Doctoral theses at NTNU, 2022:94

Gunnar Alendal

Digital Forensic Acquisition of mobile phones in the Era of Mandatory Security

Offensive Techniques, Security Vulnerabilities and Exploitation

Doctoral thesis

NTNU
Norwegian University of Science and Technology
Thesis for the Degree of Philosophiae Doctor
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of Science and Technology

Gunnar Alendal

Digital Forensic Acquisition of mobile phones in the Era of Mandatory Security

Offensive Techniques, Security Vulnerabilities and Exploitation

Thesis for the Degree of Philosophiae Doctor

Gjøvik, March 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the Degree of Philosophiae Doctor

Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

ISBN 978-82-326-6191-6 (printed ver.)
ISBN 978-82-326-5328-7 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

Doctoral theses at NTNU, 2022:94

Printed by NTNU Grafisk senter

Abstract

The increased use of consumer electronics like computers, mobile phones, smart watches, external hard drives, etc. has made digital forensics more important for law enforcement. Consumer products now contain more information about a person's life than ever before, useful in any criminal investigation. Gaining access to forensically valuable data is often crucial for a successful law enforcement investigation. At the same time, the mandatory security and complexity of these devices have increased, making successful acquisition of forensically valuable data more difficult.

Successful acquisition now requires law enforcement to understand the underlying technology and possibly bypass security schemes protecting the user data. This thesis contributes with knowledge in this setting, by looking at different security challenges law enforcement meet when trying to acquire data from digital devices, and especially mobile phones. This thesis aims at increasing the knowledge on how law enforcement can use security vulnerabilities in digital forensic acquisition of modern mobile phones, improve the effectiveness of such use and gain knowledge on new attack surfaces.

Preface

This thesis is submitted in partial fulfilment of the requirements for the degree of Philosophiae Doctor (PhD) at the Norwegian University of Science and Technology (NTNU). The presented work was carried out at the Faculty of Information Technology and Electrical Engineering, Department of Information Security and Communication Technology (IIK) at NTNU from 2016 until 2021. This research was carried out under supervision from Associate Prof. Dr. Geir Olav Dyrkolbotn and Prof. Dr. Stefan Axelsson. This research received funding from the Research Council of Norway programme IKTPLUSS, under the R&D project "Ars Forensica", grant agreement 248094/O70.

Acknowledgements

Travelling the land of reverse engineering, security vulnerabilities and exploitation can be frustrating and lonesome, hunting flaws in the dark corners of technology. Remember to bring motivation:

'*In theory, there is no difference between theory and practice, while in practice, there is.*' (Benjamin Brewster)

'*Trying is the first step towards failure.*' (Homer Simpson)

I would first like to thank my supervisors Geir Olav Dyrkolbotn and Stefan Axelsson. This has been a journey I couldn't have made without your support, expertise, guidance, patience and honesty. You have brought me up when I was down, and brought me down when I was up. All of this has been a lesson for life that I will forever be thankful for. Your supervision is 31337.

I would also like to thank NTNU and Kripos/NCIS Norway for giving me this opportunity. Especially I want to thank the people in the Ars Forensica project, Prof. Dr. Katrin Franke, Lasse Øverlier and the many project partners, particularly my fellow PhD candidates: Jens-Petter Skjelvåg Sandvik, Jan William Johnsen, Kyle Porter, Stig Andersen, Nils Martin Mikael Karresand and Jul Fredrik Kaltenborn. I am also thankful to the Department of Information Security and Communication Technology at NTNU for aiding me in completing this research.

Also I would like to thank Ruhr-Universität Bochum (RUB) and University of Oslo (UiO) for great training and inspiration in the information security field.

In addition, numerous people have helped and inspired me throughout these years. Big thanks go out to Dragan Mitrevski, Bjørn Greve, Tormod Tjaberg and all other great colleagues at Kripos.

And without music, there wouldn't be much work done. Thanks Robert Smith and The Cure for a lifelong motivation and inspiration through your music. Without it, I would be lost in a forest, all alone.

But I wouldn't be anywhere without my fantastic friends, family, brothers and parents. Thanks for the regular "are you done yet?", mom. It helps. And one last mjau to our late furry family member. Thanks Julian and Louise for being so underwhelmed by my research and all the time I spend in front of the keyboard, constantly reminding me how little it matters compared to you, and your fantastic mother. There's ordinary, and then there's you. Jeg elsker og forguder deg, Margrethe.

List of Papers

Paper I — G. Alendal, G. O. Dyrkolbotn and S. Axelsson, 'Forensics acquisition - Analysis and circumvention of samsung secure boot enforced common criteria mode,' *Digital Investigation*, 2018.

Paper II — G. Alendal, S. Axelsson and G. O. Dyrkolbotn, 'Exploiting Vendor-Defined Messages in the USB Power Delivery Protocol,' in *Advances in Digital Forensics XV*, 2019.

Paper III — G. Alendal, S. Axelsson and G. O. Dyrkolbotn, 'Leveraging the USB Power Delivery Implementations for Digital Forensic Acquisition,' in *Advances in Digital Forensics XVII*, 2021.

Paper IV — G. Alendal, G. O. Dyrkolbotn and S. Axelsson, 'Digital Forensic Acquisition Kill Chain - Analysis and Demonstration,' in *Advances in Digital Forensics XVII*, 2021.

Paper V — G. Alendal, S. Axelsson and G. O. Dyrkolbotn, 'Chip chop - smashing the mobile phone secure chip for fun and digital forensics,' *Forensic Science International: Digital Investigation*, 2021.

Paper VI — G. Alendal, 'Breaking Android Security by Abusing Implicit HW Trust,' In submission.

Contents

Abstract . . . . . **i**
Preface . . . . . **i**
Acknowledgements . . . . . **i**
List of Papers . . . . . **i**
Contents . . . . . **iii**
Figures . . . . . **vii**
Tables . . . . . **ix**
Code Listings . . . . . **xi**
Acronyms . . . . . **xiii**

I Overview **1**
1 Introduction **3**
1.1 Background and Motivation 3
1.1.1 The “equity issue” of security vulnerabilities 5
1.1.2 The end user perspective 6
1.2 The Existence of Useful Security Vulnerabilities 7
1.3 The Nature of a Security Vulnerability 8
1.4 Approaching the Challenge 9
1.5 Research Questions 11
1.6 Thesis Layout 11
2 Background **13**
2.1 Digital Forensics and Digital Forensic Acquisition 13
2.2 Data Sources 14
2.2.1 Current State of DFA on Android 16
2.2.2 Current DFA Challenges and Security Vulnerabilities 18
2.3 Mobile Security preventing DFA 19
2.3.1 Encryption 21
2.3.2 DFA Layered Attack Approach 25
2.4 Android DFA Attack Path 26
2.5 Security Vulnerability Identification and Exploitation 28
3 Ethical Considerations of *0-day* discovery in Digital Forensics **29**

3.1 Introduction 29
3.2 Good vs. Evil 30
3.3 EQ1 30
3.4 EQ2 30
3.5 EQ3 31
3.6 Fighting Evil with Evil 31
3.7 Summary and Future Work 34
4 Summary of Work 37
4.1 Main Research Question 38
4.2 RQ1: How can modern security measures be bypassed by exploiting security vulnerabilities? 39
4.3 RQ2: Can we identify potential future attack surfaces useful for digital forensic acquisition? 41
4.4 RQ3: How can digital forensic acquisition benefit from published security vulnerabilities? 42
4.5 Additional Work 42
4.5.1 Additional Research Presentations and Awards . . . 42
4.5.2 Unpublished Work 43
4.5.3 Preliminary Work 43
5 Discussion and Conclusion 45
5.1 DFA Challenges 46
5.2 Vulnerability Discovery and Exploitation in DFA 46
5.3 New Attack Surfaces for DFA 47
5.4 Improving DFA Method Development 47
5.5 Ethical Discussions 48
5.6 Conclusions and Future Work 48
Bibliography 51

