%)	5 (17.24%)
PING_InetAddress_isReachable	13 (0.89%)	97 (12.55%)	16 (55.17%)
UDP_DatagramSocket_send	379 (26.01%)	535 (69.21%)	21 (72.41%)
UPnP_MulticastSocket_send	28 (1.92%)	207 (26.78%)	24 (82.76%)
TCP_Socket	906 (62.18%)	703 (90.94%)	21 (72.41%)
GATEWAY_IP_1	34 (2.33%)	248 (32.08%)	14 (48.28%)
GATEWAY_IP_2	14 (0.96%)	183 (23.67%)	7 (24.14%)
Unix Command Rules			
ifconfig_eth0	0 (0.00%)	1 (0.13%)	0 (0.00%)
ip_neigh	0 (0.00%)	7 (0.91%)	1 (3.45%)
PING_cmd	54 (3.71%)	203 (26.26%)	4 (13.79%)
Native Rules			
C_socket_usage	280 (19.22%)	320 (41.40%)	13 (44.83%)
C_ICMP	26 (1.78%)	86 (11.13%)	10 (34.48%)
C_http_request_parts	208 (14.28%)	272 (35.19%)	5 (17.24%)
Misc Rules			
NSD_ServiceTypes	0 (0.00%)	2 (0.26%)	0 (0.00%)
SOAP_xml	220 (15.10%)	293 (37.90%)	25 (86.21%)
Get_Local_via_WEB	1 (0.07%)	11 (1.42%)	0 (0.00%)

Table 5.5: General statistics for app dataset traffic dumps.

	GP	IoT
Amount of traffic dumps	323	479
Avg. packets per traffic dump	2,046	517
Avg. packet size (byte) per traffic dump	980	739
Avg. traffic dump size (kb)	2,036	390

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

```
1  public boolean m7299a(Context context, String str, String str2, String
       str3, String str4) {
2      Class cls = null;
3      File file = new File(str);
4      File file2 = new File(str + ShareConstants.JAR_SUFFIX);
5      C2997e.m7193a(context.getFilesDir().getAbsolutePath(), str4, false);
6      C2997e.m7188b(file, file2, str3);
7      if (file2.exists()) {
8          try {
9              try {
10                 cls = new DexClassLoader(file2.getAbsolutePath(), context
                       .getFilesDir().getAbsolutePath(), null, context.
                       getClassLoader()).loadClass(str2);
11             } catch (Exception e) {
12                 ActivityC2755b.m8045a(f9404a + "|" + e.toString());
13             }
14             file2.delete();
15             C2997e.m7193a(context.getFilesDir().getAbsolutePath(), str4,
                   true);
16             if (cls == null) {
17                 return false;
18             }
19             IPushExtension iPushExtension = (IPushExtension) cls.
                   newInstance();
20             if (iPushExtension != null) {
21                 try {
22                     iPushExtension.init(C2922g.f9218f);
23                     this.f9406b.add(iPushExtension);
24                     ActivityC2755b.m8045a(f9404a + "| [main] extension
                           loaded: " + str);
25                     return true;
26                 } catch (Exception e2) {
27                     ActivityC2755b.m8045a(f9404a + "|" + e2.toString());
28                 }
29             }
30         } catch (Throwable th) {
31             ActivityC2755b.m8045a(f9404a + "|" + th.toString());
32             if (file2.exists()) {
33                 file2.delete();
34             }
35         }
36     }
37     return false;
38 }
```

Listing 5.10: igexin library code which loads external code.

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

Table 5.6: Protocol statistics for app dataset traffic dumps (per dump).

	GP	IoT
Dumps with HTTP packets	27/323 (8.3%)	65/479 (13.5%)
Avg. HTTP packets	0.36	0.49
Avg. HTTPS packets	518.92	166.09
Avg. TCP packets	540.33	177.49
Avg. UDP packets	33.04	23.34
Avg. ARP packets	5.85	16.07
Avg. DNS packets	8.06	3.48

Table 5.7: Top 20 domains of GP and IoT dataset

GP		**IoT**	
Domain	**Amount**	**Domain**	**Amount**
googleapis.com	431	googleapis.com	359
facebook.com	347	crashlytics.com	120
voodoo-ads.io	112	facebook.com	86
crashlytics.com	102	google.com	83
appsflyer.com	77	jpush.cn	79
unity3d.com	70	app-measurement.com	72
app-measurement.com	60	gstatic.com	46
google.com	57	doubleclick.net	41
applovin.com	57	reolink.com	33
gstatic.com	56	applovin.com	32
doubleclick.net	34	qq.com	31
adcolony.com	34	googleusercontent.com	25
branch.io	30	amazonaws.com	23
vungle.com	28	googlesyndication.com	21
rayjump.com	25	umeng.com	20
googleusercontent.com	25	tuyaeu.com	17
amazonaws.com	24	googletagservices.com	15
startappservice.com	22	mob.com	15
supersonicads.com	22	taobao.com	14
appcenter.ms	20	flurry.com	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 Bibliothek Your knowledge hub WIEN

Table 5.8: Top 20 TLD's of GP and IoT dataset

GP		**IoT**	
Domain	**Amount**	**Domain**	**Amount**
com	2,145	com	1,409
io	177	cn	93
net	102	net	74
ms	30	io	32
mobi	22	ms	12
cn	17	tv	9
eu	13	arpa	7
org	12	co	5
at	11	org	5
tv	9	in	3
info	8	cloud	3
me	7	cc	2
video	7	bike	1
sg	6	me	1
website	3	za	1
arpa	2	de	1
app	2	es	1
cloud	2	eu	1
world	2	it	1
hk	2	br	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.

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

Discussion

In this chapter we talk about limitations of our approach and future work.

6.1 Limitations

In this section we discuss the limitations and shortcomings of our approach.

Limited User Simulation/Interaction

In our approach we only start the Android apps and do not provide any user simulation/interaction. We tried to use Android Monkey (see Section 3.4.2) for basic user simulation/interaction (see Section 4.4), however, as Monkey is not designed to be used as user simulation but rather as stress testing tool, we did not get transparent, repeatable results. We argue that our approach without user simulation catches the most interesting apps: apps which might try to hide their LAN scanning activities. However, a full and smart user simulation/interaction will most likely result in finding more LAN scanning activities.

Simulation Time

In our dynamic analysis approach, we start the Android apps for 60 (120 seconds if the app provides broadcasts or services, see Section 4.4) seconds. For this 60 seconds timeframe, we dump the network traffic and look for LAN scanning activities. After 60 seconds we kill and uninstall the Android apps. This approach misses all the LAN scanning activities which happen after 60 seconds. We argue that we catch most of the LAN scanning activities without user interaction in the 60-second period. However, increasing the timeframe will likely result in more findings.

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.

