SECURITY ANALYSIS OF SELECTED ANDROID TV BOX

BEZPEČNOSTNÍ ANALÝZA VYBRANÉHO ANDROID TV BOXU

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR
AUTOR PRÁCE
Bc. ADAM ŠVENK

SUPERVISOR
VEDOUCÍ PRÁCE
Ing. MAREK TAMAŠKOVIČ

BRNO 2024

Master's Thesis Assignment

155989

Institut:	Department of Intelligent Systems (DITS)
Student:	**Švenk Adam, Bc.**
Programme:	Information Technology and Artificial Intelligence
Specialization:	Cybersecurity
Title:	**Security Analysis of Selected Android TV Box**
Category:	Security
Academic year:	2023/24

Assignment:

1. Explore and describe the current state of the art in firmware analysis, forensic methods. Next, describe possible vulnerabilities in Android devices especialy in TV Boxes.
2. Choose a popular Android TV Box with suitable parameters (manufacturer, USB connection, Android version) for analysis.
3. Do security analysis that may include more than just these techniques:
    1. Capture external network traffic at least 72 hours (between the Internet and the device) and then analyze the network traffic. Identify if unexpected communication was taking place in the captured traffic and analyse it.
    2. Identify and analyse the running services on the device, list and identify potential vulnerabilities. Then try to use the identified vulnerabilities to exploit those weak spots.
    3. Test the device web management according to the OWASP Mobile Application Security.
    4. Inspect the device without a protective cover and identify the debug interface. If found, describe it and test if the interface is functional and connect through it to the device. Then analyse the running services and find any vulnerabilities and exploit them.
    5. Analyse if the device is prone to network attacks on multiple layers of TCP/IP stack (at least 5 for IPv4 and IPv6).
    6. Analyse the firmware used on the device and try to find vulnerabilities (outdated software versions, password security, remote access, ...) using reverse engineering technique.

4. Review the security of the device, enumerate the severity and likelihood of found vulnerabilities and propose how to mitigate them.

Literature:
- https://mas.owasp.org/
- https://github.com/scriptingxss/owasp-fstm
- HOU, Qinsheng, et al. Large-scale security measurements on the android firmware ecosystem. In: *Proceedings of the 44th International Conference on Software Engineering*. 2022. p. 1257-1268.
- CAM, Nguyen Tan, et al. Detect malware in android firmware based on distributed network environment. In: *2019 IEEE 19th International Conference on Communication Technology (ICCT)*. IEEE, 2019. p. 1566-1570.
- MCLAUGHLIN, Niall, et al. Deep android malware detection. In: *Proceedings of the seventh ACM on conference on data and application security and privacy*. 2017. p. 301-308.

Requirements for the semestral defence:
1,2 and two parts of step 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Tamaškovič Marek, Ing.**
Head of Department:	Hanáček Petr, doc. Dr. Ing.
Beginning of work:	1.11.2023
Submission deadline:	17.5.2024

Approval date:	6.11.2023

Abstract

The popularity of Android TV boxes has increased significantly in recent times. In addition to offering a wide range of functionality, the question of whether they are adequately secured is becoming increasingly pertinent. This thesis performs a comprehensive security analysis of selected Android TV boxes, covering both the hardware and software components. By examining the vulnerabilities present in the device, this thesis aims to identify potential risks to user privacy and security. Additionally, it proposes recommendations to mitigate these vulnerabilities.

Abstrakt

Popularita TV boxov so systémom Android v poslednom čase výrazne vzrástla. Okrem toho, že ponúkajú širokú škálu funkcií, je čoraz aktuálnejšia otázka, či sú dostatočne zabezpečené a chránené. Táto práca popisuje komplexnú bezpečnostnú analýzu vybraného Android TV boxu, ktorá zahŕňa hardvérové aj softvérové komponenty. Skúmaním zraniteľností prítomných v zariadení sa táto práca zameriava na identifikáciu potenciálnych rizík pre súkromie a bezpečnosť používateľov. Okrem toho navrhuje odporúčania na zmiernenie týchto zraniteľností.

Keywords

security analysis, Android, Android TV, OWASP, vulnerability, firmware, reverse engineering

Kľúčové slová

bezpečnostná analýza, Android, Android TV, OWASP, zraniteľnosť, firmvér, reverzné inžinierstvo

Reference

ŠVENK, Adam. *Security Analysis of Selected Android TV Box.* Brno, 2024. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Marek Tamaškovič

Rozšírený abstrakt

Popularita TV boxov so systémom Android v poslednom čase výrazne vzrástla. Okrem toho, že ponúkajú širokú škálu funkcií, je čoraz aktuálnejšia otázka, či sú dostatočne zabezpečené a chránené. Táto práca popisuje komplexnú bezpečnostnú analýzu vybraného Android TV boxu, ktorá zahŕňa hardvérové aj softvérové komponenty. Skúmaním zraniteľností prítomných v zariadení sa táto práca zameriava na identifikáciu potenciálnych rizík pre súkromie a bezpečnosť používateľov. Okrem toho navrhuje odporúčania na zmiernenie týchto zraniteľností. Prvá časť tejto práce, teoretická, je venovaná samotnému operačnému systém Android, z ktorého operačný systém Android TV vychádza. Obsahuje úvod do sveta mobilných zariadení, po ktorom nasleduje analýza architektúry samostatného operačného systému. Po nej nasleduje detailnejší popis architektúry so zameraním sa na samotnú bezpečnosť a použité bezpečnostné mechanizmy, zraniteľnosti a systém bezpečnostných opráv. Druhou časťou teoretickej časti je popis metód bezpečnostného testovania. Na úvod sú uvedené štandardizované metriky, ktoré sú používané na sledovanie životného cyklu zraniteľností. Na záver sa teoretická časť zameriava na samotné metódy a postupy testovania bezpečnosti zariadení spoločne so softvérovými nástrojmi uľahčujúcimi analýzu. Tieto postupy, procesy a metodológie predstavujú kľúčové elementy pre nasledujúcu praktickú časť. Druhá časť práce, praktická, obsahuje popis vykonania samotnej bezpečnostnej analýzy. Na začiatku je prezentované vybrané zariadenie, Mecool KM6 Deluxe, ktoré bolo analyzované. Sú popísané jeho parametre, softvérové, aj hardvérové. Prvú časť praktickej časti bezpečnostnej analýzy tvorí analýza zachytenej komunikácie. Na zachytenie komunikácie bol použitý jednodoskový počítač Raspberry Pi. Zachytená komunikácia je následne analyzovaná manuálne s pomocou softvérových nástrojov. Ďalšia časť testuje náchylnosť zariadenia na útoky smerujúce na TCP/IP zásobník. Zariadenie bolo otestované viacerými typmi útokov, pričom niektoré z nich boli úspešné a dokázali ovplyvniť správanie zariadenia, ako aj jeho bezpečnosť. Analyzované zariadenie bolo následne rozobraté s cieľom lokalizovať ladiace rozhrania. Na zariadení bolo úspešne otestované ladiace rozhranie UART, ktoré odkrylo samotnú inicializáciu softvéru. Tá odhalila možnosť jej modifikácie. Okrem ladiaceho rozhrania bola na zariadení lokalizovaná dióda prijímajúca infračervené žiarenie, ktorú môže útočník zneužiť. Nasleduje analýza služieb bežiacich na zariadení. Táto analýza navrhla potencionálne zraniteľnosti, avšak ich overenie nebolo možné. Jednou zo služieb, ktorej zraniteľnosť bola úspešne overená, je Bluetooth. Predposlednou častou analýzy bola analýza predinštalovanej aplikácie na zariadení, ktorej výsledkom bolo primerané aplikovanie bezpečnostných opatrení. Poslednou časťou analýzy bola komplexná analýza firmvéru. Táto analýza odhalila viacero potencionálnych bezpečnostných nedostatkov, ktoré vychádzali najmä z použitia zastaraných verzií použitých softvérov. Okrem toho však definuje veľké množstvo zraniteľností, ktoré nie je z viacerých dôvodov možné prakticky overiť. Posledná časť práce sumarizuje nájdené zraniteľnosti, ktoré sú ohodnotené pomocou štandardizovaných metrík, spoločne s odporúčaniami ako ich zmierniť, prípadne úplne odstrániť. Cieľom tejto práce je vykonanie bezpečnostnej analýzy vybraného Android TV boxu. Analýza bola prakticky vykonaná pomocou definovaných procesov a metodík v teoretickej časti. Jednotlivé časti analýzy sa zameriavajú na špecifické vrstvy tvoriace komplexný operačný systém, akým je Android TV. Nájdené zraniteľnosti boli otestované, diskutované, spoločne s odporúčaniami na ich odstránenie, či zmiernenie.

Security Analysis of Selected Android TV Box

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Ing. Marek Tamaškovič. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

. . . .
Adam Švenk
May 17, 2024

Acknowledgements

I want to thank my master's thesis supervisor Ing. Marek Tamaškovič for providing guidance together with advice.

Contents

Chapter 1

Introduction

Smart products and intelligent technologies dominate the current state of the world. Devices that a few years ago performed one specific function are now capable of performing several functions at the same time. This trend also created smart TV boxes that transformed traditional televisions into smart entertainment hubs that changed the way of content consumption. They provide extensive functionality, including the possibility of extending it by installing additional applications. However, this smartness brings with it certain negative qualities. The use of the Android TV operating system reduces the cost of software development and reduces the time required to develop a new product for device manufacturers. Nevertheless, it is uncertain whether this expedited approach to development also carries potential drawbacks, such as overlooking the security of the device, which should be among the first priorities during development. In the current state of computer technology, the number of computer systems connected to the Internet is rapidly enlarging. Due to the increasing complexity of these systems, this contributes to more vulnerabilities present on those devices, although creating a vulnerability-free system can be considered impossible. In production software, it can be expected to have 1-25 errors per 1000 lines of code for the software delivered [26]. This thesis aims to perform a comprehensive security analysis of a selected smart TV box based on an Android TV operating system. The architecture of the Android TV operating system is first described in Chapter 2 which provides a theoretical background. Next, the security testing methodology is explained in Chapter 3. Afterwards, the selected Android TV box's security is analysed in Chapter 4 with a summary of discovered vulnerabilities.

Chapter 2

Android TV Box

This chapter describes the architecture of the Android TV Box both from the hardware and software point of view. The Android TV operating system is based on the Android operating system, which is described first.

2.1 Android Operating System

Android is currently one of the major operating systems used on both mobile devices and embedded platforms. Since its initial commercial version release in 2008, it has evolved into a fully featured operating system. Due to its relatively low hardware requirements and modularity, Android can run on various types of devices: mobile phones, tablets, notebooks, TV boxes, smart TVs, smart watches, smart speakers, automobiles, and many others. In December 2023, 70.48% of mobile phones were using Android as their operating system, surpassing iOS by 41.68%, thus making Android dominant [32]. It can be said with certainty that its popularity is also since it was designed and developed to be open-source. This is true for the Android Open Source Project (AOSP)[1], which is free and open-source software (FOSS) developed and maintained by Google. In addition to the fact that AOSP is free and open-source software, most devices run a proprietary version of Android, which is built upon the Google closed-source flavour comprising Google Mobile Services (GMS), one of many closed-source software pre-installed. Every device manufacturer can customize the Google-provided Android operating system image. However, this also affects the security, which is then heavily dependent on the device manufacturers [21]. Regardless of the continuous and significant effort of Google in collaboration with device manufacturers, Android devices have become the perfect target device for malware development [12].