II Included Publications 63
I Forensics Acquisition — Analysis and Circumvention of Samsung Secure Boot enforced Common Criteria Mode 65
I.1 Introduction 66
I.2 Related Work and Contributions 67
I.3 CC Mode and Methodology 69
I.4 Samsung Secure Boot Model 71
I.5 Samsung CC mode and SBOOT 73
I.5.1 The PARAM Partition 74
I.5.2 SBOOT enforcing CC mode 76
I.5.3 MDM mode 80
I.6 Unauthorised disabling of CC mode 81
I.6.1 Modifying the PARAM partition 82

I.6.2 SBOOT exploitation 82
I.6.3 Setting DN_ERROR 82
I.7 Conclusion 83
I.8 References 84
II Exploiting Vendor-Defined Messages in the USB Power Delivery Protocol 87
II.1 Introduction 88
II.2 Related Work 89
II.3 USB Power Delivery Protocol 90
II.4 Methodology 93
II.5 Experimental Results 95
II.6 Conclusions 102
II.7 References 103
III Leveraging the USB Power Delivery Implementation for Digital Forensic Acquisition 105
III.1 Introduction 106
III.2 USB Power Delivery Protocol 108
III.3 Research Methodology 110
III.4 Results 114
III.4.1 Information Gathering 114
III.4.2 Passive Monitoring 115
III.4.3 Firmware Files 116
III.4.4 Firmware Reverse Engineering 118
III.4.5 Apple Vendor-Defined Protocol 119
III.4.6 Firmware Modification and Rollback 121
III.5 Conclusions 122
III.6 References 123
IV Digital Forensic Acquisition Kill Chain — Analysis and Demonstration 127
IV.1 Introduction 128
IV.2 Related Work 129
IV.3 Digital Forensic Acquisition Kill Chain 130
IV.3.1 Background 130
IV.3.2 Kill Chain Overview 131
IV.3.3 Kill Chain Phases 134
IV.4 Case-Motivated Kill Chain Example 138
IV.5 Conclusions 140
IV.6 References 141
V Chip Chop — Smashing the Mobile Phone Secure Chip for Fun and Digital Forensics 145
V.1 Introduction 146
V.2 Background 149

V.2.1 Embedded Secure Element 149
V.2.2 CC EAL 149
V.2.3 eSE Threat Model 150
V.3 Related Work 151
V.4 The Attack 152
V.4.1 Attack Assumptions 153
V.4.2 Information gathering 154
V.4.3 0-day Information Leak Oracles 156
V.4.4 0-day Vulnerability Discovery and Exploitation . . . 158
V.4.5 Attack Capabilities and AES Key Exposure 161
V.5 Attack Implications 162
V.5.1 Android User Screen Lock Brute Force 163
V.6 Discussion 164
V.7 Conclusions and Future Work 166
V.8 References 167
V.9 Appendix 173
VI Breaking Android Security by Abusing Implicit HW Trust . . . 177
VI.1 Introduction 178
VI.2 Background 180
VI.2.1 File Based Encryption and Credential Encrypted Storage 180
VI.2.2 The eSE HW Target 182
VI.2.3 The Original eSE Exploit 182
VI.2.4 The Original Brute Force Attack 182
VI.3 The Implicit Trust Attack 183
VI.3.1 Implicit HW Trust 183
VI.3.2 Abusing Vulnerable HW Trust 184
VI.3.3 Attack Implementation 185
VI.3.4 Android User Screen Lock Brute Force 186
VI.4 Attack Countermeasures and Anti Countermeasures 186
VI.4.1 Avoiding Detection 187
VI.4.2 Staying Persistent 188
VI.5 Discussion 189
VI.6 Conclusions and Future Work 190
VI.7 References 191
VI.8 Appendix 194

Figures

2.1 Digital Forensics 15
2.2 Availability of FBE storage for different user unlock states. . 22
2.3 Simplified `unwrapPasswordBasedSyntheticPassword()`, Credential Encrypted (CE) storage unlock utilising eSE HW [6] 23
2.4 Simplified Android execution domains. 27

4.1 Thesis contributions 37

I.1 Overview of the Samsung Secure Boot model from BootROM to an Android kernel. 72
I.2 PARAM Partition 75
I.3 Simplified pseudo code of `S_boot_enter_download_mode` . 79
I.4 *emergency* ODIN/download mode 80

II.1 USB Type-C pinout [5]. 90
II.2 Data message packet. 90
II.3 Simplified explicit contract negotiation. 92
II.4 Vendor-defined message packet. 93
II.5 Unstructured vendor-defined message header. 93
II.6 Structured vendor-defined message header. 93
II.7 Discover Identity reply packet. 94
II.8 Samsung Anyway S103. 100

III.1 USB Power Delivery data message packet. 109
III.2 Generic, source-initiated explicit contract negotiation. . . . 111
III.3 Experimental setup. 113
III.4 Unstructured VDM header. 120

IV.1 Generic digital forensic acquisition needs. 131
IV.2 Digital forensic acquisition kill chain phases. 132

V.1 eSE logical interface using APDU 151

V.2 eSE stack leak using the `APDU_writeWeaver` / `APDU_readWeaver` oracle 157
V.3 Buffer overflow in eSE APDU_writeWeaver handler 159
V.4 Full eSE flash layout 160

VI.1 Simplified `unwrapPasswordBasedSyntheticPassword()`, Credential Encrypted (CE) storage unlock utilising eSE HW [16] 181
VI.2 eSE logical interface using APDU [16] 183

Tables

I.1 SBOOT environment variables, stored in *adv-env.img* 78
I.2 REBOOT_MODE variable values 80

II.1 Control and data messages in Revision 3.0 (Version 1.2). . . 92
II.2 Structured commands in Revision 3.0 (version 1.2). 94
II.3 Test devices with USB Type-C connectors and protocol support. 96
II.4 Huawei Mate 10 Pro (BLA-L29) message capture. 97
II.5 Samsung Galaxy S9 (G960F) message capture. 98
II.6 Samsung Galaxy S9 (G960F) message capture. 99
II.7 Samsung Anyway S103 and Samsung Galaxy S9 message capture. 101

III.1 Power Delivery protocol messages. 110
III.2 Structured Vendor_Defined message commands. 111
III.3 Explicit Contract between source (non-Apple) power supply (Rev. 3.0) and sink iPhone X (iOS 13.2.2). 115
III.4 Explicit contract between source Apple power supply (Rev. 2.0) and sink iPhone 10,6 (iOS 13.2.2) 116
III.5 Firmware files with their PIDs and test iPhone models . . . 117
III.6 `USB-C_HPM,4.bin` in various iOS versions 118
III.7 `USB-C_HPM,4.bin` details for various iOS versions (see Table III.6) 118
III.8 Data exchanged between source Apple power supply (Rev. 2.0) and sink iPhone 10,6 (iOS 13.2.2). 120
III.9 Discover Identity VDMs between source Apple power supply (Rev. 2.0) and sink iPhone 10,6 (iOS 13.2.2). 121

V.1 Analysed eSE FW images 175
V.2 eSE attack vectors: Exposed valid eSE APDU CLA and INS . 176

VI.1 Tested eSE FW images 195

Code Listings

2.1 `unwrapPasswordBasedSyntheticPassword(), using` *`weaver`* 21
2.2 `unwrapPasswordBasedSyntheticPassword(), using` *`gatekeeper`* 24
V.1 Simple APDU brute force pseudo code 155
V.2 ROP gadget for arbitrary flash and RAM read 160
V.3 `unwrapPasswordBasedSyntheticPassword()` code 163
V.4 Simplified screen lock brute force pseudo code 164
VI.1 `unwrapPasswordBasedSyntheticPassword()` code 180
VI.2 Simplified `APDU_readWeaver` pseudo code 185
VI.3 Assembly code before and after modification 186