Encrypted Traffic

As our approach of detecting LAN scanning activities relies on unencrypted protocols (ARP and SSDP, see Section 3.2.1) there was no need to implement a functionality (e.g., HTTPS mitmproxy[1]) that help in analyzing encrypted traffic like HTTPS. However, such functionality would have allowed us to dig deeper into the traffic dumps to find more interesting data for the Miscellaneous Findings Section 5.5, like PII leaks.

App Entrypoints

Android apps have multiple different ways to trigger functionality, like broadcasts and services. In our approach, except for starting the app, we trigger one broadcast and one service (if existing) for each app, see Section 4.4. With this approach, we did not trigger all possible entrypoints and thus not we might have missed LAN scanning activities. However, we argue that this is negligible.

6.2 Future Work

First, it would be interesting to add a real/smart user simulation to our dynamic analysis. This would trigger more app functionality and result in more network traffic and thus most likely in more LAN scanning activities to be found. For the scope of this work we omitted the user simulation (except for some tests with Monkey, see Section 4.4) as this topic is not straight-forward and there is interesting ongoing research in this field.

Also, we like to add functionality (like HTTPS proxies) that makes it possible to analyze encrypted traffic. We then would be able to analyze the network dumps more thoroughly. For the scope of this work we did not implement such functionality as some apps might not work with it, e.g., some apps might use TLS certificate pinnging and will not behave as normal.

Additionally, it would be interesting to start the apps inside a network with additional different devices. This would allow us to see if and how the apps are trying to interact with devices on the same network. Further, this would allow us to analyze if apps are only scanning for devices or if they also try to interact.

In our work we did not find Android apps which circumvent the Android permission system. However, we would like to research the field of Android permission circumvention related to network permissions in the future.

Furthermore, as we found some apps with a privacy policy differing from their Google Play Store “Data Safety” statement, e.g., Magic Home Pro - com.zengge.wifi 5.4.2, we would like to run a large-scale analysis to find more apps with differing privacy statements.

As this approach covers only Android apps, in future work it would be interesting to also cover iOS apps. Since iOS 14, apps that want to interact with devices on the local

[1]https://mitmproxy.org/, last accessed: 20.10.2022

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

network have to ask for permission [8] and it would be interesting to see if some apps can circumvent that permission. However, dynamic analysis for iOS apps is more dificult, as there are less tools and frameworks for analyzing apps, compared to Android 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

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 7

Conclusion

Apps that are able to access the LAN can collect information about other devices connected and use this information for profiling and fingerprinting. This is the reason iOS 14 introduced a new permission which any app that wants to interact with the LAN has to ask for. Android does not have such a permission. Thus we show that Android apps scanning the LAN without a valid reason exist and that there is need for a similar permission in the Android system.

We found LAN scanning Android apps by implementing a hybrid analysis approach, consisting of three steps: We first pre-filtered the app dataset with over 40 manually crafted Yara rules to find Android apps that contain data/functions related to LAN scanning activities. In the next step, we started each app from the pre-filtered dataset on a real smartphone and capture its network traffic. After the dynamic execution of the Android apps, we analyzed the traffic dumps and look for ARP and SSDP scanning activities.

We used three different Android app datasets for our analysis: the top 1,000 general purpose (GP) apps from the Austrian Google Play Store, 1259 companion/IoT apps, and 117 known malware apps. We found 7 ARP and 28 SSDP scanning apps in the IoT dataset and 1 ARP and 1 SSDP scanning app in the top 1,000 GP dataset. We did not find any LAN scanning app in the known malware app dataset.

Based on the found apps we created four case studies to analyze the reasons for LAN scanning. The AliExpress Android app, see case study 5.4.1, was found to scan the LAN without stating it in their privacy policy and without a clear reason for doing it. We reason that this is most likely done for fingerprinting reasons. Magic Home Pro, see case study 5.4.2, and ENSPIRE Controller, see case study 5.4.3, both IoT apps, also perform LAN scanning activities. Magic Home Pro somehow states it in its privacy policy but does not actively ask the user for permission. ENSPIRE Controller does not even have a privacy policy. Both apps are used to controll IoT devices, which makes this behavior expected, however, we expected this behavior after some kind of user interaction and

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.

Bibliothek, Your knowledge hub

to be clearly stated in their privacy policy. Hot Pot Browser, see case study 5.4.4, also scans the LAN and uses libraries that are known to spy on its users.
With our work, we are making a step towards finding mobile apps performing LAN scanning attacks and providing a basis for considering new Android permissions regarding LAN access.

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

Appendix

A.1 Yara Rules

A.1.1 Main Rules

```
rule AndroidManifest_Permission_CHANGE_WIFI_STATE {
    meta:
        description = "Detects permissions in AndroidManifest"
    strings:
        $x4 = "android.permission.CHANGE_WIFI_STATE" ascii
    condition:
        filename == "AndroidManifest.xml" and 1 of them
}

rule AndroidManifest_Permission_CHANGE_WIFI_MULTICAST_STATE {
    meta:
        description = "Detects permissions in AndroidManifest"
    strings:
        $x3 = "android.permission.CHANGE_WIFI_MULTICAST_STATE" ascii
    condition:
        filename == "AndroidManifest.xml" and 1 of them
}

rule AndroidManifest_Permission_ACCESS_NETWORK_STATE {
    meta:
        description = "Detects permissions in AndroidManifest"
    strings:
        $x2 = "android.permission.ACCESS_NETWORK_STATE" ascii
    condition:
        filename == "AndroidManifest.xml" and 1 of them
}

rule AndroidManifest_Permission_ACCESS_WIFI_STATE {
    meta:
        description = "Detects permissions in AndroidManifest"
    strings:
        $x1 = "android.permission.ACCESS_WIFI_STATE" ascii
    condition:
        filename == "AndroidManifest.xml" and 1 of them
}



rule Hardcoded_local_IPs {
    meta:
        description = "Detects hardcoded local IPs"
```