2.1.1 Architecture

Understanding the basics of the Android platform architecture is a necessary step for faster and better vulnerability detection and understanding. The main components of the Android platform are shown in Figure 2.1.

- Linux Kernel: Linux kernel layer contains drivers that provide abstract access at a higher level to multiple hardware components together with process management, memory management, file system access, networking and communication access, power management, security management, etc., as in any Unix system [30]. The Android

[1] `https://source.android.com/`

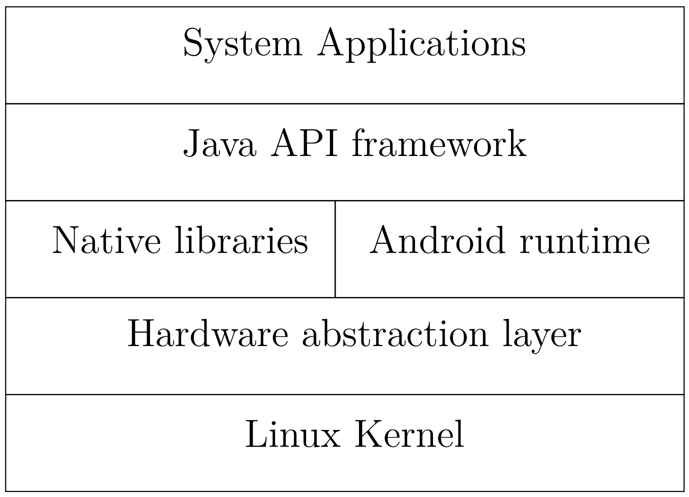

Figure 2.1: Android Software Stack

kernel is based on an open-source, upstream Linux Long Term Supported (LTS) kernel with additional Android Common Kernels (ACKs), which are provided by Google. Every device vendor has the option to include manufacturer-specific kernel modules to provide device-specific functionality.

- Hardware abstraction layer: The hardware abstraction layer (HAL) builds additional higher-level functionality on top of the Android Kernel itself. As the name suggests, it provides an abstraction layer that interfaces with the kernel and provides unifying hardware interfaces for user-space processes [37].

- Android runtime: Each application running on the Android operating system is running in its own process and in its own instance of the Android Runtime (ART) [1].

- Native libraries: Although most of the applications made for the Android operating system are written in Java and Kotlin programming languages, the Android operating system relies on numerous core components and services that are built using native code, which necessitates the use of libraries written in C and C++. The functionality of those components is exposed by APIs from the Java and Kotlin frameworks.

- Java API framework: The application developers have the entire feature set of the Android operating system available to them through building blocks, which are provided to them in the form of APIs, which are written in the Java programming language. Those features are also used by the system apps themselves.

- System applications: Android is equipped with a collection of essential applications for tasks such as email clients, SMS messaging, calendars, Internet browsers, and contacts. These applications do not have special privileges compared to the applications installed by the end user. That implies that user-installed applications like Internet browsers, messengers, and keyboards can become the default web browser, SMS messenger, or even the default keyboard. However, there are certain exceptions, such as the System Settings application. System applications serve a dual purpose, functioning as both user applications and providers of essential functionality blocks that developers can use in their own developed, custom applications.

2.1.2 Security and Privacy

The nature of desktop or server operating systems is distinct from that of mobile devices and their operating systems. This implies that security threats for mobile devices are different in their nature, due to the fact, that the vast majority of devices powered by the Android platform are mobile and thus, easily physically accessible. This is also true for Android devices like cars, smart TVs, and TV boxes, that can be accessible in close proximity. There are multiple architectural improvements from the Android developers team to increase the security of the Android operating system. Those started back in 2015 in the form of monthly security bulletins. However, the heavily customisable nature of the Android operating system results in different approaches from device manufacturers and attitudes towards security [15]. Now, Google releases Android Security Bulletin[2] every month. Every Bulletin includes Android platform fixes, upstream Linux kernel fixes and fixes from SoC manufacturers. The security patches are then distributed and made available to device manufacturers. Nevertheless, the distribution of the security patches is up to each device manufacturer. Many device manufacturers do not release over-the-air (OTA) updates to devices in a reasonable timeframe [23]. Some do not even release the updates at all, or they will just change the update date without actually releasing the new security patch [21].

Android Kernel Security

The Android kernel, the lowest layer of the Android layered architecture, provides basic security features that are taken advantage of throughout the Android system. Linux, the operating system of origin of the Android kernel, is an operating system that supports multiple user accounts and groups. In a Linux system, it is not possible for one user to access the files of another user unless explicit permission has been granted. Each process, that is run, has its own identity: the user and group ID, also referred to as UID and GID of the user, that started the process. Those low-level security features, coming from desktop Linux systems, are also implemented in the Android operating system. However, the applications, each running in its own, isolated, process, have to communicate with other applications. The Android operating system implements secure inter-process communication to provide this functionality [6].

Android Application Security

Each application that is installed on Android is given a unique UID, also called an app ID, that also identifies the process that is responsible for the application execution with the same UID value. The applications are distributed in packages in .apk file format. In addition, each application is given a dedicated data directory, which only has permission to read and write data. As a result, the applications are isolated, and sandboxed, at both the process and the file levels. This establishes a kernel-level sandbox for applications, irrespective of whether they are executed in a native or virtual machine process. This basically implies that the Android application sandbox prohibits one application resource from being used by another [17]. By default, applications have a limited range of accessible resources provided by the Android operating system. Access to additional types of resources is protected by protected APIs. Those APIs provide access to cameras, location data (GPS), local storage, Bluetooth functionality, and many others. For granting access to those APIs, the Android

[2] https://source.android.com/docs/security/bulletin

operating system implements a permission model. If the running application requires access to the protected APIs, the Android operating system prompts the user to deny or allow permission to use the APIs [2].

2.2 Android TV Operating System

The current TV market has reached the point where it is almost impossible to buy a new TV that does not have any smart functionality. Smart TVs are becoming more popular because they provide additional software that goes beyond the capabilities of legacy (conventional) TVs, such as browsing the Internet, playing games, watching Netflix, Twitch, YouTube, or other platforms that provide streaming of content, and many others, thus creating a central hub for home entertainment [13]. Most of the currently available smart TVs also allow users to install additional applications. Android TV operating system has emerged as one of the most popular smart TV platforms in the market due to its numerous advantages derived from the widely used Android ecosystem. Similarly to Android smartphones, Android TV users can access the Google Play Store to update installed applications or even download and install new ones, as well as to use Google Assistant for convenient hands-free operations [25].

2.2.1 Vulnerabilities

From a hardware perspective, the primary distinction between devices running the Android TV operating system and Android smartphones and tablets lies in their focus on design and development. Unlike smartphones and tablets, which prioritise size and mobility, Android TV devices are, in most cases, not space-constrained. This circumstance implies that these devices are simpler to handle physically, as they can be effortlessly disassembled, examined, or reverse-engineered. However, since the Android TV operating system is based on the Android operating system and is part of the Android platform, in addition to sharing features, it also shares the same weaknesses. That means that security testing of an Android TV operating system-powered embedded device is practically the same as testing of any Android-powered device. However, the Android TV operating system is extended by providing specific applications and services that have to be analysed and tested for vulnerabilities.

Chapter 3

Security Testing

This chapter describes the theoretical methodology of security testing from a general perspective. First, vulnerability tracking is described. Next, penetration testing is introduced as an analysis method. The methods of security testing, network analysis, hardware analysis, mobile application security, and firmware analysis follow next, together with a commonly used software tool for each of the analysis stages.

3.1 Vulnerability Tracking

Vulnerability tracking is an essential part of modern-day security testing. Together with the discovery, it is also important to manage the whole life cycle of the vulnerability. Active tracking of vulnerabilities is a good practice to maintain high cyber-safety standards and overall security. There exist multiple standards and metrics to provide a complex approach to this task. Common Vulnerability Scoring System (CVSS) is a widely adopted and used system that provides a method to capture key characteristics of software, hardware, and firmware vulnerabilities. It requires several metrics, which are used to calculate the numerical score value. This score represents a comprehensive assessment of the risk and severity of a vulnerability and can be used to compare it with other vulnerabilities. In addition to the CVSS score value, it is also represented in the form of a vector string representing the input values used in the calculation. It represents a significant part of numerous vulnerability scanners. Common Vulnerabilities and Exposures (CVE) is a database of publicly known vulnerabilities and exposures. Each CVE entry has a unique identifier, the CVE ID, which is used to identify and reference vulnerabilities, together with additional information about the vulnerability, vulnerable severity, affected hardware of software versions, and vulnerable configuration. The CVE database is maintained by a non-profit organization, the MITRE[1] [7]. Common Platform Enumeration (CPE) is a standardised method used for the identification of hardware and software products. It is a part of the Security Content Automation protocol (SCAP), proposed by the National Institute of Standards and Technology (NIST). It provides a structured naming scheme that allows for consistent identification of systems and their components to efficiently manage vulnerabilities, track hardware and software assets, and others [29]. Those mentioned methods provide a unified set of metrics that are applied in the vulnerability tracking process and security testing.

[1] https://cve.mitre.org/

3.2 Penetration Testing

Penetration testing, also known as ethical hacking, is one of the key methods used to analyse the security of all types of computer systems. It can be defined as a legal and authorised attempt to locate vulnerabilities and successfully exploit computer systems for the purpose of making those systems more secure and, thus, better and more reliable. The process includes probing for vulnerabilities and providing proof of concept (POC) attacks to demonstrate that the vulnerabilities are real. Well-executed penetration testing does not end only with a list of vulnerabilities found but also offers specific recommendations for addressing and repairing the issues that were discovered during the test. On the whole, this process is used to help secure computer systems and networks against future attacks. The general idea is to find security issues using the same tools and techniques as an attacker [18]. The penetration testing is a structured process that usually consists of the following parts:

- Pre-engagement: Before actual penetration testing can begin, it is crucial to define the target objectives of the penetration testing.
- Information gathering: This phase covers the analysis of freely available sources of information, a process known as open source intelligence (OSINT).
- Threat Modelling: Thanks to the information collected from the previous phase, the penetration tester can develop plans of attack. These attacks describe strategies to penetrate the system being tested.
- Vulnerability analysis: Pentenster began to do the vulnerability discovery based on their plans of attack. The quality of the attack plans corresponds directly to the quality of the vulnerabilities discovered.
- Exploitation: After the vulnerabilities have been discovered, the pentester actively exploits them in an attempt to access the system.
- Post-exploitation: Before the final phase, the pentester gathers information about the attacked system, attempts to elevate privileges, and attempts to gather as much sensitive, valuable, information as possible.
- Reporting: All findings have to be properly documented, including the penetration testing processes themselves. The pentest report should include both a summary of findings and a technical report [36].

3.3 Network Analysis

Network analysis is a crucial part of security analysis in the modern world, in which almost every embedded device is part of some network. This connection, besides extending the functionality of the embedded devices, might also introduce possible vulnerabilities and attack vectors. The analysis can be, for simplicity, divided into two stages:

- Passive: Passive analysis consists of capturing the network communication between the analysed device and the network. The captured communication is analysed to discover potential vulnerabilities, unencrypted communication, or others, to evaluate the security measures of the analysed device.