Acronyms

ADB Android Debug Bridge. 16, 17

AFU After-First-Unlock. 21, 24, 25, 27, 28, 39, 46

ART Android Runtime. 17

BFU Before-First-Unlock. 21, 26, 28, 39, 40, 46

CE Credential Encrypted. 21, 23, 24, 26, 27, 39, 40

COTS Commercial off-the-shelf. 19

CVE Common Vulnerabilities and Exposures. 18

DE Device Encrypted. 21, 23, 27

DF Digital Forensics. 13–15, 17, 19, 28

DFA Digital Forensic Acquisition. 11, 13–21, 25, 26, 38–43, 45–49

DFAKC Digital Forensic Acquisition Kill Chain. 42, 48

eSE Embedded Secure Element. 20, 22, 23, 26, 27, 40, 46, 49

FBE File-Based Encryption. 21, 23, 24, 28

FW Firmware. 40–42, 46, 47, 49

GDPR General Data Protection Regulation. 7

HID Human Interface Device. 23

HW Hardware. vii, 9, 22–24, 39–41, 43, 46, 47, 49

I^2C Inter-Integrated Circuit. 39

IC Integrated circuit. 20, 49

JTAG Joint Test Action Group. 16

KDF Key Derivation Function. 23

LE Law Enforcement. 3–7, 10, 11, 13, 14, 19, 41–43, 45–48

LOC Lines of code. 19

LPE Local Privilege Escalation. 25

OS Operating System. 7, 8, 15, 40

PCB Printed Circuit Board. 16, 17

RAM Random Access Memory. 15

REE Rich Execution Environment. 26, 27, 46

SCA Side-channel Attack. 26, 40, 49

SMS Short Message Service. 47

SoC System-on-Chip. 41

TCB Trusted computing base. 20

TEE Trusted Execution Environment. 20, 22, 24, 26

TZ TrustZone. 20, 22

USB Universal Serial Bus. 47

USB PD USB Power Delivery. 41, 47

VDM Vendor Defined Message. 47

Part I

Overview

Chapter 1

Introduction

This chapter will give a generic introduction to the challenge this thesis is addressing and how this relates to digital forensics. The motivation, problem formulation and research questions are then introduced. The last section contains the thesis scope and layout.

1.1 Background and Motivation

The increasing use of embedded devices, like smart phones, with the increasing amount of private data contained within, is becoming more and more valuable in criminal investigations. Their increasing complexity and mandatory security is challenging for Law Enforcement (LE), and gaining access to this valuable data is getting increasingly harder for law enforcement. Denied access to this data might be crucial to investigations, ultimately resulting in serious crime becoming unsolved and criminals walking free from prosecution. However, this increasing complexity can also be an advantage for LE, raising the bar for securing such systems. There is an inherent risk of introducing exploitable vulnerabilities as a result of factors such as increased code size, more complex security designs and shorter time to market.

The challenge with increasingly secure devices and more resource demanding digital forensic can be seen in a well known terrorist case in the US [7]. This resulted in a demand from the FBI to Apple to create an *official* (cryptographically signed) and *vulnerable* version of the suspect's iPhone operating system, and then update the device with this vulnerable version to weaken the security so FBI could successfully use this introduced security vulnerability [8] to bypass the unknown user credentials and acquire data from the iPhone. This is in effect introducing what one in computer security would call a *backdoor* [9, 10]: a deliberate feature to bypass a given security feature. This FBI request raised a great deal of discussion [11],

whether Apple should comply and create a vulnerable and "backdoored" version of their own product which, if leaked outside Apple/FBI, could enable other attackers to break the security of other iPhone devices. This would be a disaster for the security of the iPhone as well for the security reputation of Apple products. Apple refused to comply, even when the US court supported the FBI's view [12]. After some dispute, the result was that the FBI looked elsewhere for a solution to acquire data from the suspect's iPhone. The FBI dropped the request to Apple, as the FBI got hold of the user credentials (passcode) protecting the iPhone by other means. Though the FBI has not officially stated how they got hold of the user credentials of the deceased suspect, rumours and speculations suspect the use of a security vulnerability found in the device to recover the user credentials [13, 14].

So it's challenging for LE to gain access to user data on commercial end products used by normal users, because of mandatory security. Commercial and publicly available tools and methods exist, but are challenged by modern mobile phones security measures, like encryption [15–18]. A similar challenge is known from the "cryptowar" history [19] and policy options for decryption by government agencies, getting hold of encryption keys *ex ante* (backdoors) or *ex post* (like decryption orders). Both are controversial. A third option is *ex nunc*, accessing encryption keys in real time, through "legal hacking" and "government hacking powers" [19]. This *ex nunc* might be an option for digital forensics as well.

This FBI case is thus an good illustrative example of these current challenges LE faces in digital forensics, both technically and policy wise. The owner in this specific case was a suspected terrorist, but the phone used was a stock commercial product, with no extra security features added beyond the mandatory security enforced by Apple on their iPhone products. The stance made by Apple sets an important stage for digital forensics in years to come. This discourages LE to request vendors to implement backdoors to bypass security. At the same time, this encourages LE to look for ways to bypass security mechanisms without help from vendors, even when the vendor is within the same jurisdiction. Exploiting security vulnerabilities is one way to bypass security mechanisms, turning LE into an attacker of the same system. The increasing complexity with added features might increase the rate of implementation errors [20], where security vulnerabilities is a subgroup of such errors, potentially exploitable to bypass security measures. Such security vulnerabilities are referred to as *0-day* vulnerabilities when unknown to the vendor, where no patch exists [21]. The corresponding *n-day* vulnerabilities are when vulnerabilities are known to the vendors and patches for affected systems are available, but not necessarily installed. LE can utilise both *0-day* and *n-day* vulnerabilities, with *0-day*

being more useful due to both current and *future* seized devices being vulnerable. But an *n-day* can also be equally useful for unpatched devices, but the frequency of such affected devices might be lower. As LE can seize devices, and take them offline, *n-day* vulnerabilities can be exploited by waiting for patches to be available, rediscover potentially fixed security vulnerabilities, and develop exploits for the unpatched seized devices. This "patch preventing" capability of LE can potentially become important.

The challenge in this FBI case, mandatory security of the Apple iPhone, shows that even the expertise of the FBI might not be able to technically bypass such security measures, even in a terrorist case where one would expect every available resource to be available in the FBI. Officially admitting this and asking for vendor assistance emphasises this challenge. A vendor backdoor would technically solve the case, but the vendor refused, leaving the FBI to turn elsewhere for a solution, like security vulnerabilities, to solve the case [22]. As the current lack of official support from vendors to implement LE access through e.g. backdoors or other forms of vendor modifications, new forensic methods for securing data might need a push towards the use of more offensive techniques. Using offensive techniques and exploitation of security vulnerabilities, in order to gain access to valuable information, as seen from a digital forensics perspective, raises both technical and ethical questions as described in the following.

1.1.1 The "equity issue" of security vulnerabilities

Turning back to the FBI case, we can already see some potential ethical challenges that follow in the wake of discovering security vulnerabilities. The FBI could discover an exploitable security vulnerability by their own research into either *0-day* or *n-day* vulnerabilities, or they could pay for such research to be done. Either way, if the vulnerability is unknown to the vendor, a *0-day* vulnerability, this knowledge also raises important dilemmas.

Discovering *0-day* vulnerabilities to bypass security mechanisms comes with a great responsibility. A security vulnerability represents knowledge of a weakness other attackers, such as foreign state actors, also can discover and abuse. This raises the question if such discovered security vulnerabilities should be used defensively, being reported to the affected vendors, so they could mitigate the risk. Keeping a security vulnerability from disclosure might result in numerous future successful, offensive, use in digital forensics, at the expense of a much bigger set of unknown, uninformed and vulnerable users. These conflicting interests between offensive and defensive use of security vulnerabilities are not new [23–25]. In the

US, this challenge is publicly discussed [26] and addressed by the United States Government [27]. Whether to restrict discovered security vulnerabilities for offensive use or disclose for defensive use, is decided case by case, evaluating new security vulnerabilities in a Vulnerability Equities Process (VEP). Representatives from United States Government agencies gather regularly to evaluate and decide the fate of newly discovered security vulnerabilities discovered by government agencies [27]. This policy is not without debate [28, 29] and only represents one of potentially many conflicting national policies. This "equity issue" is thus a political issue, where states and jurisdictions develop their own standards and policies on how to handle discovered security vulnerabilities. As in the US, several government agencies are stakeholders of such security vulnerabilities, with potentially conflicting interests. Knowledge of security vulnerabilities is sensitive, as such government agencies often represent intelligence interests, further restricting the *use* of such vulnerabilities. Every exploitation of a security vulnerability, either by intelligence agencies or by LE in digital forensics, is running a risk of exposing the vulnerability, rendering it less useful if a patch is created, turning it into an *n-day* vulnerability. So policies need to account for a plethora of conflicting and challenging needs from the stakeholders. If the FBI used a discovered *0-day* security vulnerability on the suspect's iPhone in the above mentioned case, this was probably regulated by such a policy. To further complicate the situation, such a *0-day* security vulnerability might be independently discovered by multiple states and jurisdictions, with bilateral and multilateral cooperation, but conflicting policies.