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

```
42      strings:
43          $x1 = /(127\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})
                 |(10\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})
                 |(172\.1[6-9]{1}[0-9]{0,1}\.[0-9]{1,3}\.[0-9]{1,3})
                 |(172\.2[0-9]{1}[0-9]{0,1}\.[0-9]{1,3}\.[0-9]{1,3})
                 |(172\.3[0-1]{1}[0-9]{0,1}\.[0-9]{1,3}\.[0-9]{1,3})
                 |(192\.168\.[0-9]{1,3}\.[0-9]{1,3})/
44      condition:
45          1 of ($x*)
46  }
47
48  rule Special_MAC {
49      meta:
50          description = "Detects special MAC addresses"
51      strings:
52          $ = "ff:ff:ff:ff:ff:ff" ascii fullword
53          $ = "00:00:00:00:00:00" ascii fullword
54      condition:
55          1 of them
56  }
57
58  rule UPnP_Multicast_Addr {
59      meta:
60          description = "Detects UPnP multicast address"
61      strings:
62          $x1 = "224.0.0.251" ascii
63          $x3 = "239.255.255.250" ascii
64      condition:
65          1 of ($x*)
66  }
67
68
69
70  rule Scan_Keywords {
71      meta:
72          description = "Detects keywords associated with network scanning"
73      strings:
74          $ = "portscan" ascii nocase
75          $ = "tcpscan" ascii nocase
76          $ = "udpscan" ascii nocase
77          $ = "arpscan" ascii nocase
78          $ = "upnpscan" ascii nocase
79          $ = "pingscan" ascii nocase
80          $ = "NetworkScanEngine" ascii nocase
81      condition:
82          1 of them
83  }
84
85  rule ICMP_Keywords {
86      meta:
87          description = "Detects ICMP keywords"
88      strings:
89          $ = "ICMP_ECHOREPLY" ascii
90          $ = "ICMP_DEST_UNREACH" ascii
91          $ = "ScannerIcmp" ascii nocase
92          $ = "ScannerPing" ascii nocase
93          $ = "Destination Host Unreachable" ascii
94          $ = "Time to live exceeded" ascii
95      condition:
96          1 of them
97  }
98
99  rule Interesting_Keywords {
100     strings:
101         $ = "malware" ascii nocase
102         $ = "backdoor" ascii nocase
103         $ = "keylogger" ascii nocase
104         $ = "stealer" ascii nocase
105         //$ = "wiper" ascii nocase
106         $ = "exploit" ascii nocase
107     condition:
108         1 of them
109 }
```


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

```
110
111 rule Network_Interface {
112     meta:
113         description = "Detects names of network interfaces"
114     strings:
115         $x1 = "wlan" ascii fullword
116         $x2 = "eth" ascii fullword
117     condition:
118         1 of ($x*)
119 }
120
121 rule UPnP_Keyword_http {
122     meta:
123         description = "Detects UPnP keywords"
124     strings:
125         $ = "urn:schemas-upnp-org:device:InternetGatewayDevice:1" ascii
126         $ = "upnp:rootdevice" ascii
127         $ = "M-SEARCH * HTTP/1.1" ascii
128         $ = "_services_dns-sd_udplocal" ascii
129     condition:
130         1 of them
131 }
132
133
134 rule proc_net_tcp {
135     meta:
136         description = "Detects linux filesystem access to /proc/net/tcp"
137     strings:
138         $x1 = "/proc/net/tcp"
139     condition:
140         1 of them
141 }
142
143 rule proc_net_udp {
144     meta:
145         description = "Detects linux filesystem access to /proc/net/udp"
146     strings:
147         $x1 = "/proc/net/udp"
148     condition:
149         1 of them
150 }
151
152 rule dev_tcp {
153     meta:
154         description = "Detects linux filesystem TCP sockets"
155     strings:
156         $x1 = "/dev/tcp/"
157     condition:
158         1 of them
159 }
160
161 rule dev_udp {
162     meta:
163         description = "Detects linux filesystem UDP sockets"
164     strings:
165         $x1 = "/dev/udp/"
166     condition:
167         1 of them
168 }
169
170 rule proc_x_cmdline {
171     meta:
172         description = "Detects reading /proc/.*/cmdline"
173     strings:
174         $re = /\/proc\/.{2,10}\/cmdline/
175     condition:
176         1 of them
177 }
178
179 rule proc_x_task_x_fd {
180     meta:
181         description = "Detects reading /proc/%s/task/%s/fd"
182     strings:
```

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

```
183         $re = /\/proc\/.{2,10}\/task\/.{2,10}\/fd/
184     condition:
185         1 of them
186 }
187
188 rule proc_x_stat {
189     meta:
190         description = "Detects reading /proc/.*/stat"
191     strings:
192         $re = /\/proc\/.{2,10}\/stat/
193     condition:
194         1 of them
195 }
196
197 rule proc_self_fd {
198     meta:
199         description = "Detects reading /proc/self/fd/"
200     strings:
201         $x1 = "/proc/self/fd/" ascii fullword
202     condition:
203         1 of ($x*)
204 }
205
206 rule proc_net_arp {
207     meta:
208         description = "Detects reading /proc/net/arp"
209     strings:
210         $x1 = "/proc/net/arp" ascii fullword
211     condition:
212         1 of ($x*)
213 }
214
215
216 rule WifiInfo_getIpAddress {
217     meta:
218         description = "Detects calls to 'WifiInfo.getIpAddress()'"
219         ref = "https://stackoverflow.com/a/6071963"
220     strings:
221         $ = "Landroid/net/wifi/WifiInfo;->getIpAddress()I" ascii
222     condition:
223         1 of them
224 }
225
226 rule Get_Local_IP_hashcode {
227     meta:
228         description = "Detects retrieving of local IP via hashcode instead of getHostAddress"
229         ref = "https://stackoverflow.com/a/10199498"
230     strings:
231         $ = "Ljava/net/NetworkInterface;->getInetAddresses()Ljava/util/Enumeration;" ascii
232         $ = "Ljava/net/InetAddress;->hashCode()" ascii
233         $ = "formatIpAddress" ascii
234     condition:
235         all of them
236 }
237
238 rule Get_Local_IP {
239     meta:
240         description = "Detects retrieving of local IP, combination of
                InetAddress_getHostAddress & NetworkInterface_getInetAddresses"
241         ref = "https://stackoverflow.com/a/13007325"
242     strings:
243         $ = "Ljava/net/NetworkInterface;->getInetAddresses()Ljava/util/Enumeration;" ascii
244         $ = "Ljava/net/InetAddress;->getHostAddress()Ljava/lang/String;" ascii
245     condition:
246         all of them
247 }
248
249 rule InetAddress_getHostAddress {
250     meta:
251         description = "Detects calls to 'InetAddress.getHostAddress()'"
252     strings:
253         $ = "Ljava/net/InetAddress;->getHostAddress()Ljava/lang/String;" ascii
254     condition:
```