- Active: Active analysis consists of active, penetration, and testing of the whole network stack implemented on the analysed device. The network stack is usually on multiple underlying layers, and each might contain vulnerabilities that have to be tested individually. Usually, it analyses if the device is susceptible to various types of denial-of-service (DoS) attacks, man-in-the-middle (MITM) attacks, and many others.

3.3.1 Software Tools

Wireshark

Wireshark[2] is an open-source software tool used for network packet capture and extensive analysis. It supports capturing packets from various available network interfaces present on the running machine. It shares some similarities with a popular software tool, tcpdump[3], however, it also provides a graphical user interface (GUI) and a complex approach to analysis. It supports packet inspection, filtering, generating statistics, and various information about captured network traffic that can be valuable.

Hping3

Hping3[4] is an open-source software tool used in network analysis. It can display packet routes, scan open ports, test firewalls, fingerprint the victim's operating system, audit the TCP/IP stack, and especially send custom-made, crafted, and spoofed ICMP, UDP, and TCP packets.

Bruteshark

bruteshark[5] is a Network Forensic Analysis Tool (NFAT) that performs deep processing and inspection of network traffic. It is capable of reconstructing TCP and UDP sessions, extracting passwords, extracting hashes of encrypted passwords, and even converting them to a Hashcat format to be cracked later on.

Nmap

Nmap[6], short for short for Network Mapper, is an open-source software tool used for network analysis and evaluation. It supports open port scanning with service detection, operating system detection thanks to fingerprinting, service version detection, and many others. In summary, it is a complex software tool used for security audit of devices and systems.

3.4 Hardware Analysis

Security testing combines multiple approaches to test the security of the system. While security testing focused on the software part of the tested device usually dominates the analysis, the hardware analysis should not be neglected and deserves the same attention and focus. It applies especially during the device's development stage. In contrast to developing secure software, which can be patched and fixed later during the device life

[2] https://www.wireshark.org/
[3] https://www.tcpdump.org/
[4] https://github.com/antirez/hping
[5] https://github.com/odedshimon/BruteShark
[6] https://github.com/nmap/nmap

cycle as vulnerabilities are discovered, this principle could not be applied to the hardware. Embedded devices, in general, are easily accessible at the physical level. Neglecting the hardware security creates possible attack vectors that can be leveraged by the attacker and is usually almost impossible for the device manufacturer to fix easily [22]. Besides exposing peripheral interfaces meant to be used by the end users, the embedded devices might also expose interfaces used for development and debugging purposes. The most commonly used debug interfaces, especially in embedded devices built around ARM architecture, are JTAG and UART [34]. After that, the hardware layer of the device provides comprehensive access to the device that can expose hardware components hidden by the protective cover.

3.4.1 UART

UART, the full name of which is the Universal Asynchronous Receiver Transmitter, is one of the most widely used interfaces in embedded systems [35]. The UART converts parallel bytes of data that are fed from the transmitter system to serial bit streams for outbound transmission. In the receiving system, serial bit streams are converted back to parallel bytes and handled by the system [28]. The UART interface consists of the transmitter, receiver, and baud rate generator. The baud rate generator divides the system clock and provides a bit clock [28], which is used to generate all clock signals within the transmitter and receiver systems. The baud rate indicates the number of bits transmitted through the UART interface itself, not the number of bits of actual data transmitted. This includes the bits used for an indication of the start and end of the transmission and the parity bits [11]. The UART interface does not utilise a shared common clock, so the baud rate must be agreed upon by both the transmitting and receiving systems before the initial communication [31]. An improperly configured baud rate can lead to data corruption, communication errors, loss of synchronisation, or garbage data. The values of commonly used baud rates are 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800 and 921600 [8]. However, the transmitting and receiving hardware can support different, not commonly used, values of the baud rate in addition to the ones used in general.

3.4.2 Software Tools

PuTTY

The PuTTY[7] is an open-source software tool besides supporting SSH and Telnet protocol as a client, also supports monitoring of serial ports. It does support the configuration of the custom value of baud rate, saving sessions, logging to text files and others.

3.5 Open Worldwide Application Security Project

The Open Worldwide Application Security Project (OWASP) is a non-profit organisation that was launched on December 1, 2001. In summary, the primary goal of all OWASP activities is to continuously improve software security. OWASP community is dedicated to enabling organisations to conceive, develop, acquire, operate, and maintain applications that can be trusted. All of their projects, tools, documents, forums, and chapters are free and open to anyone interested in improving the security of the applications. Thanks to that, they are the largest non-profit organisation concerned with software security.

[7] https://putty.org/

3.6 OWASP Mobile Application Security

Thanks to the wide adoption of mobile technologies and devices in recent years, it is necessary to define and recognise the different characteristics and features of applications developed for mobile platforms in contrast to desktop and web applications. The mobile platforms implement different approaches in case of permissions, protection of local data, client-server transactions, etc. Most of the code developed for the mobile platform runs on the end-user devices. However, the developer does not control the mobile platform's runtime environment. This leads to unique security risks, vulnerabilities, and threats that the developer is faced with and must prepare for. The OWASP's project, Mobile Application Security, provides tools, the OWASP Mobile Application Security Verification Standard (MASVS) and the OWASP Mobile Application Security Testing Guide (MASTG), that provide guidelines, techniques, and processes for both the development and security testing of mobile applications. The crucial part of the whole OWASP Mobile Application Security suite is based on defining key areas in mobile application security:

- Data storage and Privacy: Protecting sensitive data, like user credentials and private information, is a crucial principle in mobile security. Mobile applications must implement the proper use of security APIs, notably those used for handling local storage or inter-process communication. In the case of improper implementation, the mobile application can expose or leak sensitive data. This might benefit the attacker since mobile devices are easily physically accessible and might be lost or stolen. The developer should use available key storage APIs and hardware-backed security features. This is a disadvantage of the Android platform, thanks to the wide variety of devices and supported platforms. It is advised to develop mobile applications to use the newest available APIs and security features, not focusing on supporting the widest variety of devices, neglecting security.

- Cryptography: In protecting and storing sensitive data, cryptography plays an essential role. The mobile application should use proven libraries that provide cryptographic operations. It is even more critical that the cryptographic primitives are appropriately chosen and configured with a sufficient random number generator in case randomness is needed.

- Network Communication: The mobile devices connect to various types of networks, including non-protected public Wi-Fi networks. Those networks are usually shared with other users and devices, creating an entry point for the attacker to perform various types of attacks. Thus, mobile applications must implement security features to create an encrypted network communication channel to provide confidentiality and integrity between the communicating devices.

- Interaction with the Mobile Platform: The mobile operating systems, Android and iOS, create a layer between the applications and the system environment. This layer regulates the application's access to system APIs and inter-process communication (IPC) resources. In the case of misuse of APIs or IPCs, mobile applications could expose sensitive data that is meant to be protected.

- Code Quality and Exploit Mitigation: In traditional, non-mobile systems, memory management issues represent a significant threat. In the case of mobile systems, those types of issues usually don't pose a significant weakness. The mobile systems implement a sandboxing mechanism in which the running applications primarily interact with trusted backend services and the user interface. However, this does not mean that application developers should not focus on proper memory management.

- Anti-Tampering and Anti-Reversing: The code obfuscation methods increase the application's resilience against reverse engineering, tampering, and other types of attacks, possibly resulting in the extraction of valuable intellectual property or sensitive data. However, not implementing those methods does not automatically make the applications more vulnerable. Those methods should not replace security mechanisms, although they can be implemented with a clear purpose to add a layer of security to the applications.

3.6.1 Software Tools

JADX

JADX[8] is an open-source software tool used to decompilate Android applications. It reverse engineers and reproduces source code written in the Java programming language from Android Dex and Android .apk formatted files. The version with a graphical user interface (GUI) also provides an integrated development environment (IDE) together with the option to export the decompiled source code of the application as Gradle[9] project.

APKHunt

APKHunt[10] is an open-source analysis software tool used to perform complex static analysis. The analysis process is based on the OWASP Mobile Application Security Verification Standard (MASVS).

3.7 OWASP Firmware Security Testing Methodology

One of their comprehensive guidelines, part of the OWASP Internet of Things (IoT)[11] project, the OWASP Firmware Security Testing Methodology (FSTM), defines several stages of the recommended process for the firmware analysis process. Firmware analysis is a thorough examination and evaluation of the software in embedded devices, the main objective of which is to examine the code for vulnerabilities, security gaps, and potential weak spots that could be exploited. It is based on various techniques, creating a complex methodology that combines methods such as reverse engineering, static analysis, and dynamic analysis, to disassemble and understand the inner architecture of the firmware. The OWASP Firmware Security Testing Methodology can be summarised and described in the following stages:

1. Information gathering and research: First of all, it is necessary to gather as much information about both hardware and software to gain knowledge about the target system and its overall composition and underlying technology. It is possible to

[8] `https://github.com/skylot/jadx`
[9] `https://gradle.org/`
[10] `https://github.com/Cyber-Buddy/APKHunt`
[11] `https://owasp.org/www-project-internet-of-things/`

gather data using multiple techniques, including open-source intelligence (OSINT) techniques, especially in cases where open-source software is used.

2. Obtaining firmware: In the second step, the obtaining of the firmware image of the system is required. Although obtaining the firmware image might seem as hard at first, there are multiple methods that can lead to the firmware acquisition, such as directly downloading the firmware image directly from the system vendor's support website, man-in-the-middle (MITM) attack during on-device firmware updates, extracting firmware directly from hardware via UART, JTAG or other debug interfaces, removing the flash chip from the system's board and extracting it's content using flash storage chip programmer and many others. The proper method has to be selected appropriately according to the characteristics of the analysed device.

3. Initial firmware analysis: Firmware analysis is a crucial part of the process. It can leverage software utilities such as `file`, `strings`, `Binwalk`, `hexdump` or others, which can provide important, valuable information about the firmware file type, potential filesystem metadata, encryption, or gain additional parts of knowledge that can lead to a better understanding of the platform and firmware itself.

4. Extracting the filesystem: In most cases, the firmware is in the form of a binary file. Every filesystem has its own signatures, which can lead to proper detection of the used filesystem. The filesystem can be extracted by previously mentioned software utilities.

5. Firmware analysis: The contents of the filesystem might contain a lot of clues and even valuable data that can be gathered for later analysis stages. During this stage, static analysis is performed, leveraging software utilities like Firmwalk, FACT, EMBA, and others. The firmware files might contain information about the insecure software used in the firmware, hardcoded credentials, API and SSH keys, API endpoints, backdoors, and many other clues that can prove to be valuable in the further analysis process.

6. Emulating firmware: Thanks to the obtained firmware and clues about its architecture and various pieces of information, the firmware, as well as its encapsulated binaries, can be emulated to verify potential vulnerabilities detected in previous analysis stages. There are a few approaches, such as partial emulation of the standalone binaries or emulation of the entire firmware image on either a virtual machine or an actual physical device.

7. Dynamic analysis: While the firmware runs on a virtual machine or an actual physical device, dynamic testing is to be performed. The objectives may vary according to the target system and access level and can define various paths for dynamic analysis. This frequently involves tampering with hardware or various parts of software.