In Europe, a recent EU draft raised a similar concern on the LE challenges with mandatory security and encryption of consumer devices, but does not currently provide a solution. There is only a suggestion that cooperation is needed to create a balance between consumer privacy and LE needs: "Since there is no single way of achieving the set goals, governments, industry, research and academia need to work together to strategically create this balance" [30, p. 4]. The EU draft does not consider any technical alternatives for LE, like backdoors or the use of security vulnerabilities.

1.1.2 The end user perspective

The increase in mandatory security of mobile phones has a huge benefit for the data and communication security of end users. The argument from LE is that criminals use this technology to prevent prosecution and thus this has a negative impact on society. However, users in countries where citizens, journalists, dissidents, etc. might be victims of surveillance and

risk of false prosecution, this has a huge positive impact on their security. Mandatory security and encryption thus protects the democratic values of privacy and freedom [31].

User data is also worth much for other organisations than LE. The need to protect the population against other threats, like personal data being used without consent by companies like Facebook and Google, has resulted in laws and regulations. The General Data Protection Regulation (GDPR) is an attempt from the European Union to regulate organisation's use of end user data without consent [32].

So the benefits of mandatory device security for individual citizens must be weighted against the potential negative effect of the potential obstruction of justice. An open question might thus be if laws and regulations should regulate the use of security vulnerabilities in digital forensics, in an attempt to prevent abuse [33, 34]. In Norway such regulations are in place for the use of "data reading" technology, where modifications, software or hardware, are installed on devices to extract data, including sound, video streams, keyboard logs, and so on from devices in use by suspects [35].

Ethical and other non-technical challenges surrounding research that might be used for both good and bad are very interesting and important and are discussed further in Chapter 3, but we will now shift the focus back to the main focus of this thesis: Technical challenges with the discovery and use of security vulnerabilities in digital forensics.

1.2 The Existence of Useful Security Vulnerabilities

To develop new digital forensic methods based on security vulnerabilities, one is of course in need of exploitable security vulnerabilities. In addition, these security vulnerabilities need to be exploitable in a digital forensics context. The most prevalent threat of any interconnected device is over the internet as a carrier, often referred to as "cyber security" [36]. However, as LE lawfully can seize devices, the physical or "near device" attack surfaces are equally valuable. The security of mobile devices regarding both these attack surfaces has been evaluated and extensively researched [37–42].

Mazuera-Rozo et al. [43] studied the existence of security vulnerabilities in the Android Operating System (OS) and the number of days they existed before getting patched. They studied 1,489 security vulnerabilities that have been reported in the years 2015-2017. According to [43] the trend is an increase in security vulnerabilities in mobile phones, in parallel to the increase in mandatory security. Three important results are presented in [43].

The first is that the existence of security vulnerabilities in the Android OS is not decreasing, but increasing. The increased complexity might introduce more security vulnerabilities. This can of course have other explanations, like improved methods to discover security vulnerabilities, according to [43]. Either way, this is good news if we want to use such security vulnerabilities in digital forensics.

A second result is the duration security vulnerabilities exist in the Android OS. Their results estimated an average of 770 days from when a security vulnerability is introduced in the source code, until it is patched. This is *not* the time from the discovery of a security vulnerability until a patch is available. They in effect measured how long a security vulnerability is a *0-day* vulnerability. This is a great motivation to look for unknown vulnerabilities, as a digital forensic acquisition method based on a *0-day* security vulnerability has a potential to be used for a long time, solving many cases.

A third result is *where* the Android OS security vulnerabilities exist. Mazuera-Rozo et al. showed that 82.46% of their analysed vulnerabilities were in the Android kernel drivers and native libraries. This is in the very heart of the Android OS, and hopefully such security vulnerabilities can have an impact when used in digital forensic acquisition as core security features are expected to be enforced by the kernel and native libraries.

1.3 The Nature of a Security Vulnerability

To get into a better position to discuss the potential use of security vulnerabilities in digital forensics, we need to start with the basic nature for any exploitation of security vulnerabilities. How such security schemes can be manipulated and bypassed, and move backwards towards an understanding of what is actually needed from a security vulnerability and where to locate them.

In general terms, one can say that anything that can be manipulated as part of a security scheme is susceptible to a security vulnerability. The most simple example is normal *user input*; anything the user can type and the security scheme has to evaluate, like username or password, is a potential security vulnerability. Improper validation of user input can lead to security vulnerabilities like *buffer overflows* [44, 45]. We can broaden this by saying that input from e.g. a fingerprint sensor should also be considered *user input* to the security scheme [46]. Although a normal user cannot manipulate this communication channel, a resourceful attacker with physical or local access to the security scheme might.

In the rising complexity of embedded devices, the number of security

schemes with potential *user input* open for manipulation are expanding. As an example, one can unlock a Samsung Galaxy Android phone today by a vast number of ways; the normal user screen lock (pin, password or pattern), fingerprint, trusted voice, trusted location, trusted device and even remotely through "find my phone" services. These ever-expanding ways to authenticate and unlock a device increase the attack surface and possibilities to access user data [47]. However, many of these additional authentication methods are only available after the user unlocks the device with the most trusted method, the user screen lock authentication.

In addition to the numerous direct input of user credentials, the security scheme has to consider the trust in its own running environment; if an attacker can modify the execution of code in the security scheme, breaking the integrity of the code, the security scheme might fail [48, 49].

A different example is logical implementation errors in security schemes. One descriptive example being Hardware (HW) encryption on embedded devices, enrolling mandatory encryption and authentication. Our earlier research discovered security vulnerabilities in both the encryption scheme, using weak cryptographic keys, and in the authentication scheme of an external hard drive series featuring HW AES encryption [50]. This research and the security vulnerabilities found, has resulted in new methods to do digital acquisition of such devices based on security vulnerabilities that do depend on manipulation of *user input*.

A security scheme might also have implicit trust on non-internal parts of a security scheme. For example it might trust the storage (like flash or hard drive), the baseband processor, RAM, a.o. [51].

So a security vulnerability can take the form of a.o. an erroneous implementation bug, a code integrity flaw, a design flaw and even improper trust relations between parts of a security scheme.

1.4 Approaching the Challenge

Researching and locating security vulnerabilities to develop new methods for digital forensic acquisition might not be a straightforward task. Even if the goal of locating security vulnerabilities is clear, there is not a well defined way to reach that goal, simply because one cannot know which security vulnerabilities are required to bypass a particular security scheme, nor where to locate them. This is the very nature of security vulnerabilities: they are not supposed to be present and it's hard to predict their location.

The general approach in this thesis is therefore to attempt to start by identifying security schemes that prevent successful acquisition and then identify potential attack surfaces therein. Further we attempt to identify

examples of potential security vulnerabilities and try to exploit such vulnerabilities to develop new digital forensic acquisition methods. Backtracking from challenging security measures to solutions can be referred to as a *bottom-up* approach.

Thus our approach is simple: identify and isolate security schemes that are challenging for digital forensic acquisition of modern mobile phones, identify potential attack surfaces for these security schemes, attempt to discover security vulnerabilities that can be exploited to bypass the security scheme, reaching the end goal of acquiring data for digital forensics.

A *top-down* approach would start with evaluating the design and documentation of a security scheme and correlate this against common criteria [52] and best practises [53]. This requires access to documentation and possibly source code of the security scheme. The situation for most security researchers is however quite the opposite, being forced to utilise the *bottom-up* approach. Very often one has to start with the actual implementation, the end product, and gain most of the information to form the vulnerability research from this. The process of gaining knowledge from such closed source products is often referred to as *reverse engineering* [54]. The big benefit from this approach is that one evaluates the actual implementation of a security scheme and not the *intended* design and implementation. Very often security vulnerabilities are introduced between the design process and the actual end product [43].