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

```
255         1 of them
256     }
257
258     rule NetworkInterface_getInetAddresses {
259         meta:
260             description = "Detects calls to 'NetworkInterface.getInetAddresses()'"
261         strings:
262             $ = "Ljava/net/NetworkInterface;->getInetAddresses()Ljava/util/Enumeration;" ascii
263         condition:
264             1 of them
265     }
266
267     rule NetworkInterface_getNetworkInterfaces {
268         meta:
269             description = "Detects calls to 'NetworkInterface.getNetworkInterfaces()'"
270         strings:
271             $ = "Ljava/net/NetworkInterface;->getNetworkInterfaces()Ljava/util/Enumeration;" ascii
272         condition:
273             1 of them
274     }
275
276     rule NsdManager_discoverServices {
277         meta:
278             description = "Detects NSD scanning via NsdManager.discoverServices()"
279         strings:
280             $x1 = "Landroid/net/nsd/NsdManager;->discoverServices(" ascii
281         condition:
282             1 of ($x*)
283     }
284
285     rule PING_InetAddress_isReachable {
286         meta:
287             description = "Detects isReachable() function call often used for ping scanning"
288         strings:
289             $x1 = "Ljava/net/Inet4Address;->isReachable(" ascii
290             $x2 = "Ljava/net/InetAddress;->isReachable(" ascii
291         condition:
292             1 of ($x*)
293     }
294
295     rule UDP_DatagramSocket_send {
296         meta:
297             description = "Detects <DatagramSocket>.send() function call often used for UDP port
                    scanning"
298         strings:
299             $x1 = "Ljava/net/DatagramSocket;->send(Ljava/net/DatagramPacket;)" ascii
300         condition:
301             1 of ($x*)
302     }
303
304     rule UPnP_MulticastSocket_send {
305         meta:
306             description = "Detects <MulticastSocket>.send() function call often used for UPnP"
307         strings:
308             $x1 = "Ljava/net/MulticastSocket;->send(Ljava/net/DatagramPacket;)" ascii
309         condition:
310             1 of ($x*)
311     }
312
313     rule TCP_Socket {
314         meta:
315             description = "Detects 'Socket(ip, port)' usage, often used for TCP port scanning"
316         strings:
317             $x1 = "Ljava/net/Socket;-><init>(Ljava/net/InetAddress;" ascii
318             $x2 = "Ljava/net/Socket;->connect(Ljava/net/SocketAddress;" ascii
319         condition:
320             1 of ($x*)
321     }
322
323     rule GATEWAY_IP_1 {
324         meta:
325             description = "Detects '((WifiManager) systemService).getDhcpInfo().gateway' usage"
326         strings:
```


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

```
327         $ = "getSystemService(" ascii
328         $ = "WifiManager;->getDhcpInfo()" ascii
329         $ = "DhcpInfo;->gateway" ascii
330     condition:
331         all of them
332 }
333
334 rule GATEWAY_IP_2 {
335     meta:
336         description = "Detects '((WifiManager) systemService).getDhcpInfo().gateway' usage"
337         ref = "https://stackoverflow.com/a/30200861"
338     strings:
339         $x1 = "getSystemService(" ascii
340         $x2 = "WifiManager;->getDhcpInfo()" ascii
341         $s2 = "netmask" ascii
342     condition:
343         all of ($x*) and 1 of ($s*)
344 }
345
346
347
348 rule ifconfig_eth0 {
349     meta:
350         description = "Detects calling 'ifconfig eth0'"
351     strings:
352         $x1 = "ifconfig eth0" ascii fullword
353     condition:
354         1 of ($x*)
355 }
356
357 rule ip_neigh {
358     meta:
359         description = "Detects calling 'ip neigh'"
360     strings:
361         $x1 = "ip neigh" ascii fullword
362         $x2 = "ip neighbour show" ascii fullword
363     condition:
364         1 of ($x*)
365 }
366
367 rule PING_cmd {
368     meta:
369         description = "Detects ping command"
370     strings:
371         $ping1 = "ping" ascii fullword
372         $ping2 = "/system/bin/ping" ascii fullword
373         $s1 = "-c " ascii fullword
374         $s2 = "-W " ascii fullword
375     condition:
376         (
377             1 of ($ping*) and 1 of ($s*)
378         ) or $ping2
379 }
380
381
382
383 rule C_socket_usage{
384     meta:
385         description = "Detects socket functions used in C libs"
386     strings:
387         $x1 = "gethostbyname" ascii fullword
388         $x2 = "setsockopt" ascii fullword
389         $x3 = "gethostbyname" ascii fullword
390         $x4 = "recvfrom" ascii fullword
391         $x5 = "recvmsg" ascii fullword
392         $x6 = "inet_addr" ascii fullword
393         $x7 = "inet_ntoa" ascii fullword
394         $x8 = "socket" ascii fullword
395         $x9 = "sendto" ascii fullword
396         $x10 = "gethostbyaddr" ascii fullword
397     condition:
398         ext == ".so" and 3 of ($x*)
399 }
```


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

```
400
401 rule C_ICMP {
402     meta:
403         description = "Detects icmp string in C libs"
404     strings:
405         $ = "icmp" ascii fullword nocase
406     condition:
407         ext == ".so" and 1 of them
408 }
409
410 rule C_http_request_parts {
411     meta:
412         description = "Detects HTTP string parts in C libs"
413     strings:
414         $ = "Host:" ascii fullword
415         $ = "Cookie:" ascii fullword
416         $ = "Content-Type:" ascii fullword
417         $ = "Cookies-Set:" ascii fullword
418         $ = "Content-Length:" ascii fullword
419         $ = "Connection:" ascii fullword
420         $ = "Keep-Alive:" ascii fullword
421         $ = "Connection:" ascii fullword
422         $ = "POST" ascii fullword
423         $ = "GET" ascii fullword
424         $ = "text/plain" ascii fullword
425         $ = "text/html" ascii fullword
426     condition:
427         ext == ".so" and 3 of them
428 }
429
430
431
432 rule NSD_ServiceTypes {
433     meta:
434         description = "Detects NSD scanning"
435     strings:
436         $ = "_services._dns-sd._udp"
437         $ = "_workstation._tcp"
438         $ = "_companion-link._tcp"
439         $ = "_ssh._tcp"
440         $ = "_adisk._tcp"
441         $ = "_afpovertcp._tcp"
442         $ = "_device-info._tcp"
443         $ = "_googlecast._tcp"
444         $ = "_printer._tcp"
445         $ = "_ipp._tcp"
446         $ = "_http._tcp"
447         $ = "_smb._tcp"
448         $ = "_hap._tcp"
449         $ = "_coap._tcp"
450     condition:
451         6 of them
452 }
453
454 rule SOAP_xml {
455     meta:
456         description = "Detects SOAP xml"
457     strings:
458         $x1 = "</s:Body></s:Envelope>" ascii fullword
459         $x2 = "<s:Envelope xmlns:s=" ascii fullword
460         $x3 = "http://schemas.xmlsoap.org/soap" ascii fullword
461     condition:
462         1 of ($x*)
463 }
464
465 rule Get_Local_via_WEB {
466     meta:
467         description = "Detects retrieving of local IP via web calls"
468         ref = "https://stackoverflow.com/a/51840861"
469     strings:
470         $ = "://checkip.amazonaws.com/" ascii
471         $ = "://api.ipify.org/" ascii
472         $ = "://ipinfo.io/ip" ascii
```


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

```
473         $ = "://bot.whatismyipaddress.com" ascii
474         $ = "://ipecho.net/plain" ascii
475         $ = "ifconfig.co" ascii
476         $ = "ifconfig.me" ascii
477         $ = "icanhazip.com" ascii
478     condition:
479         1 of them
480 }
```