8. Runtime analysis: While the software binary or process is running on a real physical device or a virtual machine, other software can be attached to it and analyse behaviour.

9. Binary Exploitation: After successful vulnerability identification is performed, it is necessary to create a working proof of concept (PoC) to demonstrate the possibility of taking advantage of the particular vulnerability [20] with possible countermeasures to mitigate the abuse by the attacker.

Overall, the OWASP Firmware Security Testing Methodology provides a comprehensive approach to firmware analysis combining multiple techniques.

3.7.1 Software Tools

Binwalk

Binwalk[12] is an open-source software tool used to analyse the firmware image, especially of embedded systems. Besides the analysis performed on the filesystem level, it provides another important feature: the extraction of the filesystem from the firmware image itself. This feature predisposes it to be used in more complex software tools and analysers as the software tool used for filesystem extraction, which is a crucial part of the whole firmware analysis process.

Firmwalker

Firmwalker[13] is an open-source static analyser that aims to search for password files (`passwd`, `shadow`, `.psk`), SSL-related files (`.crt`, `.pem`, `.cer`, `.p7b`, `.p12b`, `.key`), certificates, keys, configuration files, script files, database files, binary files, keywords (admin, password, remote, AWS), familiar web servers, common binaries (SSH, TFTP, Dropbear, etc.), banned C functions, common command injection vulnerable functions, URLs, email addresses, IP addresses, and many others. It is designed as a bash script that can be easily extended. It does not perform firmware extraction automatically; the input firmware has to be extracted before the analysis using Binwalk or a similar software tool.

EMBA

EMBA[14] is an open-source security analyser focusing on embedded device firmware. It is built upon multiple software tools that are used to create one tool that combines different approaches to firmware analysis, static and dynamic. Thanks to that, EMBA is a versatile software tool. The aim is to be easy to use, versatile, modular, and user-customisable without losing the ability to identify weaknesses. The analysis process of the EMBA software tool can be summarised into the following stages:

1. Pre-analysis stage: The EMBA does utilise modules whose primary task is to extract the files and archives and These modules will be loaded and called first. Their main goal is to extract firmware from files and archives to gather as much initial information about the files and binaries as possible.

2. Core analysis stage: After the firmware extraction and initial basic analysis, the core modules are used to enumerate and check possible vulnerabilities and security measures and collect possibly valuable clues and pieces of information. The currently deployed modules are optimised Linux-based systems.

3. Live-testing stage: The core analysis modules are used to perform static analysis. On the other hand, the live-testing modules emulate the embedded device's system.

[12]https://github.com/ReFirmLabs/binwalk
[13]https://github.com/craigz28/firmwalker
[14]https://www.securefirmware.de/

4. Finishing stage: After static and dynamic analysis, the results must be aggregated, summarised and visualised. Multiple modules are used to summarise the licensing information, aggregate CVEs, and create a summary in the form of a website.

It is also interesting to see an implementation of AI in one of the newly developed modules, which uses Open-AI's API to send queries to the ChatGPT language model to extend the explanation of the results in the modules used. In summary, the EMBA provides a complex analysis software tool that can be modified to the user's needs.

Chapter 4

Security Analysis

This chapter describes the analysis performed. First, the selected device is presented and described. Next, the analysis follows. The first stage of the analysis consists of capturing the wireless network traffic communication between the analysed device and the Internet and analyzing the captured communication. The second stage focuses on network stack analysis After that, the hardware analysis follows. Running services are analysed and followed by application testing. The analysis is finished with a firmware analysis.

4.1 Selected Device

I chose the Mecool KM6 Deluxe Android TV Box from the devices I had the option to choose from. The Mecool KM6 Deluxe is one of the best-equipped Android TV boxes available to date. The Mecool KM6 Deluxe package contains the power adapter supplying 5V and 2A, HDMI to HDMI cable, remote controller, and TV box itself. From a connectivity point of view, Mecool KM6 Deluxe comes equipped with an HDMI port, used for video and audio output, AV output in the form of a 3.5mm jack, optical audio output, USB 2.0 and USB 3.0 interfaces, Micro SD card slot, and coaxial power connector used to power the device. In addition to wireless connectivity, Mecool KM6 Deluxe supports Bluetooth and Wi-Fi. The box is powered by an Amlogic S905X4 SoC paired together with 4 GB of RAM and 64 GB of eMMC flash storage. It comes preinstalled with the Android TV 10 operating system. The overall analysed system specifications can be seen in Table 4.1.

Brand	Mecool
Model	KM6 Deluxe
SoC	Amlogic S905X4
CPU	4x ARM Cortex-A55 @ 1.91 GHz
Architecture	ARMv8.2-A
RAM	4 GB DDR4
Storage	64 GB eMMC
Operating System	Android TV 10.0
Android security patch level	September 5, 2020
Google Certified	Yes
Ethernet	Yes, 1000M

Table 4.1: Selected Device Specifications

4.2 Network Traffic Analysis

The objective of the first step of the Android TV box security analysis is to capture all network traffic between the device and the Internet for at least 72 hours. First, the setup of hardware and software created to capture the network traffic is described. The captured network traffic is then analysed by the Wireshark and BruteShark software tools.

4.2.1 Capture Setup

As mentioned in Subsection 4.1, the Mecool KM6 utilises Wi-Fi wireless networking connectivity besides having the physical RJ45 connector with Ethernet support. This led to a setup with a wireless network interface. I decided to use the single-board computer (SBC) Raspberry Pi 4 Model B with 2GB of RAM, which is powerful enough to function as a router and a wireless access point. The Raspberry Pi 4 Model B has a wireless network adapter onboard, so there is no need to use another wireless network adapter. As the Raspberry Pi Foundation recommends, I flashed the image with Raspberry Pi OS (previously called Raspbian) to the 128GB microSDXC memory card. The Raspberry Pi OS comes with a kernel, which does support the wireless network adapter out of the box. Furthermore, the functionality of a wireless router was achieved using RaspAP[1] software utility, which creates a wireless access point, as can be seen in Figure 4.1. RaspAP software was then used to configure the wireless access point, which uses the built-in wireless network interface `wlan0`. However, this is not enough to capture all the network traffic. To capture all packets that go through the `wlan0` interface, additional software was installed, tcpdump[2], which is one of the most popular software used among network administrators [19]. The network traffic capture on the `wlan0` wireless interface was then initiated.

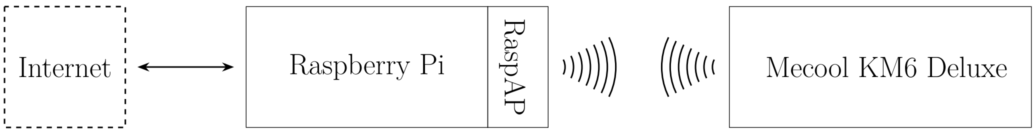

Figure 4.1: Raspberry Pi Capture Configuration

During the capture of network communication, the Xiaomi TV Stick 4K was used to simulate the usage of an average user. This mixed usage contained listening to music, watching videos and live streams using pre-installed YouTube applications, browsing the Internet, using Google Assistant, checking over-the-air (OTA) updates and others. The captured traffic characteristics can be seen in Table 4.2.

Device	Number of packets	Size (Bytes)	Duration (Hours)
Mecool KM6 Deluxe	5592257	5666716622	93

Table 4.2: Characteristic of the Captured Traffic

[1]https://raspap.com/
[2]https://www.tcpdump.org/

4.2.2 Captured Network Traffic Analysis

The resulting captured communication file in pcap file format contained captured communication between the device and the Internet for 93 hours. The initial analysis was performed using the Wireshark software tool. First, I analysed the statistical hierarchy of protocols. I did not discover any protocol that would be unexpected or interesting in the captured communication. Although no unexpected communication had been found yet, I started manually exploring communication protocols that do not utilise encryption mechanisms. During the manual analysis, I discovered the HTTP POST request to the over-the-air (OTA) updates server with data about the Android TV operating system version, device manufacture, model, MAC address, firmware version and others. I focused on HTTP POST requests and also discovered that the device sends some data that seems to be encrypted to a server with the same domain name as the over-the-air (OTA) update server. It can be assumed that the server is owned by the device manufacturer. Searching multiple forums, I found that the data transmitted are probably metadata about the usage of the Mecool KM6 Deluxe. The statistics of the hierarchy of protocols also revealed the type of communication protocols that were used. Almost 70.0% of the captured communication consisted of the GQUIC protocol, which is like the Android TV operating system also developed and widely deployed by Google [24]. The GQUIC protocol uses the UDP transport protocol, and the traffic is encrypted, thus revealing no valuable information. Another protocol that uses UDP as a transportation protocol, which is present in the captured communication, is DNS. DNS queries revealed multiple domain names that the Mecool KM6 Deluxe was looking for. The transport protocol TCP had the largest representation among the protocols, with 23.1% of all captured packets. However, almost all of the captured TCP communication was encrypted using the TLS security protocol. The packet capture file was then processed by the BruteShark software tool. The BruteShark analysis, besides aggregating already obtained DNS-resolved names, also detected usernames and passwords from HTTP Basic Authentication methods used in HTTP GET requests. All the detected usernames and passwords were the same. I filtered out those packets in Wireshark and followed all their streams. All of those HTTP GET requests were sent to various websites. I proceeded with finding the responses to those requests and discovered that all of the responses contained only plain text with the public IPv4 address of the tested Mecool KM6 Deluxe. However, I was able to replicate those requests without using HTTP Basic Authentication methods and providing any username and password. Those requests probably do not represent a security problem.

4.2.3 Summary

First, the setup consisting of a Raspberry Pi 4 single-board computer together with appropriate software tools was used to capture wireless network traffic. Next, the captured network traffic was manually analysed by Wireshark. I discovered HTTP POST requests to the manufacturer's over-the-air (OTA) updates server to be present in the captured traffic. Besides that, I also discovered multiple HTTP POST requests containing metadata about the device usage sent to the manufacturer's server. However, a substantial majority of the captured network traffic was encrypted. I continued the network traffic analysis with the BruteShark software tool. It revealed multiple uses of the HTTP Basic Authentication method, which does not use any encryption, thus revealing usernames and passwords. Later analysis described in Subsection 4.2.2 revealed that those requests are not leaking potentially valuable or sensitive data. Overall, the Mecool KM6 Deluxe network traffic analysis did not reveal any possible vulnerabilities or weak spots.

4.3 Network Stack Analysis

This section describes the analysis performed to test whether the device is prone to network attacks on multiple layers of the TCP/IP stack.