Going from an evaluation phase to the discovery of a, potentially new, attack surface, continuing to the discovery of potentially several unknown (*0-day*) and known (*n-day*) security vulnerabilities, and further to successfully exploit such vulnerabilities, might be a tremendous task. Even having a successful exploitation of a specific security scheme isn't enough, as multiple security schemes might be bypassed to fully develop a new digital forensic acquisition method.

Thus, going from an unknown device with mandatory security, to a fully developed digital forensic acquisition method based on the use of exploitation of security vulnerabilities might not be straightforward and given. How can such new digital forensic acquisition methods be researched and developed in this increasingly complex and secure device design? Is it a feasible task and do powerful actors like LE have advantages that can be beneficial when exploiting security vulnerabilities? Can LE, being able to seize devices, control and deny installation of any released security updates released for the device? This could open for new digital forensic acquisition methods based on *published*, *n-day*, security vulnerabilities. Can such *patch preventing* advantages be utilised?

1.5 Research Questions

In what way can security vulnerability discovery and exploitation contribute to the improvement of digital forensic acquisition?

RQ1: How can modern security measures be bypassed by exploiting security vulnerabilities?

- **RQ1.1:** How can mandatory encryption of user data be bypassed by exploiting security vulnerabilities, without knowledge of the user credentials?
- **RQ1.2:** What security schemes other than encryption must be bypassed to access encrypted user data?

RQ2: Can we identify potential future attack surfaces useful for digital forensic acquisition?

- **RQ2.1:** How can USB Power Delivery be an attack surface for DFA?
- **RQ2.2:** Can attacks on USB Power Delivery be generalised to other architectures?

RQ3: How can digital forensic acquisition draw benefit from published security vulnerabilities?

- **RQ3.1:** How can a methodical approach help LE discover and exploit security vulnerabilities?

1.6 Thesis Layout

The rest of this thesis is organised as follows: Chapter 2 presents necessary background to understand some of the technical challenges addressed in the contributions of this thesis. Chapter 3 is our contribution to the discussion on the ethical dilemma of LE discovering *0-day* security vulnerabilities. Chapter 4 presents a short summary of the relationship between published papers and the research questions. Chapter 5 gives a summary of the contributions in this thesis. Part II contains all published papers that contribute to this thesis.

Chapter 2

Background

In this chapter related background is presented to set the contribution of this thesis in context. The focus will be on embedded devices, typically mobile phones, and limiting the scope mainly to Android, as this is the dominant operating system on mobile devices [55, 56]. First we'll present a general introduction to the Digital Forensics (DF) process and the relation to the Digital Forensic Acquisition (DFA) process. Further we'll give an overall view of data sources of interest, with the state of DFA research. Further, we will introduce technical details on current security schemes preventing DFA on modern Android mobile phones. We will follow the outline from the previous chapter, looking first at where digital forensics look for valuable data on mobile phones, to challenges with security schemes preventing acquisition of this data, and from there dive into concrete and specific technical security schemes that need bypassing for successful Digital Forensic Acquisition (DFA). This background will set the stage for our research into using security vulnerabilities to aid in this bypass of security.

The first sections will discuss current state-of-the-art and challenges of DFA with respect to the increased mandatory security of embedded devices. The following section will discuss the most prevalent security features current DFA is facing, to get an overview of where focus and DFA research should be targeted. The last section will summarise with an example Android DFA attack path.

2.1 Digital Forensics and Digital Forensic Acquisition

Criminal investigations require LE to gather forensically valuable data from many different sources. As more and more of a person's life is digitised, digital sources have become more and more important. Embedded devices, like mobile phones, have become a portable personal computer, containing our most sensitive personal data. This makes mobile phones a primary

source in almost any criminal investigation.

Digital Forensics (DF) is the process of seizing devices, acquiring data, analysing data and reporting [57]. See Figure 2.1 for a simplified view of this process. The first phase focuses on the seizure of devices, including locating and selecting devices to prioritise from a crime scene. The second phase, acquiring data to be analysed, typically involves the mirroring of device data from storage, like hard drives and flash storage. The third phase analyses the data acquired, and can be a challenging task, given the increasing amount and complexity of data stored. The analysis phase should produce relevant data for the specific investigation and thus must adapt to both the amount and context of data. The last phase presents results and findings from the analysis phase in a format useful for investigators to collect and compare with data from other forensic sources.

Much focus has been on the analysis phase of DF, as increasing amounts of data are being acquired and processed by LE [58–62]. Simson L. Garfinkel [63] evaluated the challenges ten years ahead for DF, published in 2010. The paper feared that DF tools and methods would fall behind. The major challenges the author points out are the increasing amounts of data acquired, difficulty of access to low level imaging of storage (like embedded flash storage), increasing number of data formats to analyse (like file formats), increasing number of devices in cases, cloud storage (which includes legal challenges when seizing data across borders), access to volatile storage (RAM) and lastly encryption of data. The challenge with encryption of data means that even a full image of any device storage is forensically worthless without the means to decrypt the data, challenging the acquisition phase of DF [64–67].

The process of acquiring digital data and information from seized devices falls under Digital Forensic Acquisition (DFA). DFA gains access to the plaintext data, enabling analysis.

No data from DFA, no analysis.

The DFA phase is thus a crucial part, a bottleneck, of DF and this phase is increasingly being challenged by increasing, *mandatory*, device security and complexity [33]. If this challenge is not solved, LE might lose an important source of data.

2.2 Data Sources

Digital forensics often seeks data sources that contain user generated data. On embedded devices like mobile phones this is typically personal data,

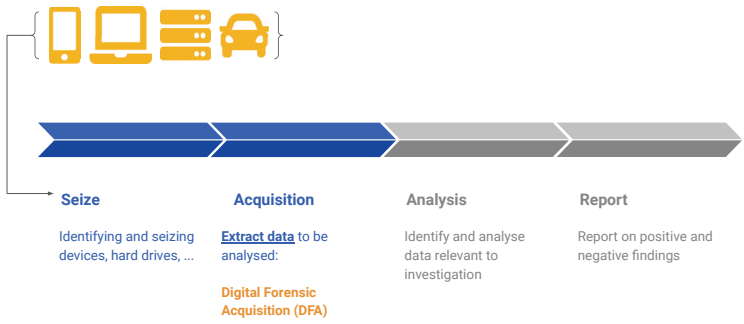


Figure 2.1: A simplified Digital Forensics process

such as messages, pictures, GPS locations, call logs, etc. Data is valuable in most criminal investigations and any source containing such data is important.

Generally speaking, there are two main data sources for such personal data on most digital devices, for example computers and mobile phones: volatile memory (RAM) and long-term storage, like flash and traditional hard drives. These two sources differ in many ways, both the data stored and the way data is to be acquired in DFA. Long-term storage consists mainly of well structured data, for instance a file system, meant to store data for later reuse. File systems and file formats tend to use well documented and static storage formats which can be parsed and interpreted as part of the analysis phase of DF. Long-term storage contains most of a device's code and data and is thus the main target for DFA. RAM consists mainly of volatile and short-lived data, not meant to be stored across a power cycle, and is thus repopulated on every restart. The data is mostly unstructured and dynamic, important for the execution of code running on the device as well as data processed at a given time. These structures are often undocumented and the dynamic nature and use of RAM, e.g. by the Operating System (OS), will result in completely different sequence of data from acquisitions. Parsing and interpreting RAM is therefore a completely different challenge of the analysis phase of DF.

Gaining access to and acquiring plaintext data from any of these two data sources is the main goal of DFA.

The challenge for both these data sources is two-fold. One challenge is to access and read the stored data and provide a copy that can be analysed further. An additional second challenge is any decoding of the data into a meaningful plaintext state, including any decryption.

So simply reading data from flash storage or RAM on modern mobile devices is not enough, as major parts of data might be encrypted by mandatory security policies, resulting in the challenging task of decrypting data, where access to encryption key material might be required [68]. The goal of DFA is therefore to acquire *plaintext* data. This might include bypassing any confidentiality technology, like encryption, so the data is ready to be parsed and examined in the analysis phase. Parsing and analysing data content, however, is part of the following analysis phase of digital forensics.