A.1.2 Meta Rules

```
1  rule LAN_Scanning_APK {
2      meta:
3          description = "Detects combination of rules which are most likely scanning LAN"
4      strings:
5          $ = "Hardcoded_local_IPs" ascii
6          $ = "proc_net_arp" ascii
7          $ = "GATEWAY_IP_" ascii
8          $ = "UPnP_Keyword_http" ascii
9          $ = "UPnP_Multicast_Addr" ascii
10         $ = "Scan_Keywords" ascii
11         $ = "NsdManager_discoverServices" ascii
12         $ = "PING_InetAddress_isReachable" ascii
13         $ = "PING_cmd" ascii
14         $ = "WifiInfo_getIpAddress" ascii
15         $ = "Get_Local_IP" ascii
16         $ = "Get_Local_via_WEB" ascii
17         $ = "Get_Local_IP_hashcode" ascii
18         $ = "InetAddress_getHostAddress" ascii
19     condition:
20         4 of them
21 }
```

A.1.3 Android Malware MD5 Hashes

A.2 Hashes

```
1  006ead0cabf1312dbce67ed42d524bfc
2  0294f46d0e8cb5377f97b49ea3593c25
3  037a384d211021c5cf2c1b830cdf2a4d
4  07e01c2fa020724887fc39e5c97eccee
5  0ce1648ff7553189e5b5db2252e27fd5
6  b712f4576332b7ae443abb0be0765024
7  0d5c03da348dce513bf575545493f3e3
8  103cfbc4f61dd642f9f44b8248545831
9  d33afd39b3dbb50cf7c32a38f3fb2f84
10 1e5213e02dd6f7152f4146e14ba7aa36
11 1fb9d7122107b3c048a4a201d0da54cd
12 201d55ffca3b469cf3f0a9bdc78483e3
13 805874eb17894224cb0b4dbc9bd6521d
14 2e06bbc26611305b28b40349a600f95c
15 2e1ed1f4a5c9149c241856fb07b8216b
16 2e49775599942815ab84d9de13e338b3
17 f303e8ef98e6326545838cee0105cd4e
18 3285ae59877c6241200f784b62531694
19 34a5b1b6c61d75b92476e3be2379b934
20 389c20a9a4a4aada461535ad22e0dc2a
21 3d4373015fd5473e0af73bdb3d65fe6a
22 3eb36a9853c9c68524dbe8c44734ec35
23 3fe46408a43259e400f7088c211a16a3
24 4139bc61833f61365f49dc165aab0ae5
25 6ccd8006bb11dcfc2f2bb08565d61181
26 426351383dfe8f88a0959a9d5e8c43c7
27 428c9aea62d8988697db6e96900d5439
28 43aac5543b41bc2272b590e4901bebae
29 4556ccecbf24b2e3e07d3856f42c7072
30 4626ed60dfc8deaf75477bc06bd39be7
31 46cd3890b5d6586bfcc940beb7d6bfe4
32 bf2ddaf430243461a8eab4aa1ed1e80d
```


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

```
33  49d1c82a6b73551743a12faec0c9e8b1
34  4a530c949efbe3a1c99a48c51a641f55
35  4ae13489e22c79cc794d59ff74cb1aee
36  511e2e53e79dd8818287e7a283ec1760
37  51b9d09d57365fa4e09251b0072eff1d
38  53cd72147b0ef6bf6e64d266bf3ccafe
39  54777021c34b0aed226145fde8424991
40  55a3a52a0a74fe9415a4bfd381f8e059
41  dc4fbadc2d6e0210d0ec3b99a07e0002
42  58333095cd9c36b7388901ce997baa0c
43  e173d533a004027de26222f76181daad
44  5a49013b1e49c7a5bce1755cdb36519c
45  662cf5d1eaee03c3b96200346bd66178
46  5f563a38e3b98a7bc6c65555d0ad5cfd
47  618e30a0f5aa6119ea7687399227e776
48  14d9cea1080b4ef3e41329d7fb84f70b
49  627aa2f8a8fc2787b783e64c8c57b0ed
50  62fad3ac69db0e8e541efa2f479618ce
51  65d399e6a77acf7e63ba771877f96f8e
52  66f5e93d654b6157ec296e770fcf951f
53  2cb58ccb6461e4fe22bb22c0a5f78f9e
54  6b323840d7cb55bb5c9287fc7b137e83
55  6bf9b834d841b13348851f2dc033773e
56  6c3308cd8a060327d841626a677a0549
57  6f7710e122e547d1bcd8eb496a30ec01
58  705313f75d9048531c21b3d7d123007c
59  72104cd624f9f6eb691e6b2c8a0a99a0
60  7285f44fa75c3c7a27bbb4870fc0cdca
61  73636b094276a1f918d73f94d38d4185
62  76265edd8c8c91ad449f9f30c02d2e0b
63  79f06cb9281177a51278b2a33090c867
64  80079907e8324f454977947661c48d2e
65  8008bedaaebc1284b1b834c5fd9a7a71
66  6b2d8b82efb9990b4d5e2687e4cad11d
67  83c423c36ecda310375e8a1f4348a35e
68  a492cf1a84c67ed311f4f519082956be
69  8a75b7e4075ed1c237d64940f13dbc70
70  8d5c64fdaae76bb74831c0543a7865c3
71  8d8c011ae462913386f63974bd239a60
72  2e56a5dd927f20d9306c9d9656cc5aef
73  931435cb8a5b2542f8e5f29fd369e010
74  94a3ca93f1500b5bd7fd020569e46589
75  c81e236e8e7445375ee40d8e3f327873
76  997bc9a539b2decd5bb3b3e6799f55e2
77  9c4cb389e8ef10b78b64df982bc0a032
78  9eb18198d02001614f19a9b2822dcb33
79  a097b8d49386c8aab0bb38bbfdf315b2
80  a20fc273a49c3b882845ac8d6cc5beac
81  a330456d7ca25c88060dc158049f3298
82  a57bac46c5690a6896374c68aa44d7b3
83  a795f662d10040728e916e1fd7570c1d
84  a7a07b5c9d606fbc5480ebd5acd2cf1d
85  a912e5967261656457fd076986bb327c
86  aac2942b2193cb4f011d62b1d74f7e61
87  abdef021da3fcb8082c82743c2e730ae
88  035a8b191d2e92da307b07d8e66ed91c
89  af0e580b67938afaeb783b72cf2a1c61
90  af44bb0dd464680395230ade0d6414cd
91  b107c35b4ca3e549bdf102de918749ba
92  b4706f171cf98742413d642b6ae728dc
93  b7784d5f2f4967c3dbf8f5773db11c76
94  b8006e986453a6f25fd94db6b7114ac2
95  b91491c2525b4a578a88b7a13df679aa
96  bae69f2ce9f002a11238dcf29101c14f
97  c334bade6aa52db3eccf0167a591966a
98  c36de50fe488e5015a58a241eb9b2411
99  4fe2d12c67a7f5360dd6d57ce2402e6a
100 dd4596cf68c85eb135f7e0ad763e5dab
101 cc96f03b5d13d2549304d49c4df2c3e3
102 cf71ba878434605a3506203829c63b9d
103 5d45ce9395a95f1b76ae8e40a9ef0262
104 dd772975bc0a360f7696bde8a7148a03
105 d23472f47833049034011cad68958b46
```