4.3.1 SYN Flood Attack

The SYN flood attack targets the transport layer of the TCP/IP stack. During the attempt to establish a TCP connection, the connection initialising system, the client, begins with sending a SYN-type message to a server system. After that, the server system sends a SYN+ACK message to the client system, confirming the initial SYN message from the client system. This is also referred to as the half-open connection. The last part of TCP connection establishment is then done by sending an ACK-type message to the server system [9]. The multistateness of the TCP creates a potential opportunity for a DoS attack. The attacker can send multiple SYN messages, which puts the victim system in the state of waiting for the SYN+ACK message for each initialised TCP connection [16]. In this direct approach to the attack, the victim system might start blocking the SYN messages simply by creating a firewall rule. However, the attacker might use a more complex approach by spoofing the source IP address of the SYN message packet [14]. The analysis was performed by sending large amounts of spoofed packets with SYN messages and randomly generated IP addresses to the Mecool KM6 Deluxe, successfully creating the SYN Flood Attack. The animations of the Android TV operating system were sluggish, but the system did not crash. This was due to the high CPU load, which was also checked in `top`. The applications installed that did not use any network connectivity worked without notable issues. However, applications that used network connectivity became practically unusable. Targeting specific services using opened ports on the device resulted in the same behaviour. This means that Mecool KM6 Deluxe does not utilise protection against SYN Flood Attacks, and legitimate user services are deprived, resulting in a denial-of-service (DoS).

4.3.2 ACK Flood Attack

The ACK Flood Attack is similar to the attack previously tested in Section 4.3.1. Unlike the previous attack, in which the attacker sends SYN messages, creating large amounts of half-open connections, in the case of the ACK Flood Attack, the attacker sends SYN+ACK messages. Those messages have spoofed values and do not belong to any current TCP sessions on the victim system. In theory, this can lead to the exhaustion of system resources. The analysis procedure was similar to that in Subsection 4.3.1 besides modifying the message type to SYN+ACK. The device behaved similarly, with a little slowdown, and applications and services that used network connectivity became unusable, resulting in a denial-of-service (DoS).

4.3.3 Ping of Death Attack

Another type of attack, the Ping of Death Attack, is based on the ICMP protocol. The ICMP stands for Internet Control Message Protocol. It is used to send operational, control, issue, and error messages across devices in the network, for example, if a host cannot be reached. It supports multiple types of messages, one of which is `Echo Request` and `Echo Reply`, which are used by the software utility `ping` to test the host's reachability. The attack consists of sending an ICMP message with a type set to `Echo Request` (ping), which exceeds the maximum allowed size of 65,535 bytes. The attacker's system automatically fragments the malicious packet. On the other hand, the victim's system automatically reassembles the fragments back. The core of the attack is that the fragments do not contain any information about the total size of the packet. During the reassembly, the victim's system does not know how many of the packet fragments are missing, thus exceeding the allocated size, resulting in freezing, crashing, or rebooting the victim's system. This attack dates back to 1996 when it was first demonstrated. As already mentioned, the attack is based on creating a specific ICMP message packet that exceeds the maximum allowed size. After sending the malformed packet to the Mecool KM6 Deluxe, it showed no symptoms of being prone to this attack.

4.3.4 ICMP Flood Attack

As the ICMP protocol is already described in the Subsection 4.3.3, it can be used for sending request messages to the network devices. Besides crafting malicious `Echo Request` packets, the ICMP messages can also be used to congest the device network buffers and exhaust system resources intentionally. To test the device, I flooded it with ICMP `Echo Request` messages. The device behaved practically the same as the already performed SYN Flood Attack in Subsection 4.3.1, and ACK Flood Attack in Subsection 4.3.2; the Android TV operating system's animations became sluggish, and the application that requires networking connectivity became essentially unusable due to exhaustion of system resources and network buffers, resulting in denial-of-service (DoS).

4.3.5 LAND Attack

The LAND attack consists of creating a TCP SYN message that has both the source and destination IP addresses and ports set to the victim's system. In theory, if the system is vulnerable, it will reprocess the packet in the infinite loop, consuming system resources, resulting in system freezing, crashing, or rebooting up. The attack was created by sending a crafted packet with spoofed IP addresses and ports to the Mecool KM6 Deluxe. The device did not show any symptoms of being prone to this attack; it probably filtered out the malicious packet.

4.3.6 ARP Spoofing Attack

The ARP Spoofing Attack is one of the ways to perform man-in-the-middle (MITM) or denial-of-service (DoS) attacks. It targets the ARP (Address Resolution Protocol), which is used in the IPv4-enabled devices within the LAN to identify themselves based on their MAC addresses. The ARP protocol supports two types of messages: request and response. Those messages are used to resolve the MAC address, which is at the link layer, based on the provided IPv4 address. As of its nature, the ARP protocol is unauthenticated and stateless. This creates a possible vulnerability exploitable by the attacker. However, this type of attack requires the attacker to have access to the LAN. The attacker sends spoofed ARP messages and poisons the ARP tables of the victim's devices. By this, the attacker has the ability to intercept the traffic and effectively perform an MITM attack [33]. The ARP Spoofing Attack was performed by sending spoofed ARP messages to the Mecool

KM6 Deluxe. The device itself did not show any symptoms of the ARP spoofing, unlike the system performing the attack. The transmitted communication from the device can be observed and analysed, possibly revealing sensitive and valuable information, especially from non-encrypted communication. To make the device not prone to the attack, it could implement ARP poisoning detection and prevention, long-term leased IPv4 address and MAC address mapping table and voting, or encrypt all the network traffic [27].

4.3.7 Summary

The Mecool KM6 Deluxe was tested against multiple network attacks targeting various TCP/IP stack layers. The analysis showed that the device does not implement any security mechanisms against various tested types of flood attacks, resulting in sluggish performance and denial-of-service (DoS) attacks. However, it is resilient against the Ping of Death and LAND attacks. Another tested attack, ARP Spoofing, does not affect the device directly; however, it can compromise security by intercepting the network traffic and creating a base layer upon which other MITM attacks can be built. In summary, the device showed varying levels of being prone to different attack strategies, highlighting the need for strong network security strategies.

4.4 Hardware Analysis

The main objective of this analysis task is to remove the protective cover of the Mecool KM6 Deluxe and disassemble the device to localise debug interfaces. In the first stage, the printed circuit board (PCB) analysis is performed. Next, the UART debug interface is localised and connected to a USB-to-UART bridge. Possible vulnerabilities are then discussed and practically demonstrated. In addition to the presence of a UART debug interface, another possible vulnerability is also analysed.

4.4.1 Disassembly Process

The Mecool KM6 Deluxe has a protective case made of two parts: one made of ABS plastic and one made of metal, which is at the bottom of the device and also serves as the base of the device. The metal base has four feet, each in the corner. There are four hidden screws under each foot with Philips flat head type (crosshead). After the screws and bottom cover are removed, the printed circuit board can be removed from the case and proceed with further analysis.

4.4.2 Printed Circuit Board Analysis

It is worth mentioning that the printed circuit board has traces for a protective shield designed to protect internal electronic components from electromagnetic interference (EMI) and radio frequency interference (RFI), but the shield is not installed on the printed circuit board. There is only an aluminium cooling heat sink to cool the SoC with thermal adhesive between the heat sink and the SoC itself. All components mounted on the printed circuit board are easily visible, as can be seen in Figure 4.2. During the printed circuit board analysis, I discovered multiple surface-mounted pads with no components mounted on them. I discovered four pads with labels printed beside them with the following text: `GND`, `RX`, `TX` and `3.3V`. As mentioned later in Subsection 4.4.3, those pins are used by the serial communication interface UART. In addition to the labelled surface-mounted pads, I also discovered a button mounted directly behind the 3.5mm jack connector that is used for the A/V output, which can be seen in Figure 4.2. Experimenting with the devices, I discovered that if the button is pressed and the device is powered on, the Mecool KM6 Deluxe boots directly into Android Recovery. Another interesting component worth mentioning found on the printed circuit board is the IR receiving diode, which is located on the front side of the Mecool KM6 Deluxe.

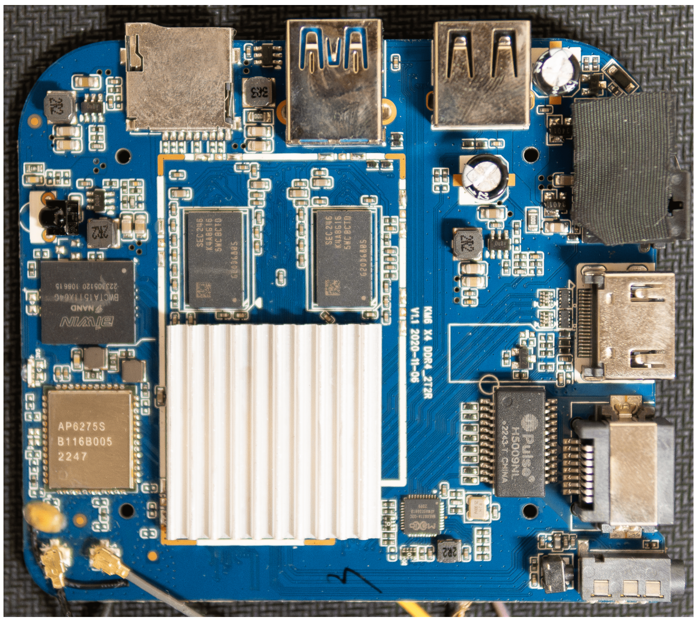
Figure 4.2: Disassembled Mecool KM6 Deluxe

4.4.3 Debug Interface

For connecting to the UART communication interface, I used a module based on the Silicon Labs CP2102 integrated circuit, which is a single-chip USB-to-UART bridge [10]. As can be seen in Figure 4.3, the UART pins are only in the form of surface-mounted pads on the printed circuit board. As a result of this, two wires were required to be soldered to the RX and TX pads with a DuPont-type connector on the other side. Then, an additional wire was soldered for the GND signal to the ground plane on the printed circuit board. After connecting the soldered wires to the USB-to-UART bridge module with DuPont-type connectors, the UART console was opened by the `Putty`[3] software. Testing the standard baud rates mentioned in Section 3.4.1 revealed that the UART communication does not use any of the standard baud rates. Upon trying various values of the baud rate, I found that the Mecool KM6 Deluxe does use a baud rate of 1,000,000. I captured the UART communication after plugging the power plug into the device, which resulted in a clean boot from the eMMC flash storage. Analysis of the communication captured revealed that Mecool KM6 Deluxe uses the vendor-specific U-Boot bootloader based on open source `U-Boot 2019.01`. After initialising the bootloader, the KM6 Deluxe boots the Android TV operating system and provides shell access as `root`. Upon initialising the basic hardware functionality, the automatic boot process can be interrupted by sending any key. I was

[3] https://www.putty.org/

able to successfully stop the automatic boot sequence by sending a carriage return (CR) and a line feed (LF) characters (`\r\n`) simply by hitting the `Enter` key repeatedly during the boot sequence of the device. After that, the U-Boot command shell is launched and accessible. This provides the attacker with various ways to attack the KM6 Deluxe, as is mentioned in Subsection 4.4.4.