2.2.1 Current State of DFA on Android

Acquiring access to plaintext data sources has different approaches, depending on the type of device control, physical or logical, and the state of the device, powered off or on. The RAM and long-term storage sources are common for most mobile phone manufacturers, but due to technical and implementation differences, acquisition access methods will be different from devices with e.g. Apple iOS and Google Android operating systems. Trying to limit the scope of our research we focused on devices with the most common mobile operating system, Android. Though most research mentioned in this section is focused on Android devices, the general challenge should be transferable to other vendors and mobile operating systems.

In a review of Android mobile device forensics, Tayeb et al. [15] discusses the most influential papers in the field. They summarise and discuss selected papers on Android forensics, including the crucial DFA phase, covering both sources of RAM and long-term storage. A common denominator of most of the DFA methods discussed is that they apply to older, out-of-date, Android versions and devices.

There are a vast amount of different options for reading data from long-term storage. Nathan Scrivens et al. [69] gives many examples of such access on Android devices. The main options are chip-off/de-soldering storage chips for off-device reading, JTAG (Joint Test Action Group) interface for in-circuit reading of storage, rooting and exploitation solutions, debug interfaces (Android Debug Bridge (ADB)) and backup solutions, to name a few. These methods vary in requirements, like physical access for chip-off and JTAG and logical access for ADB and backup. Chip-off requires access to safely remove the chip from the device's Printed Circuit Board

(PCB) and JTAG requires access to test pads, often undocumented and hard to find. JTAG test pads are also normally disabled on consumer released devices, as they are mostly used during testing and in production. ADB is a powerful Android debug interface, but normally requires physical access and is normally disabled if the user has not explicitly enabled it. Backup of data can also be hard to come by and is purely up to the user to utilise.

One drawback of most of the above methods is the lack of encryption bypass, giving only access to potentially encrypted data. They represent the traditional reading of data and any added encryption transformation on data before storage will result in the acquisition of encrypted data, not useful for digital forensics.

A different data access example is demonstrated by Seung Jei Yang et al. [70]. They access data through the misuse of device firmware update protocols. This will gain access to *read* long-term storage. Again, this will not be a successful DFA if the data is encrypted, which we will see is mandatory on most modern mobile phones (Section 2.3.1).

Seung Jei Yang et al. [71] also demonstrated a different misuse of the firmware update protocols, to dump RAM. This could give access to encryption keys stored in RAM. Thus combined with other *read* access to long-term storage, this is in general a powerful approach. Additional advantages of acquiring RAM is the ability to analyse in-RAM user data, e.g. from active applications, with active data in RAM, at the time of acquisition. However, access to valuable RAM data, like encryption keys, requires the devices to be seized in a powered-on state, with the encryption keys unlocked in RAM, which is often not the case until the user authenticates for the first time after power on. Section 2.3.1 discusses the encryption challenge in more detail.

Ali-Gombe et al. [72] presented *DroidScraper*, a tool to interpret RAM acquisition of Android devices, focusing on the analysis of Android Runtime (ART) processes. A very efficient and powerful analysis, given a device RAM acquisition. However, the authors express the challenge of acquiring RAM on modern devices and utilise different methods to acquire RAM [71, 73, 74], not readily available on consumer devices seized by law enforcement. There is no generic method to access RAM on modern devices [75]. RAM acquisition is part of a DFA and to be of practical use in DF the RAM acquisition method should be available on seized devices and not constrained to artificial test scenarios.

So we can observe that the current state of DFA is challenged by encryption and that much of the previous research is invalidated when encryption becomes mandatory on modern mobile devices.

2.2.2 Current DFA Challenges and Security Vulnerabilities

Many of the challenges solved by current DFA research (Section 2.2.1) are often out of date and not relevant for the current state of embedded device security. This emphasises that keeping up to speed with technology evolvement is crucial if DFA is to keep up and be relevant. The future of DFA relies on quick adaptation to new technology trends.

The evolution of mandatory security requires DFA researchers to act, sharing challenges with security vulnerability research: to bypass a security scheme, possibly using security vulnerabilities in the process. Security vulnerabilities can potentially be used to both read and decrypt data.

A ***security vulnerability*** is, according to the Common Vulnerabilities and Exposures (CVE) system, "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." [8].

In this paper we often use the two terms *0-day* and *n-day* when referring to security vulnerabilities. Our use of the terms in this research is:
0-day: A potentially exploitable *security vulnerability*, unpublished and unknown to the vendor of the affected product.
n-day: A potentially exploitable *security vulnerability*, published or otherwise known to the vendor and where a patch is readily available for the affected product.

Looking at security vulnerability research in general, there is a difference in interest between traditional security research, trying to discover security vulnerabilities for patching and increasing the security of the product, and DFA, trying to discover security vulnerabilities to bypass security to access user data. This might lead to a development where law enforcement and vendors of forensic tools develop methods to acquire data that uses both known *n-day*, and unknown and undisclosed *0-day* security vulnerabilities. Ideally, all tools and techniques used in digital forensics used by law enforcement to acquire evidence should be open source and widely accessible [76].

Raghavan et al. [77] suggested that current research challenges for digital forensics were divided into five major challenges: complexity problem, diversity problem, consistency and correlation, quantity or volume problem and unified time-lining problem [77]. However, none of these challenges address the increasing challenge of digital acquisition itself and

increasing mandatory security. The main focus is on what happens with digital data *after* acquisition, the analysis phase. More recent research by Montasari et al. [78] points to encryption as one of the most difficult challenges in DF, preventing successful DFA. They point to the paramount importance that researchers are able to design workarounds and exploits to bypass encryption.

This shift in focus from LE on the increasing challenge with mandatory security and encryption is starting to happen in European countries, with increasing focus on the DFA phase of embedded devices. The FORMOBILE [79] project is an EU funded project to develop a complete forensic investigation chain, targeted at mobile devices. EXFILES [80] is another new EU funded project, with a more narrow scope of solving challenges with encrypted mobile phones. Both of these projects try to address similar goals as this thesis: to improve the success of DFA of embedded devices.

2.3 Mobile Security preventing DFA

The steep increase in the adaptation of mobile phones, with increasing amounts of sensitive user data, has escalated the need for security. The increased demand for new features and wider applicable areas of use has increased both the amount of sensitive data to protect and the complexity of such products. Smartphones today are far more advanced than just a decade ago. This increase in both complexity and sensitivity of data has upped the security game. But as security concepts and protection mechanisms quickly become too complex for the average user to understand and manage, consumer product vendors have made many security mechanisms mandatory and transparent to the user. The result is that Commercial off-the-shelf (COTS) products today are far more advanced regarding security mechanisms, even without the user being aware. Many security features of modern mobile phones today are turned on by default and simply cannot be disabled by the user. One example is mandatory user data encryption on all Android phones from version 10 and higher [68]. Thus modern mobile phones have more mandatory security features enabled to protect the increasingly sensitive consumer data.

The steep increase in complexity, both hardware and software wise, has also greatly increased the probability of security vulnerabilities. The increase in complexity and Lines of code (LOC) needed to implement features raises the probability for introduced faults, with security vulnerabilities as a sub category. The ratio of security vulnerabilities per LOC is challenging to estimate. Hatton [81] estimates a defect (bug) density of

< 10 per thousand lines of code (KLOC). Research in OpenBSD suggests a vulnerability rate of much less, with densities three orders of magnitude less [20]. All bugs are not security vulnerabilities, and to transfer these results from OpenBSD to e.g. Android, with all sub-components, might not be fruitful. However, we can say that this is an indication that an increased number of lines of code increases the probability of security vulnerabilities. Thus modern mobile phones have an increased attack surface, with a potentially higher probability of security vulnerabilities, as the complexity and the number of features continue to grow.

This complexity challenge has already been identified in other computer security domains, like the Trusted computing base (TCB) [82] and trusted computing. Trusted computing is the concept where a system is expected to behave as intended, withstanding outside influence, and enforced by stand-alone hardware and software.