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

```
106   d26681348c0df5cfadd3e00a029dfe8c
107   d924211bef188ce4c19400eccb6754da
108   f2b54eda7c3e19c4e429d7adb1b7560c
109   dd0062b572261e989b4b2f47c9d194bb
110   dea4161f076a3d2b52e3c8be7a97d242
111   e5811485b2185e4cebb60425b6a63c99
112   e7fdc332b5018d5b21f05324be027f01
113   ecdf36149b3fface308607f17133a80b
114   f449cca2bc85b09e9bf4d3c4afa707b6
115   fbc117cb98053dc31d52eb677dab496c
116   fce625ece62c1ff41ea0bbfb57a521af
117   fe15c0eacdbf5a46bc9b2af9c551f86a
```

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

List of Figures

3.1 Architecture of Android app isolation. 11
3.2 TCP Three-Way Handshake. 12
3.3 Components of the Android Debug Bridge (ADB). 19

4.1 Overview of our hybrid (static and dynamic) analysis framework. 22
4.2 Architecture of the dynamic analysis testbed. 25

5.1 AliExpress Data Safety statement summary. 36
5.2 Magic Home Pro Data Safety statement summary. 38
5.3 LAN scanning activity of Magic Home Pro. 38
5.4 SSDP traffic dump of ENSPIRE Controller. 40
5.5 ENSPIRE Controller Data Safety statement summary. 41
5.6 Country distribution of IPs from the GP and IoT app dataset. 44

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

Bibliothek Your knowledge hub
TU 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.

List of Tables

4.1 Timeouts of ADB commands. 25

5.1 Hybrid Analysis Results 30
5.2 Categorized Yara rules and their matches on the app datasets 46
5.3 Categorized Yara rules and their matches on: ignored app dataset (apps which did not survive the pre-filtering; 1,457 apps), filtered app dataset (apps which did survive the pre-filtering; 794 apps), ARP scanning app dataset (apps which were found ARP scanning; 8 apps). 47
5.4 Categorized Yara rules and their matches on: ignored app dataset (apps which did not survive the pre-filtering; 1,457 apps), filtered app dataset (apps which did survive the pre-filtering; 773 apps), SSDP scanning app dataset (apps which were found SSDP scanning; 29 apps). 48
5.5 General statistics for app dataset traffic dumps. 48
5.6 Protocol statistics for app dataset traffic dumps (per dump). 50
5.7 Top 20 domains of GP and IoT dataset 50
5.8 Top 20 TLD's of GP and IoT dataset 51

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

Bibliothek Your knowledge hub TU 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.

Bibliography

[1] Gunes Acar, Danny Yuxing Huang, Frank Li, Arvind Narayanan, and Nick Feamster. „Web-based Attacks to Discover and Control Local IoT Devices“. In: *Proceedings of the 2018 Workshop on IoT Security and Privacy*. ACM SIGCOMM 2018 Conference. Association for Computing Machinery, 2018. ISBN: 978-1-4503-5905-4. DOI: 10.1145/3229565.3229568. URL: https://dl.acm.org/doi/10.1145/3229565.3229568.

[2] AliExpress. *AliExpress - Google Play Store*. URL: https://play.google.com/store/apps/details?id=com.alibaba.aliexpresshd (visited on 08/01/2022).

[3] Android. *Android NDK*. 2022. URL: https://developer.android.com/ndk (visited on 07/18/2022).

[4] Android. *Application Sandbox*. 2022. URL: https://source.android.com/security/app-sandbox (visited on 07/18/2022).

[5] Android. *class InetAddress*. 2022. URL: https://developer.android.com/reference/java/net/InetAddress (visited on 07/18/2022).

[6] Android. *Permissions on Android*. URL: https://developer.android.com/guide/topics/permissions/overview (visited on 07/18/2022).

[7] Android. *Runtime Permissions*. 2022. URL: https://source.android.com/devices/tech/config/runtime_perms (visited on 07/18/2022).

[8] Apple. *If an app would like to connect to devices on your local network*. 2020. URL: https://support.apple.com/en-us/HT211870 (visited on 07/18/2022).

[9] archive.ph. *AliExpress - Google Play Store*. URL: https://archive.ph/iWIeL (visited on 08/01/2022).

[10] archive.ph. *AliExpress - Privacy Policy*. URL: https://archive.ph/wip/vMFrh (visited on 08/01/2022).

[11] archive.ph. *ENSPIRE Controller - Googe Play Store*. URL: https://archive.ph/2Ww3O (visited on 08/01/2022).

[12] archive.ph. *Magic Home Pro - Google Play Store*. URL: https://archive.ph/udYCz (visited on 08/01/2022).

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.

[13] archive.ph. *Privacy Policy Of MAGIC HOME*. URL: https://archive.ph/wip/yjckE (visited on 08/01/2022).

[14] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. „FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps". In: *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '14: ACM SIGPLAN Conference on Programming Language Design and Implementation. ACM, June 9, 2014. ISBN: 978-1-4503-2784-8. DOI: 10.1145/2594291.2594299. URL: https://dl.acm.org/doi/10.1145/2594291.2594299.

[15] Alexandre Bartel, Jacques Klein, Yves Le Traon, and Martin Monperrus. „Dexpler: converting android dalvik bytecode to jimple for static analysis with soot". In: *Proceedings of the ACM SIGPLAN International Workshop on State of the Art in Java Program analysis*. 2012.

[16] Bleepingcomputer. *Chinese Advertising SDK Caught Stealing Data From Android Devices*. URL: https://www.bleepingcomputer.com/news/security/chinese-advertising-sdk-caught-stealing-data-from-android-devices/ (visited on 08/01/2022).

[17] Elias Bou-Harb, Mourad Debbabi, and Chadi Assi. „Cyber Scanning: A Comprehensive Survey". In: *Proceedings of IEEE Communications Surveys Tutorials*. 2014. DOI: 10.1109/SURV.2013.102913.00020.

[18] Michael Brengel and Christian Rossow. „{YARIX}: Scalable YARA-based Malware Intelligence". In: *Proceedings of the 30th USENIX Security Symposium (USENIX Security 21)*. 2021.

[19] LED Controller. *Magic Home Pro - Google Play Store*. URL: https://play.google.com/store/apps/details?id=com.zengge.wifi (visited on 08/01/2022).

[20] LED Controller. *Privacy Policy Of MAGIC HOME*. URL: https://faqsys.magichue.net:4489/zengge_privacy_policy_en.html (visited on 08/01/2022).