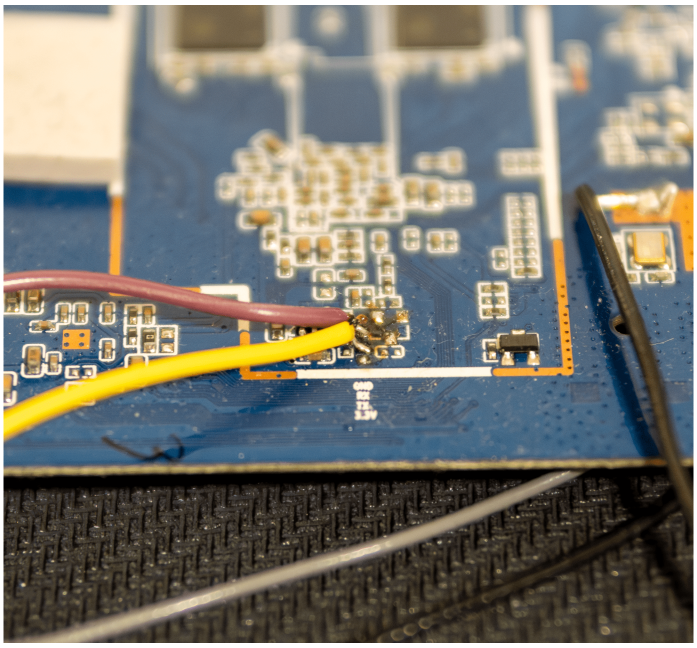

Figure 4.3: UART Solder Pads

4.4.4 U-Boot Exploitation

U-Boot is a bootloader that is widely used in various types of embedded devices. The bootloader provides a bridge between the isolated local resources to data that is located on another type of storage (flash storage, SD cards, USB flash drives, etc.). From a security standpoint, the goals of the bootloader are to boot verified images, secure trusted computing, prevent anti-rollback, and initialise the kernel securely. After accessing the U-Boot command shell, there are various ways to attack the system. The attacker can read and modify memory, resulting in access to passwords, encryption keys, system logs, or other types of sensitive data. In addition to that, the attacker can alter the boot process, replace the boot image with a modified one, and boot the device from other sources like the internal flash memory, etc. The attacker can get access to the environmental variables that I tested. I discovered a potentially interesting one that sets the kernel and boot images together with additional arguments to boot Android Recovery from flash memory. By that, I was able to alter the boot process and boot the device to Android Recovery from the U-Boot command shell without pressing the recovery button mentioned in Subsection 4.4.2. After booting Android Recovery, I got an Android shell as the root user. In addition to reading the environmental variables, the attacker can also modify them, which I also tried to test and demonstrate.

One of many security features of the Android TV operating system is the use of Security-Enhanced Linux (SELinux). This enforces mandatory access control (MAC) over all processes, even processes running with root/superuser privileges (Linux capabilities). In general, SELinux operates on the principle of default denial: Anything not explicitly allowed is denied [5]. In global, SELinux can operate in two modes: `Permissive` mode and `Enforcing` mode. In `Permissive` mode, all permission denials are logged but not enforced. On the other hand, in the `Enforcement` mode, all permission denials are logged and also enforced. The mode in which SELinux is running can be checked by the command `getenforce`. As expected, the KM6 Deluxe has the SELinux set to `Permissive` mode. As mentioned in Section 4.4.3, after booting the operating system, the UART provides access to the Android TV terminal logged as `root` user. However, due to the policy set and enforced by SELinux, the root user is limited in comparison to the system with no enforced SELinux policy. The root user cannot get access to the root shell, elevate user privileges, get access to system files, etc. One of the workarounds for bypassing this limitation is to change the operating mode of SELinux. The mode can be changed while the operating system is running using the command `setenforce` with the argument `Permissive`. However, this could not be enforced correctly because the enforced policy denies it. Due to access to the U-Boot bootloader and the possibility of working with environmental variables, the attacker might be able to modify the kernel command-line parameters. As mentioned in Subsection 4.4.3, I stopped the automatic boot sequence. Next, I modified the kernel command-line parameter by adding the argument `androidboot.selinux=permissive` [3], which sets the SELinux working mode to `Permissive`. After booting up the Android TV operating system, the running mode was checked, which was successfully set to `Permissive`. This shows that manipulation with environmental variables in the U-Boot bootloader gives the attacker another possible attack vector. The device vendor could at least implement password protection in the U-Boot bootloader command shell, making tampering with the devices more difficult.

4.4.5 IR Remote Exploitation

As mentioned in Subsection 4.4.2, there is an IR receiver diode mounted on the printed circuit board. During the initial setup of the device, the initial part requires the remote controller to be paired and connected to the Mecool KM6 Deluxe using Bluetooth wireless technology. This is done by pressing a specific key combination on the remote controller while the Mecool KM6 Deluxe periodically searches for the remote controller. After the initial setup is completed, the remote controller can be removed from the list of paired Bluetooth devices. It can be assumed that after the remote controller is removed, the device cannot be operated by it. However, in addition to using Bluetooth wireless technology, the Mecool KM6 Deluxe, together with its remote controller, utilises the IR emitting diode and IR receiver. This functionality comes in handy in multiple cases; for example, during Android Recovery, in which the Bluetooth stack is not loaded, the KM6 Deluxe can be controlled by the remote controller with the IR emitting diode. During device testing, I discovered that after making a Bluetooth connection between the Mecool KM6 Deluxe and its remote controller, the remote controller does not utilise the IR emitting diode and, therefore, entirely relies on the use of Bluetooth wireless technology. Nevertheless, the KM6 Deluxe can still be controlled with another remote controller that utilises an IR-emitting diode. I was able to test it with an Android smartphone that utilises an IR-emitting diode. In the first part, the KM6 Deluxe remote controller was used as an IR emitter together with an HX1838 IR receiver connected to an Arduino UNO development board. To use the HX1838 as an IR receiver and to decode signals, the `IRremote`[4] library was used. This results in creating a list of data that is sent from the remote controller using an IR emitting diode. This data can then be used as a payload on the Android smartphone and sent to the KM6 Deluxe. I was able to successfully control the KM6 Deluxe using an Android smartphone with an IR emitting diode built in. This allows the attacker to gain access to the device, enable the Android Debug Bridge (adb), install malicious software, or use many other features that are available by using the Android Debug Bridge. It would be a good practice to disable the IR receiver on the KM6 Deluxe when the remote controller is connected using Bluetooth wireless technology, which will not provide the attacker with a possible attack vector.

4.4.6 Summary

I connected to the exposed UART interface on the Mecool KM6 Deluxe with a USB-to-UART bridge. The UART shell does not utilise any password protection or other security measures. Booting the Android TV operating system, I obtained a shell as the root user. However, the root user is limited due to the implementation of SELinux access control security policies. I discovered that the device uses the U-Boot bootloader to boot up the Android TV operating system. The automatic boot process can be stopped, and the U-Boot command shell can be accessed. The command shell provides an attacker with various attack vectors, one of which I demonstrated in Subsection 4.4.4, mitigating the use of the SELinux kernel security module. Another possible vulnerability I discovered is using an IR emitting diode to remote control the device, described in Subsection 4.4.5.

4.5 Services analysis

This section describes the security analysis of running services. The device was tested after the initial setup, with no additional user-installed applications. Open ports are discovered and analysed by the Nmap software tool. Then, possible vulnerabilities of discovered services are discussed and analysed.

4.5.1 Ports and Services Analysis

I started the analysis with a Nmap scan. First, I obtained the IPv4 address of the analysed Mecool KM6 Deluxe. Although Nmap correctly detected open ports, the services were not detected appropriately. To find the services, I used the shell that I got access to by the hardware analysis from Section 4.4. The detected open ports together with services are in Table 4.3.

4.5.2 Analysis

Chromecast built-in provides a technology to cast content from other devices in the local network. I was unsuccessful in finding any recent vulnerabilities that target and affect the Chromecast built-in services. However, the Chromecast built-in functionality can be hijacked and abused by an attacker with access to the local network.

The Android TV Remote Service provides a protocol for sending remote controlling injections is probably vulnerable to CVE-2021-0889, although, no public exploit or addi-

[4] https://github.com/Arduino-IRremote/Arduino-IRremote/

Port	Service
6466/tcp	Android TV Remote Service
6467/tcp	Android TV Remote Service
8008/tcp	Chromecast Built-in
8009/tcp	Chromecast Built-in
8443/tcp	Chromecast Built-in
9000/tcp	Chromecast Built-in
38091/tcp	Chromecast Built-in
42245/tcp	Chromecast Built-in

Table 4.3: Open Ports and Services

tional information is available. It is ranked as critical, no user interaction is needed and could lead to remote code execution. It was first publicly disclosed in Android Security Bulletin in November 2021 together with the release of security patch 2021-11-05. However, since the latest security patch to the Mecool KM6 Deluxe dates back to September 5, 2020, the security patch level was never pushed, making it almost certain that the device is vulnerable.

Due to being unsuccessful with previous analysis results of services and their vulnerabilities, I decided to choose a different approach and analyse Bluetooth wireless technology, which is widely adopted and present on probably almost every Android platform-powered device, including the analysed Mecool KM6 Deluxe. This wide adoption, however, creates possible points of interest for the attacker. At first, I used l2ping[5] software tool to flood the device with L2CAP echo request messages and create a denial-of-service (DOS) attack. The remote controller and Bluetooth wireless earbuds were connected to the Mecool KM6 Deluxe to test possible increases in latency or other symptoms of being attacked. I flooded the device from two separate Bluetooth interfaces, but the device worked correctly without any latency increase or another sign. Next, after searching multiple security-related forums, I discovered vulnerability CVE-2023-45866, which targets the Bluetooth stack implementation. The attack could, in theory, trick the victim's system to initiate pairing and establish an encrypted connection with an unauthenticated, spoofed, peripheral role HID device with no interaction from the user, which makes it effectively zero-click vulnerability. The peripheral HID device can then inject messages into the victim's system. I successfully found a proof of concept (POC) and tested the Mecool KM6 Deluxe, which was prone to vulnerability, and I could send keystrokes as an HID device. The attacker can prepare custom HID keystroke messages, install malicious applications, enable the Android Debug Bridge (adb), or many others. However, the analysed device is probably vulnerable to another vulnerability, the CVE-2021-0434, first publicly disclosed in the Android Security Bulletin in November 2021, but no additional information, proof of concept (POC) or exploits are publicly available.

4.5.3 Other Vulnerabilities

Besides the exposed services through open ports, there are many other services running on the Mecool KM6 Deluxe. However, due to making security the top priority in the Android operating system development, vulnerability reward program and others, every detected

[5]https://github.com/pauloborges/bluez/blob/master/tools/l2ping.c

vulnerability is fixed using the security patching system together with releasing the Android Security Bulletin every month. Those vulnerabilities provide only vague descriptions without any proof of concept (POC), or exploits. This approach reduces the exploitability of vulnerabilities and forces the device manufacturers to regularly release updates containing the security patches to their devices. On the other hand, the AOSP project with the open-source approach to the development should provide more information about each vulnerability, thus hardening the security even more. However, this may be a subject of debate.

4.5.4 Summary

First, I scanned the open ports using the Nmap software tool. The detected services were however not properly detected. Thanks to the already performed hardware analysis, I was able to assign open ports to corresponding services. During the analysis, I was not able to successfully confirm vulnerabilities affecting those services. I proceeded with an analysis of popular and widely adopted Bluetooth wireless technology. The discovered vulnerability was also practically tested and confirmed to be present on the device. The limited vulnerability discovery is due to Google's policy of releasing security patches and security bulletins.

4.6 Application testing

This chapter describes the performed security analysis of pre-installed applications that are shipped in the firmware image of the Mecool KM6 Deluxe. The security analysis is based on the OWASP Mobile Application Security methodologies described in Section 3.6. The analysis is performed by the APKHunt software tool, and the results are then discussed. For the targets of analysis, I chose the DroidTvSettings application.