Variants of this were needed for embedded devices, and the idea of isolation of sensitive data and computations gave birth to the Trusted Execution Environment (TEE) [83]. There are many different adaptations and implementations of the TEE concept, like the TCB [82], Intel SGX [84] and ARM TrustZone (TZ) [85]. Another concept introduced to embedded devices is the fully separated Secure Element IC [86]. The SIM card is a well-known external and removable secure element, but in recent years *embedded* versions of this concept have been incorporated in e.g. Android devices, Embedded Secure Element (eSE).

The idea for the different trusted computing designs is isolation, simplicity and limitation of code base size (LOC) of critical security components. The assumption is that a smaller and isolated code base, paired with a much higher focus on secure coding standards, should make the security vulnerability rate smaller. The concept is not widely accepted as secure and has caused discussion of its benefits, and risks [87, 88].

These risks and challenges of trusted computing are something that can be used as an advantage by an attacker. The increasing complexity of the individual security features, like trusted computing, together with the increasing total complexity of devices, like mobile phones, also increases the complexity of getting the security right. Many features, many developers, many technology groups, many companies and vendors need to work individually, and in cooperation. Keeping the overall system security intact across all these boundaries might be challenging, providing the potential for introduced security vulnerabilities that can be exploited to create new Digital Forensic Acquisition (DFA) methods.

DFA need not break all security schemes implemented on a modern embedded device, only the ones preventing access to valuable user data.

The biggest challenge preventing successful DFA is currently encryption.

2.3.1 Encryption

On a modern mobile phone, the user data contained might provide valuable input to any investigation. Thus accessing this data might be a crucial step. The mandatory encryption of user data is the basis for the confidentiality in modern secure systems and bypassing this encryption is therefore a necessary step. Without access to the user's screen lock credentials, accessing encrypted data could be achieved by accessing the underlying encryption keys, attacking the encryption algorithms or attempting to regenerate the correct user screen lock credentials using brute force. To be able to evaluate ways to bypass encryption, we first need to introduce the different concepts. As most of the contributions in this thesis are on bypassing security on Android devices, we will introduce the currently preferred encryption scheme on Android 10 and above, File-Based Encryption (FBE).

Android File Based Encryption (FBE)

Android's File-Based Encryption (FBE) [68] consists of two basic encrypted storage available to applications storing user data: device encrypted and credential encrypted storage. The Device Encrypted (DE) storage is available after device boot, but *before* the user unlocks the device, Before-First-Unlock (BFU). DE storage contains files needed to start the device, like the Android OS. The Credential Encrypted (CE) storage is only available after the first user unlock after boot, After-First-Unlock (AFU). Most of the user sensitive data, emails, photos, videos, SMS, application data, etc. are stored in the CE storage and thus contains much potentially valuable data for use in digital forensics. Figure 2.2 shows the relationship between the availability of the FBE CE and DE storage in different user unlock states.

```
result.gkResponse = weaverVerify(weaverSlot, passwordTokenToWeaverKey(
    ↪ pwdToken));
if (result.gkResponse.getResponseCode() != VerifyCredentialResponse.
    ↪ RESPONSE_OK) {
    return result;
}
...
applicationId = transformUnderWeaverSecret(pwdToken, result.gkResponse.
    ↪ getPayload());
```

Code listing 2.1: `unwrapPasswordBasedSyntheticPassword(), using weaver`

Android's FBE currently supports the use of two different trusted computing concepts for protecting the CE storage encryption key mater-

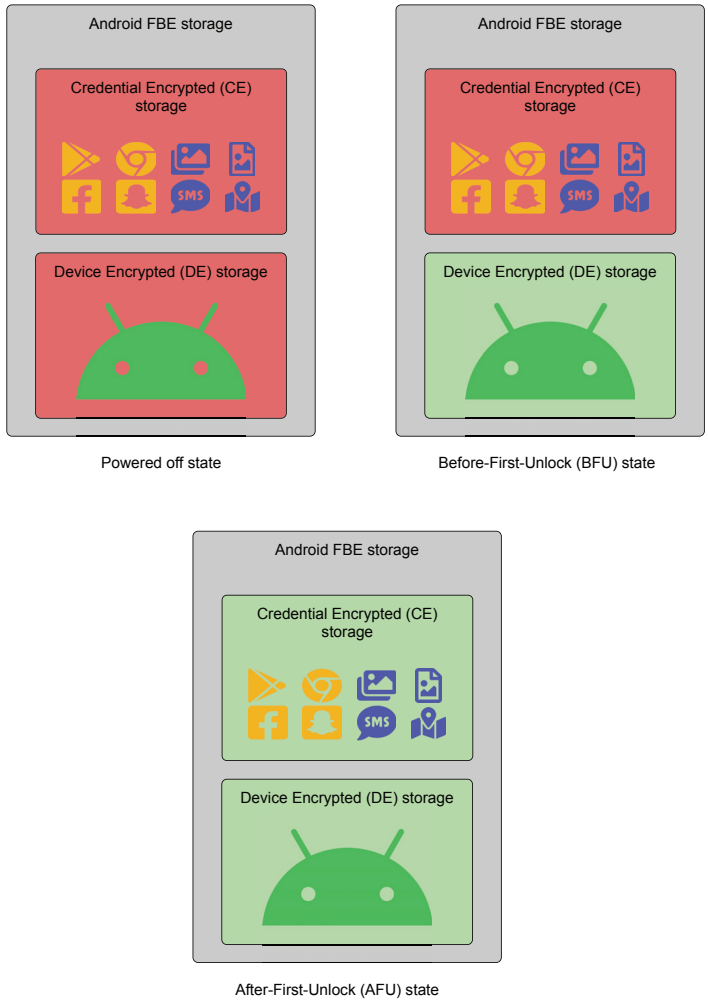


Figure 2.2: Availability of FBE storage for different user unlock states.

ial: TEE, like the ARM TZ, and eSE. These are referred to as *gatekeeper* and *weaver* in the Android source code [89, 90]. Android implements the code interfacing with the Android OS, but vendors, like Samsung, are free to implement the underlying support for *gatekeeper* and *weaver* features independently, based on chosen HW and the different mobile model's spe-

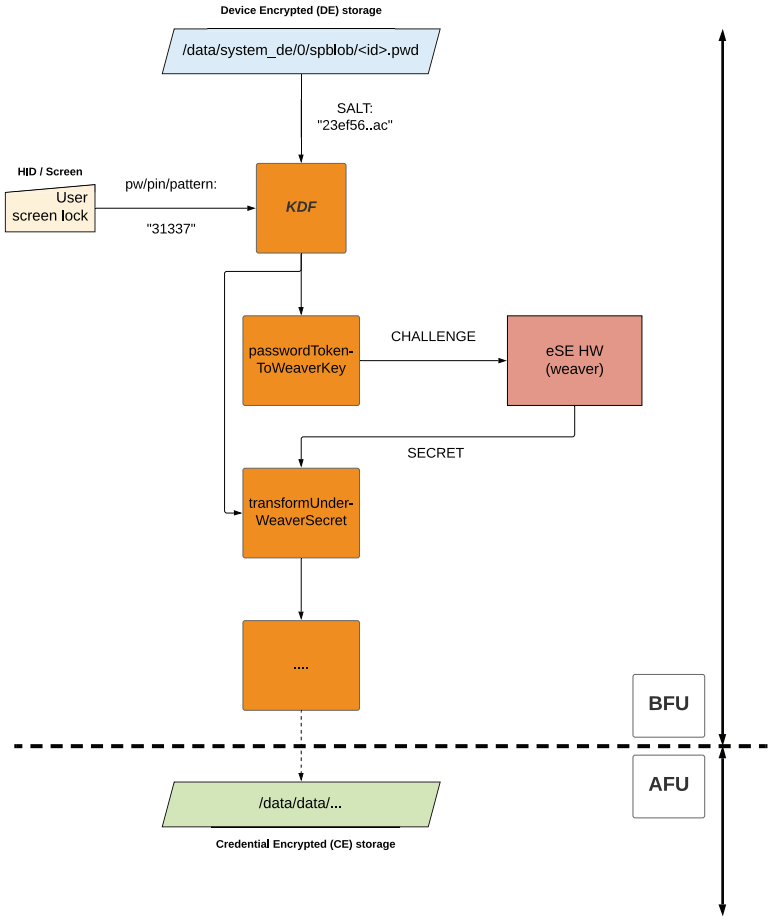


Figure 2.3: Simplified `unwrapPasswordBasedSyntheticPassword()`, Credential Encrypted (CE) storage unlock utilising eSE HW [6]

cifications.