[21] Apache Cordova. *Apache Cordova*. 2022. URL: https://cordova.apache.org/ (visited on 07/18/2022).

[22] Yamaha Corporation. *ENSPIRE Controller - Googe Play Store*. URL: https://play.google.com/store/apps/details?id=com.yamaha.dkv.enspire (visited on 08/01/2022).

[23] Lucas DiCioccio, Renata Teixeira, Martin May, and Christian Kreibich. „Probe and Pray: Using UPnP for Home Network Measurements". In: *Proceedings of International Conference on Passive and Active Network Measurement*. 2012. ISBN: 978-3-642-28536-3 978-3-642-28537-0. DOI: 10.1007/978-3-642-28537-0_10. URL: http://link.springer.com/10.1007/978-3-642-28537-0_10.

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.

[24] Emmanuel Dupuy. *JD-GUI - Java Decompiler*. 2022. URL: http://java-decompiler.github.io/ (visited on 07/18/2022).

[25] Dhia Farrah and Marc Dacier. „Zero Conf Protocols and their numerous Man in the Middle (MITM) Attacks“. In: *Proceedings of the 2021 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2021.

[26] Googe. *Android Debug Bridge (adb)*. 2022. URL: https://developer.android.com/studio/command-line/adb (visited on 07/18/2022).

[27] Googe. *GoPacket - Provides packet processing capabilities for Go*. 2021. URL: https://github.com/google/gopacket (visited on 07/18/2022).

[28] Googe. *UI/Application Exerciser Monkey*. 2022. URL: https://developer.android.com/studio/test/other-testing-tools/monkey (visited on 07/18/2022).

[29] Muhammad A. Hakim, Hidayet Aksu, A. Selcuk Uluagac, and Kemal Akkaya. „U-PoT: A Honeypot Framework for UPnP-Based IoT Devices“. In: *Proceedings of the 37th International Performance Computing and Communications Conference (IPCCC)*. Proceedings of the 37th International Performance Computing and Communications Conference (IPCCC). ISSN: 2374-9628. 2018. DOI: 10.1109/PCCC.2018.8711321.

[30] https://github.com/csicar. *Ning - Network-Scanner for Android*. URL: https://github.com/csicar/Ning (visited on 08/01/2022).

[31] https://www.reddit.com/user/CruZer000/. *Why does AliExpress need access to my local network???* URL: https://www.reddit.com/r/Aliexpress/comments/m1u2en/why_does_aliexpress_need_access_to_my_local/ (visited on 08/01/2022).

[32] Center for Internet Security. *EternalBlue*. 2019. URL: https://www.cisecurity.org/wp-content/uploads/2019/01/Security-Primer-EternalBlue.pdf (visited on 07/18/2022).

[33] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. „Fingerprinting the Fingerprinters: Learning to Detect Browser Fingerprinting Behaviors“. In: *Proceedings of the 2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021. DOI: 10.1109/SP40001.2021.00017.

[34] Fox IT. *FAQ on the WanaCry ransomware outbreak*. 2017. URL: https://blog.fox-it.com/2017/05/13/faq-on-the-wanacry-ransomware-outbreak/ (visited on 07/18/2022).

[35] JesusFreke. *smali/baksmali is an assembler/disassembler for the dex format*. 2022. URL: https://github.com/JesusFreke/smali (visited on 07/18/2022).

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.

[36] Dhruv Kuchhal and Frank Li. „Knock and Talk: Investigating Local Network Communications on Websites“. In: *Proceedings of the 21st ACM Internet Measurement Conference*. Association for Computing Machinery, 2021. ISBN: 9781450391290. DOI: 10.1145/3487552.3487857. URL: https://doi.org/10.1145/3487552.3487857.

[37] Renuka Kumar, Apurva Virkud, Ram Sundara Raman, Atul Prakash, and Roya Ensafi. „A Large-scale Investigation into Geodifferences in Mobile Apps“. In: *Proceedings of the 31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, 2022. ISBN: 978-1-939133-31-1. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/kumar.

[38] Anh Le, Janus Varmarken, Simon Langhoff, Anastasia Shuba, Minas Gjoka, and Athina Markopoulou. „AntMonitor: A system for monitoring from mobile devices“. In: *Proceedings of the 2015 ACM SIGCOMM Workshop on Crowdsourcing and Crowdsharing of Big (Internet) Data*. 2015.

[39] Fing Limited. *Fing - Netzwerk-Scanner*. URL: https://play.google.com/store/apps/details?id=com.overlook.android.fing (visited on 08/01/2022).

[40] MITRE. *MITRE ATT&CK (registered trademark)*. 2022. URL: https://attack.mitre.org/ (visited on 07/18/2022).

[41] Alibaba Mobile. *AliExpress - Privacy Policy*. URL: https://terms.alicdn.com/legal-agreement/terms/suit_bu1_aliexpress/suit_bu1_aliexpress201909171350_82407.html (visited on 08/01/2022).

[42] Abhinav Mohanty and Meera Sridhar. „HybriDiagnostics: Evaluating Security Issues in Hybrid SmartHome Companion Apps“. In: *Proceedings of the 2021 IEEE Security and Privacy Workshops (SPW)*. IEEE. 2021.

[43] Amogh Pradeep, Alvaro Feal, Julien Gamba, Ashwin Rao, Martina Lindorfer, Narseo Vallina-Rodriguez, and David Choffnes. „Not Your Average App: A Large-scale Privacy Analysis of Android Browsers“. In: *Proceedings of the Privacy Enhancing Technologies Symposium (PETS) (to appear)*. 2023.

[44] Abbas Razaghpanah, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Christian Kreibich, Phillipa Gill, Mark Allman, and Vern Paxson. „Haystack: In situ mobile traffic analysis in user space“. In: *arXiv preprint arXiv:1510.01419*. 2015.

[45] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. „50 Ways to Leak Your Data: An Exploration of Apps' Circumvention of the Android Permissions System“. In: *Proceedings of the 28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, 2019. ISBN: 978-1-939133-06-9. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/reardon.

[46] Joel Reardon, Nathan Good, Robert Richter, Narseo Vallina-Rodriguez, Serge Egelman, and Quentin Palfrey. „JPush Away Your Privacy: A Case Study of Jiguang's Android SDK“. In: (2020).

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.

[47] Jingjing Ren, Martina Lindorfer, Daniel J. Dubois, Ashwin Rao, David Choffnes, and Narseo Vallina-Rodriguez. „Bug Fixes, Improvements, ... and Privacy Leaks - A Longitudinal Study of PII Leaks Across Android App Versions“. In: *Proceedings of the 2018 Network and Distributed System Security Symposium*. NDSS, 2018. ISBN: 978-1-891562-49-5. DOI: 10.14722/ndss.2018.23143.