4.6.1 DroidTvSettings

The DroidTvSettings application comes pre-installed in the analysed Mecool KM6 Deluxe. It provides configuration of the connected screen, picture mode, HDMI CEC, audio output, and others. This makes it a good adept for security analysis:

- The application requires the Fingerprint API permission, however, it is not entirely clear why, because the tested Mecool KM6 does not have a physical biometric fingerprint scanner. The use of unnecessary permissions might not be considered good security practice. However, the application is possibly developed for multiple devices and those might contain the fingerprint reader, however, the Mecool does not make any.
- The permissions in use are not deprecated.
- The biometric authentication mechanism is implemented appropriately.
- There was no log write detected, that contains data, that are sensitive or valuable in some way. However, the application writes to log output for every configuration change.

- The keyboard cache is not disabled, although the application does not use the keyboard as input.
- No exposure of user's sensitive and private data was detected.
- There were no hardcoded IP addresses, e-mail addresses, user details, database details, or other sensitive data detected.
- There were also no hardcoded keys, tokens or secrets.
- No hardcoded links were observed.
- The used pseudo-random number generators do not use precise enough seeds.
- The application's push notifications do not leak sensitive or valuable data.
- No network configuration files are present and no network use is implemented, thus it is resilient against network attacks.
- No obfuscation techniques were detected.

4.6.2 Summary

The pre-installed application DroidTVSettings was statically analysed by the APKHunt software tool. The analysis revealed some weak spots in the implementation that could be improved, however, I did not discover any major security issues, thus the application can be considered secure.

4.7 Firmware Analysis

This chapter describes the performed firmware analysis using the reverse-engineering techniques based on the OWASP Firmware Security Testing Methodology (FSTM) from Section 3.7. The initial part of the analysis consists of obtaining the firmware, which can lead to unconventional methods. It is then followed by initial analysis and extraction done by the Binwalk software tool. After the firmware extraction, the firmware content is statically analysed by Firmwalk and EMBA software analysis tools.

4.7.1 Obtaining the Firmware

At first, obtaining the firmware image of the Mecool KM6 Deluxe was necessary. Usually, firmware images can be downloaded from the official manufacturer's website. This ensures that the downloaded image is valid and authentic and was not tampered with, thanks to the integrity check done against the image's hash. However, I could not find the firmware image on the manufacturer's website. The manufacturer's official Download page contained firmware images for various products, but KM6 Deluxe was missing. However, it is important to notice that the firmware images are not hosted on the official manufacturer's website but on MEGA cloud storage, which is not a good security practice. Searching multiple sources, I discovered a download link for the firmware image on a Russian forum 4PDA[6], probably the official firmware image for the analysed device. In theory, firmware might also be obtained by dumping the content of the eMMC flash storage using the UART

[6]https://4pda.to/forum

interface; however, this would take a long time and could even contain corrupted bits due to the sensitive nature of UART. Although I could not verify whether it was an official firmware image, I continued with the analysis.

4.7.2 Initial Analysis

After downloading the firmware image, which was in `img` file format, it is necessary to explore aspects of the firmware image to acquire information about the used file types, possible filesystem metadata or any valuable information that can be used. I manually gathered information about the firmware and its filesystem using the Binwalk software tool. The entropy value was low, implying that the firmware image is probably not encrypted or compressed. I discovered that it contained four partitions of the YAFFS filesystem that had to be extracted.

4.7.3 Filesystem Extraction

As mentioned, it is necessary to extract the filesystem for further analysis. For this, I used the Binwalk software mentioned previously in Subsection 4.7.2. However, Binwalk cannot extract the YAFFS filesystem by itself. For YAFFS filesystem extraction, I used the yaffshiv[7] software tool. After the extraction, I got four partition images: `odm.img`, which contains board and SoC-specific customisations, `product.img`, storing vendor's customisations of the Android framework `system.img`, containing the Android framework and `vendor.img`, used by device vendor to distribute binaries that are not distributable to AOSP [4].

4.7.4 Filesystem Content Analysis

The filesystem content analysis finds clues such as legacy insecure network daemons, hardcoded usernames, passwords, hardcoded API endpoints, extraction of uncompiled binaries, etc. As a first step, I used the Firmwalker software tool. Due to its working architecture, the report can contain a lot of false positive results. It is necessary to manually analyse the report to filter out false positives and find clues and valuable information leading to potential vulnerabilities. As the Firmwalker can not support automatic firmware extraction, I ran it separately for every extracted partition image mentioned in Subsection 4.7.3. During manual analysis, it was detected that there is a folder named `dropbear`. Dropbear[8] is a lightweight SSH client and server built especially for embedded and resource-constrained systems, for example OpenWrt. Because I can not be sure if the downloaded and analysed firmware image is authentic, I compared the content of the folder from the downloaded image and the folder on the actual physical device, which were identical. In the `dropbear` folder, I discover a file containing an authorised key and two additional files containing RSA and DSS host keys. Besides the `dropbear` folder and keys within the `system.img` partition image, it did not contain the Dropbear server binary itself. I proceeded with the analysis of other partition images. After analysing the `vendor.img` partition image, I discovered the Dropbear server binary file. Besides the Dropbear server binary, additional binaries are part of a Dropbear multi-purpose SSH software, including the Dropbear client and Dropbear key generator, which are also in the folder. Particularly interesting is a shell script that, besides generating RSA and DSS host keys, also launches the Dropbear server

[7] `https://github.com/devttys0/yaffshiv`
[8] `https://github.com/mkj/dropbear`

binary using the host keys and the authorised key files. Thanks to the analysis performed in Subsection 4.4.4, I executed the shell script. Using the Nmap software tool, I scanned the opened ports and discovered a newly opened port, 22, usually used by the SSH protocol, together with the correctly detected SSH server. The Dropbear multi-purpose software suite is present in version v0.52, which dates back to November 2008. There are currently 15 tracked vulnerabilities, one of which is CVE-2016-7406, ranked as critical; the severity of others also ranges from medium to high, making the Dropbear multi-purpose software vulnerable. There is only one tracked vulnerability that affects the present version with the publicly available exploit, CVE-2016-3116; however, it does require a user to be already authenticated and thus can not be appropriately tested. Other vulnerabilities do not have publicly available exploits, proof of concepts (POCs), or any information. In my opinion, it is important to discuss whether the presence of Dropbear multi-purpose software is just a remnant of software development that was not removed in the production release. This theory can be supported by the fact that I did not encounter the SSH server to be enabled during the device testing. However, it might represent a real threat in the form of an SSH backdoor planted by the device manufacturer. Further on, I proceeded with the analysis using a more advanced open-source software tool, the EMBA. As already mentioned in the description of the EMBA software tool in 3.7.1, the EMBA analysis is split into multiple stages. Initially, it automatically performs filesystem extraction before the analysis stage itself. The EMBA exports logs for each of the running modules. Those logs provide a module's description of the analysis process, the output of the software tool doing the analysis and a summary of vulnerabilities, clues, hints, or any valuable pieces of information. The EMBA successfully identified the operating system as Linux-based. Another module detected the Linux kernel in version 4.9.18, which dates back to March 2017. More than 1,692 vulnerabilities are tracked for this version; however, due to the kernel modifications and the working nature of Android, many of them were not exploitable. The module focused on verifying those vulnerabilities and also did not report anything. Another module worth mentioning detected 45 configuration files. I manually searched them for possible clues or information for further vulnerability analysis, but I was unsuccessful; all of them contained configurations for Wi-Fi, Bluetooth and GPS adapters, audio effects, and others. The EMBA did not find any password-related files. Next, the EMBA detected already found and mentioned the Dropbear multi-purpose software tool. Another interesting find was the discovery of a Busybox software tool. The Busybox combines multiple Unix binaries in a single executable binary. Although the log files provide an absolute path in the filesystem to the Busybox binary, the binary was not at the specified path, which I confirmed. I tried to boot the device into Android Recovery by the methodology mentioned in Subsection 4.4.4. In the Android Recovery, I discovered the Busybox binary in version 1.22.1, which dates back to January 2014, making it ten years old. There currently exist 19 detected vulnerabilities currently exist that affect this version. The Busybox vulnerability identification and verification module verified 10 vulnerabilities based only on the BusyBox version. Most of them do not provide any additional information on how to reproduce them or affect networking, which is not available in the Android Recovery. I was able to verify and replicate one practically, the CVE-2017-16544. The Busybox's shell, the ash, does not properly sanitize escape sequences in filenames in the terminal in the use of the tab autocomplete feature of the shell. In theory, it can lead to code execution, arbitrary file writes or other attacks. Another module, which checks binaries for weak functions, detected that those functions were used hundreds of times. These were also confirmed by modules based on Ghidra[9] and Radare2[10] software tools. Other modules, checking scripts written in popular languages Python, Lua, PHP or Perl, or any frequently deployed did not discover anything. Only false positives were detected by the module identifying HTTP-related files. Also, no cronjobs, mail-related files, network configurations, weak file permissions or history files were discovered, nor verified.

4.7.5 Summary

The Mecool KM6 Deluxe's firmware image was analysed. First, the firmware image was obtained from a Russian forum because the manufacturer did not provide it on its official website. Next, the firmware image was analysed by Binwalk to gather information about partition images that Binwalk extracted together with yaffshiv and were not encrypted. The extracted images were then analysed by Firmwalker, which discovered the Dropbear multi-tool to be present in one of the images. The version was outdated, with multiple possible vulnerabilities. An additional file containing authorised keys was found. However, this raised concerns if those are remnants of development or a potential backdoor. Advanced analysis performed by EMBA identified the Linux kernel version with possible vulnerabilities. Besides the already discovered Dropbear vulnerability, another one was present in the firmware file, the Busybox. The version was outdated, multiple possible vulnerabilities were tracked, and one was practically verified. There were also hundreds of uses of weak functions.

[9] https://github.com/NationalSecurityAgency/ghidra
[10] https://github.com/radareorg/radare2

Chapter 5

Summary of Vulnerabilities

This chapter summarizes the detected vulnerabilities in the analysed Mecool KM6 Deluxe device. The severity, the score, of vulnerabilities is rated by Common Vulnerability Scoring System (CVSS) metrics. Although hundreds of vulnerabilities were found, most of them were not possible to be exploited and thus verified.

- **Unsecured UART interface:**

  CVSS Score: 7.6

  CVSS 3.1 Vector: AV:P/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

  Mitigation proposal: Implement password protection or encryption or disable the UART completely in production devices.

- **Unsecured U-Boot shell:**

  CVSS Score: 7.6

  CVSS 3.1 Vector: AV:P/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

  Mitigation proposal: Implement password protection or encryption.

- **Android TV shell with root access through UART interface:**

  CVSS Score: 7.6

  CVSS 3.1 Vector: AV:P/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

  Mitigation proposal: Provide Android TV shell as a non-root user or disable the UART completely in production devices.

- **Bluetooth HID keystrokes injection:**

  CVSS Score: 6.3

  CVSS 3.1 Vector: AV:A/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:L

  Mitigation proposal: Apply the available security patch.

- **Man-in-the-middle due to ARP spoofing:**

  CVSS Score: 9.6

  CVSS 3.1 Vector: AV:A/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

  Mitigation proposal: Implement ARP inspection or static ARP entries.

- **Denial-of-service due to packet flooding:**

  CVSS Score: 7.4

  CVSS 3.1 Vector: AV:A/AC:L/PR:N/UI:N/S:C/C:N/I:N/A:H

  Mitigation proposal: Implement rate limiting, traffic filtering or anomaly detection.

- **Outdated Dropbear software tool:**

  CVSS Score: 5.1

  CVSS 3.1 Vector: AV:L/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N

  Mitigation proposal: Replace with the newest available version.

- **Outdated Busybox software tool:**

  CVSS Score: 8.8

  CVSS 3.1 Vector: AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

  Mitigation proposal: Replace with the newest available version.

Chapter 6

Conclusion

In conclusion, this Master's thesis aims to provide a comprehensive security analysis of a selected Android TV box, the Mecool KM6 Deluxe, based on the Android TV operating system. First, an architectural description of the Android TV operating system, based on the Android operating system, is provided. Understanding the analysed system's architecture is an essential part of the preparation for the security analysis process. Then, the security testing process is specified, ranging from vulnerability tracking with appropriate metrics to a detailed description of different security analysis approaches. The analysis encompasses a comprehensive examination of the software components, hardware analysis, and firmware security testing. The initial phase of the security analysis involved network traffic capture, which was subsequently analysed using Wireshark and Bruteshark software tools. This was followed by the analysis of the network stack, which involved simulating a range of attack scenarios, including packet flooding and ARP spoofing, that resulted in denial-of-service (DOS) and man-in-the-middle (MITM) attacks. The subsequent phase of the security analysis focused on hardware analysis. The Mecool KM6 Deluxe was disassembled and the printed circuit board (PCB) was analysed. This revealed a group of exposed soldering pads. Further inspection revealed that these pins are used by the UART debug interface. This was confirmed by connecting to the USB-to-UART bridge with the appropriate baud rate. It was possible to alter the boot process or access the Android TV shell as the root user. I proceeded with the hardware analysis by exploiting the infrared receiving diode present on the analysed device. Having completed the hardware analysis, I then proceeded to examine the running services. I discovered two services with open ports but was unable to provide evidence of their vulnerability. I proceeded with analysing the Bluetooth and successfully demonstrated a weakness. The penultimate stage of the analysis was completed by a security evaluation of one of the pre-installed applications. The analysis was concluded with a firmware analysis using reverse engineering techniques, which revealed several vulnerabilities. In conclusion, a security analysis was performed on the aforementioned Android TV box, which revealed vulnerabilities and proposed mitigations. This serves to demonstrate the importance of device security and the necessity of addressing any potential weaknesses.

Bibliography

[1] *Android Open Source Project: Android runtime and Dalvik.* Available at: https://source.android.com/docs/core/runtime.

[2] *Android Open Source Project: Application security.* Available at: https://source.android.com/docs/security/overview/app-security.

[3] *Android Open Source Project: Implementing SELinux.* Available at: https://source.android.com/docs/security/features/selinux/implement.

[4] *Android Open Source Project: Partitions.* Available at: https://source.android.com/docs/core/architecture/partitions.

[5] *Android Open Source Project: Security-Enhanced Linux in Android.* Available at: https://source.android.com/docs/security/features/selinux.

[6] *Android Open Source Project: System and kernel security.* Available at: https://source.android.com/docs/security/overview/kernel-security.

[7] *Common Vulnerability Scoring System v3.1: Specification Document.* Available at: https://www.first.org/cvss/v3.1/specification-document.

[8] *UART Baud Rate and Output Rate.* Available at: https://support.sbg-systems.com/sc/kb/latest/technology-insights/uart-baud-rate-and-output-rate.

[9] TCP SYN Flooding and IP Spoofing Attacks. In: *1996 CERT Advisories.* Software Engineering Institute, 1996. Available at: https://insights.sei.cmu.edu/documents/503/1996_019_001_496172.pdf.

[10] *SINGLE-CHIP USB-TO-UART BRIDGE.* CP2102. Silicon Labs, january 2017. Rev. 1.8. Available at: https://www.silabs.com/documents/public/data-sheets/CP2102-9.pdf.

[11] Agrawal, R. K. and Mishra, V. R. The design of high speed UART. In: *2013 IEEE Conference on Information & Communication Technologies.* 2013, p. 388–390. DOI: 10.1109/CICT.2013.6558126.

[12] Ahvanooey, M. T., Li, Q., Rabbani, M. and Rajput, A. R. A survey on smartphones security: software vulnerabilities, malware, and attacks. *ArXiv preprint arXiv:2001.09406.* 2020.

[13] Alam, I., Khusro, S. and Naeem, M. A review of smart TV: Past, present, and future. In: *2017 International Conference on Open Source Systems & Technologies (ICOSST).* 2017, p. 35–41. DOI: 10.1109/ICOSST.2017.8279002.

[14] Bogdanoski, M., Suminoski, T. and Risteski, A. Analysis of the SYN flood DoS attack. *International Journal of Computer Network and Information Security (IJCNIS)*. MECS Publisher. 2013, vol. 5, no. 8, p. 1–11.

[15] Brant, C. D. and Yavuz, T. A Study on the Testing of Android Security Patches. In: *2022 IEEE Conference on Communications and Network Security (CNS)*. 2022, p. 217–225. DOI: 10.1109/CNS56114.2022.9947240.

[16] Divakaran, D. M., Murthy, H. A. and Gonsalves, T. A. Detection of Syn Flooding Attacks using Linear Prediction Analysis. In: *2006 14th IEEE International Conference on Networks*. 2006, vol. 1, p. 1–6. DOI: 10.1109/ICON.2006.302563.

[17] Elenkov, N. *Android Security Internals: An In-Depth Guide to Android's Security Architecture*. No Starch Press, 2014. ISBN 9781593275815. Available at: https://books.google.cz/books?id=y11NBQAAQBAJ.

[18] Engebretson, P. *The Basics of Hacking and Penetration Testing: Ethical Hacking and Penetration Testing Made Easy*. Elsevier Science, 2013. The basics. ISBN 9780124116412. Available at: https://books.google.sk/books?id=69dEUBJKMiYC.

[19] Fuentes, F. and Kar, D. C. Ethereal vs. Tcpdump: A Comparative Study on Packet Sniffing Tools for Educational Purpose. *J. Comput. Sci. Coll.* Evansville, IN, USA: Consortium for Computing Sciences in Colleges. apr 2005, vol. 20, no. 4, p. 169–176. ISSN 1937-4771.

[20] Guzman, A. *OWASP firmware security testing methodology*. Available at: https://github.com/scriptingxss/owasp-fstm/blob/master/README.md.

[21] Hou, Q., Diao, W., Wang, Y., Liu, X., Liu, S. et al. Large-scale Security Measurements on the Android Firmware Ecosystem. In: *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. 2022, p. 1257–1268. DOI: 10.1145/3510003.3510072.

[22] Hwang, D. D., Schaumont, P., Tiri, K. and Verbauwhede, I. Securing embedded systems. *IEEE security & privacy*. IEEE Computer Society. 2006, vol. 4, no. 02, p. 40–49.

[23] Jones, K. R., Yen, T.-F., Sundaramurthy, S. C. and Bardas, A. G. Deploying Android Security Updates: an Extensive Study Involving Manufacturers, Carriers, and End Users. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: Association for Computing Machinery, 2020, p. 551–567. CCS '20. DOI: 10.1145/3372297.3423346. ISBN 9781450370899. Available at: https://doi.org/10.1145/3372297.3423346.

[24] Langley, A., Riddoch, A., Wilk, A., Vicente, A., Krasic, C. et al. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. New York, NY, USA: Association for Computing Machinery, 2017, p. 183–196. SIGCOMM '17. DOI: 10.1145/3098822.3098842. ISBN 9781450346535. Available at: https://doi.org/10.1145/3098822.3098842.

[25] Lazić, A., Bjelica, M. Z., Nad, D. and Todorović, B. M. Google Assistant Integration in TV Application for Android OS. In: *2018 26th Telecommunications Forum (TELFOR)*. 2018, p. 420–425. DOI: 10.1109/TELFOR.2018.8612143.

[26] McConnell, S. *Code Complete*. Pearson Education, 2004. Developer Best Practices. ISBN 9780735636972. Available at: https://books.google.cz/books?id=LpVCAwAAQBAJ.

[27] Nam, S. Y., Kim, D. and Kim, J. Enhanced ARP: preventing ARP poisoning-based man-in-the-middle attacks. *IEEE Communications Letters*. 2010, vol. 14, no. 2, p. 187–189. DOI: 10.1109/LCOMM.2010.02.092108.

[28] Nanda, U. and Pattnaik, S. K. Universal Asynchronous Receiver and Transmitter (UART). In: *2016 3rd International Conference on Advanced Computing and Communication Systems (ICACCS)*. 2016, vol. 01, p. 1–5. DOI: 10.1109/ICACCS.2016.7586376.

[29] Sanguino, L. A. B. and Uetz, R. Software Vulnerability Analysis Using CPE and CVE. *CoRR*. 2017, abs/1705.05347. Available at: http://arxiv.org/abs/1705.05347.

[30] Sarkar, A., Goyal, A., Hicks, D., Sarkar, D. and Hazra, S. Android Application Development: A Brief Overview of Android Platforms and Evolution of Security Systems. In: *2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*. 2019, p. 73–79. DOI: 10.1109/I-SMAC47947.2019.9032440.

[31] Sharma, P. and Gupta, A. *Design, Implementation and Optimization of Highly Efficient UART*. SSRN, 2010.

[32] StatCounter. *Mobile Operating System Market Share Worldwide*. Available at: https://gs.statcounter.com/os-market-share/mobile/worldwide.

[33] Tripathi, N. and Mehtre, B. M. Analysis of various ARP poisoning mitigation techniques: A comparison. In: *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*. 2014, p. 125–132. DOI: 10.1109/ICCICCT.2014.6992942.

[34] Vasile, S., Oswald, D. and Chothia, T. Breaking all the things—A systematic survey of firmware extraction techniques for IoT devices. In: Springer. *Smart Card Research and Advanced Applications: 17th International Conference, CARDIS 2018, Montpellier, France, November 12–14, 2018, Revised Selected Papers 17*. 2019, p. 171–185.

[35] Wang, Y. and Song, K. A new approach to realize UART. In: *Proceedings of 2011 International Conference on Electronic & Mechanical Engineering and Information Technology*. 2011, vol. 5, p. 2749–2752. DOI: 10.1109/EMEIT.2011.6023602.

[36] Weidman, G. *Penetration Testing: A Hands-On Introduction to Hacking*. No Starch Press, 2014. ISBN 9781593275648. Available at: https://books.google.sk/books?id=T_LlAwAAQBAJ.

[37] Yan, Y., Cosgrove, S., Blantont, E., Ko, S. Y. and Ziarek, L. Real-Time Sensing on Android. In: *Proceedings of the 12th International Workshop on Java*

Technologies for Real-Time and Embedded Systems. New York, NY, USA: Association for Computing Machinery, 2014, p. 67–75. JTRES '14. DOI: 10.1145/2661020.2661026. ISBN 9781450328135. Available at: https://doi.org/10.1145/2661020.2661026.