[48] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Arnaud Legout, and David Choffnes. „ReCon: Revealing and Controlling PII Leaks in Mobile Network Traffic“. In: *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '16. Association for Computing Machinery, 2016. ISBN: 9781450342698. DOI: 10.1145/2906388.2906392. URL: https://doi.org/10.1145/2906388.2906392.

[49] Thomas Reps, Susan Horwitz, and Mooly Sagiv. „Precise interprocedural dataflow analysis via graph reachability“. In: *Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 1995.

[50] Franziska Roesner, Tadayoshi Kohno, and David Wetherall. „Detecting and Defending Against Third-Party Tracking on the Web“. In: *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. USENIX Association, 2012. ISBN: 978-931971-92-8. URL: https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/roesner.

[51] Shanto Roy, Nazia Sharmin, Jamie C. Acosta, Christopher Kiekintveld, and Aron Laszka. „Survey and Taxonomy of Adversarial Reconnaissance Techniques“. In: *Proceedings of ACM Computing Surveys (CSUR)*. 2021. URL: https://arxiv.org/abs/2105.04749.

[52] Android Runtime. *Android Runtime*. URL: https://source.android.com/docs/core/runtime (visited on 08/01/2022).

[53] David Schmidt. „Large-scale Static Analysis of PII Leakage in IoT Companion Apps“. MA thesis. TU Wien, 2021. URL: https://doi.org/10.34726/hss.2021.86548.

[54] Vijay Sivaraman, Dominic Chan, Dylan Earl, and Roksana Boreli. „Smart-Phones Attacking Smart-Homes“. In: *Proceedings of WiSec'16: 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. Association for Computing Machinery, 2016. ISBN: 978-1-4503-4270-4. DOI: 10.1145/2939918.2939925. URL: https://dl.acm.org/doi/10.1145/2939918.2939925.

[55] Kansas State University & University of South Florida. *Argus-SAF*. 2021. URL: https://github.com/arguslab/Argus-SAF (visited on 07/18/2022).

[56] statista. *Number of available apps in the Apple App Store from 2008 to 2022*. 2022. URL: https://www.statista.com/statistics/268251/number-of-apps-in-the-itunes-app-store-since-2008/ (visited on 07/18/2022).

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

[57] statista. *Mobile operating systems' market share worldwide from January 2012 to January 2022*. 2022. URL: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/ (visited on 07/18/2022).

[58] statista. *Number of available applications in the Google Play Store from December 2009 to March 2022*. 2022. URL: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/ (visited on 07/18/2022).

[59] statista. *Number of smartphones sold to end users worldwide from 2007 to 2021*. 2021. URL: https://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/ (visited on 07/18/2022).

[60] Blake E Strom, Andy Applebaum, Doug P Miller, Kathryn C Nickels, Adam G Pennington, and Cody B Thomas. *MITRE ATT&CK (registered trademark): Design and Philosophy*. 2020. URL: https://attack.mitre.org/docs/ATTACK_Design_and_Philosophy_March_2020.pdf (visited on 07/18/2022).

[61] Blake E Strom, Joseph A Battaglia, Michael S Kemmerer, William Kupersanin, Douglas P Miller, Craig Wampler, Sean M Whitley, and Ross D Wolf. *Finding Cyber Threats with ATT and CK (registered trademark)-Based Analytics*. 2017. URL: https://www.mitre.org/publications/technical-papers/finding-cyber-threats-with-attck-based-analytics (visited on 07/18/2022).

[62] Raja Vallée-Rai, Etienne Gagnon, Laurie Hendren, Patrick Lam, Patrice Pominville, and Vijay Sundaresan. „Optimizing Java bytecode using the Soot framework: Is it feasible?“ In: *Proceedings of the International conference on compiler construction*. Springer. 2000. ISBN: 978-3-540-46423-5.

[63] VirusTotal. *Yara - The pattern matching swiss knife*. 2022. URL: https://github.com/VirusTotal/yara (visited on 07/18/2022).

[64] Xueqiang Wang, Yuqiong Sun, Susanta Nanda, and XiaoFeng Wang. „Looking from the Mirror: Evaluating IoT Device Security through Mobile Companion Apps“. In: *Proceedings of 28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, 2019. ISBN: 978-1-939133-06-9. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/wang-xueqiang.

[65] Fengguo Wei, Sankardas Roy, Xinming Ou, and Robby. „Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps“. In: *Proceedings of ACM Transactions on Privacy and Security*. ACM, 2018. DOI: 10.1145/3183575. URL: https://dl.acm.org/doi/10.1145/3183575.

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

[66] Haohuang Wen, Qingchuan Zhao, Qi Alfred Chen, and Zhiqiang Lin. „Automated Cross-Platform Reverse Engineering of CAN Bus Commands from Mobile Apps“. In: *Proceedings of the 27th Annual Network and Distributed System Security Symposium (NDSS'20)*. NDSS, 2020. DOI: 10.14722/ndss.2020.24231. URL: https://dx.doi.org/10.14722/ndss.2020.24231.

[67] Connor Tumbleson & Ryszard Wiśniewski. *apktool - A tool for reverse engineering Android apk files*. 2022. URL: https://ibotpeaches.github.io/Apktool/ (visited on 07/18/2022).

[68] Daoyuan Wu, Debin Gao, Rocky K. C. Chang, En He, Eric K. T. Cheng, and Robert H. Deng. „Understanding Open Ports in Android Applications: Discovery, Diagnosis, and Security Assessment“. In: *Proceedings 2019 Network and Distributed System Security Symposium*. NDSS, 2019. ISBN: 978-1-891562-55-6. DOI: 10.14722/ndss.2019.23171. URL: https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_06B-5_Wu_paper.pdf.

[69] Daoyuan Wu, Debin Gao, Robert H. Deng, and Chang Rocky K. C. „When Program Analysis Meets Bytecode Search: Targeted and Efficient Inter-procedural Analysis of Modern Android Apps in BackDroid“. In: *Proceedings of the 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2021. DOI: 10.1109/DSN48987.2021.00063.

[70] www.lookout.com. *Igexin advertising network put user privacy at risk*. URL: https://www.lookout.com/blog/igexin-malicious-sdk (visited on 08/01/2022).

[71] Qingchuan Zhao, Chaoshun Zuo, Brendan Dolan-Gavitt, Giancarlo Pellegrino, and Zhiqiang Lin. „Automatic Uncovering of Hidden Behaviors From Input Validation in Mobile Apps“. In: *Proceedings of 2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020. ISBN: 978-1-72813-497-0. DOI: 10.1109/SP40000.2020.00072. URL: https://ieeexplore.ieee.org/document/9152205/.

[72] Chaoshun Zuo, Zhiqiang Lin, and Yinqian Zhang. „Why Does Your Data Leak? Uncovering the Data Leakage in Cloud From Mobile Apps“. In: *Proceedings of the 2019 IEEE Symposium on Security and Privacy*. 2019.

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.