A code fragment of the user screen lock verification and the FBE CE key generation [89] can be seen in Listing 2.1 and Figure 2.3 [6], using Android's *weaver* (eSE) feature. A user enters the screen lock credentials, like a pin, pattern or password, through the screen or a connected Human Interface Device (HID). This user screen lock credentials, together with a *salt* stored in the DE storage, is input to a Key Derivation Function (KDF). The output of the KDF is transformed using a function, `password-`

`TokenToWeaverKey()`, and its output is sent to the *weaver*, the eSE HW, for validation. This output, the `CHALLENGE`, is validated by the eSE HW. If the `CHALLENGE` is verified, the corresponding secret data, `SECRET`, is returned from the eSE. This `SECRET` and the KDF output are input to the function `transformUnderWeaverSecret()`. After this step no more unknown data is needed and the CE storage can be unlocked, taking the device to the AFU state.

Similarly, the screen lock verification and the FBE CE key generation [89, 90] can be seen in Listing 2.2, using the *gatekeeper* (TEE) feature. They are similar, except for the exclusion of a *salt*.

An important component of both trusted computing concepts used is their built-in brute force protection. Thus it's the responsibility of both the *gatekeeper* and *weaver* to keep a count of wrong authentication attempts and enforce time-outs to prevent brute force attempts of user screen lock credentials. This is crucial for the trusted computing design, to protect the encryption key material against even a fully compromised system.

So to access and decrypt Android's FBE CE storage, one way might be to recover the user screen lock credentials.

```
byte[] gkPwdToken = passwordTokenToGkInput(pwdToken);
GateKeeperResponse response;
try {
    response = gatekeeper.verifyChallenge(fakeUid(userId), 0L,
            pwd.passwordHandle, gkPwdToken);
} catch (RemoteException e) {
    ...
}
int responseCode = response.getResponseCode();
if (responseCode == GateKeeperResponse.RESPONSE_OK) {
    result.gkResponse = VerifyCredentialResponse.OK;
    ...
} else if (responseCode == GateKeeperResponse.RESPONSE_RETRY) {
    result.gkResponse = new VerifyCredentialResponse(response.
        ↪ getTimeout());
    return result;
} else  {
    result.gkResponse = VerifyCredentialResponse.ERROR;
    return result;
}
sid = sidFromPasswordHandle(pwd.passwordHandle);
applicationId = transformUnderSecdiscardable(pwdToken,
        loadSecdiscardable(handle, userId));
```

Code listing 2.2: `unwrapPasswordBasedSyntheticPassword()`, using *`gatekeeper`*

2.3.2 DFA Layered Attack Approach

The description of user data encryption from Section 2.3.1 opens for several attack paths to bypass the encryption of user data: recover the user screen lock or recover encryption key material from devices in the AFU state. Bypassing encryption by identifying a flaw in the encryption algorithm itself is an additional, and very powerful, attack, as it can be applied to all use this algorithm across different devices. However it would require identifying a vulnerability yet to be discovered by the cryptanalysis community [91]. Breaking the encryption algorithm itself is the focus of a large scientific community, with its own arms race. This option is not the focus of this thesis.

Recovering the user screen lock requires that the device either stores the user credentials somewhere it can be recovered, or that there is a way to recover the screen lock by abusing some oracle vulnerable to brute force. The rest of the system might be designed to prevent such attacks and this introduces new challenges and security mechanisms to bypass. Accessing such stored credentials or oracles might also require system level access to the device, like a "root" adb shell [92], connected to test devices either with a cable or over a network connection. Gaining such privileged/elevated execution is not trivial and preventing such privileges is part of the security design of the device. This is then a new layer of security to bypass for DFA. Yet another obstacle to obtaining privileged execution might be to gain initial access to execute any attacker code on the device, requiring another security vulnerability. Thus various Android security measures, like secure/verified boot [48, 93–95] might need to be bypassed, designed to thwart attacks on (privileged) code execution.

So trying to solve the encryption challenge by recovering the screen lock on Android might need up to three different security vulnerabilities: initial access to execute attacker code, gaining privilege execution (commonly known as Local Privilege Escalation (LPE)) and finally an encryption bypass vulnerability to recover the user screen lock or encryption key material.

Recovering the encryption key material from volatile memory would require access to read RAM as an unauthenticated user. This would again be considered a new security vulnerability and an additional security measure to bypass for DFA. And if an attacker can retrieve the encryption key material for encryption, one still needs access to read encrypted flash storage, e.g. from a chip-off or through logical channels like initial access and LPE as above.

There have been some fragmented attempts to address parts of these challenges in earlier work (Section 2.2.1) and the ever increasing security,

diversity and complexity of mobile devices from a vast number of different vendors should expect this challenge to grow.

DFA bypassing encryption is thus a more complex and layered challenge, with more than one security mechanism in need of bypass. This might include the use of more than one security vulnerability on more than one attack surface.

2.4 Android DFA Attack Path

To summarise where DFA might attack modern mobile security, we'll use the Android FBE encryption (Section 2.3.1) challenge as a concrete example. Bypassing the main challenge, accessing CE storage containing user data, might not be as simple as gaining hold of the encryption material. Other security features might need to be bypassed as well. We need to keep this whole picture in mind when researching ways to bypass the encryption challenge.

Powered off and BFU States

A phone in powered off or BFU state has no CE storage encryption key material unlocked (Figure 2.2). So even unprivileged RAM access on a target device will not result in the recovery of CE storage encryption key material. This state instead requires an attack to either recover the user screen lock, or an attack on the security mechanism protecting the CE storage encryption key material, like the eSE (Figure 2.3). Attacking the eSE is a stand-alone challenge and can be approached by both logical attacks, using the intended communication channel, or by other attacks, like HW and Side-channel Attack (SCA) [96–98]. The goal is to either use the eSE as an oracle for performing brute force attacks on the user screen lock, or to extract needed sensitive data used to regenerate the CE storage encryption key material. Logical attacks, using the intended communication channel, might need additional attacks to be able to execute privileged code on the device to perform the communication. The eSE logical communication channel is available in the "normal" Android world, often referred to as the Rich Execution Environment (REE) [99], through the Android OS. Figure 2.4 shows a simplified view of the REE, with the unprivileged execution of apps and privileged execution of a.o. the Android OS, and the trusted execution modules, like the eSE and the TEE. Privileged execution in the REE can be achieved through attack surfaces like the secure boot, attacks on physical channels (USB, WiFi, NFC, etc.) or logical channels (network access, SMS, etc.) [100–102].

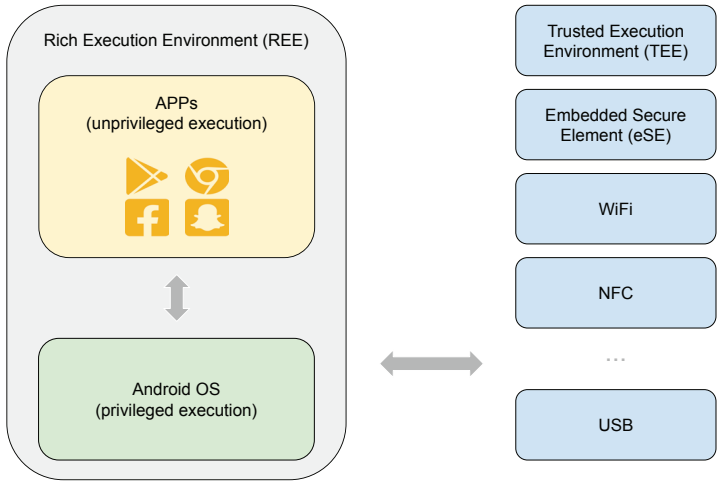


Figure 2.4: Simplified Android execution domains.

So an example attack path for devices in these states can be to locate vulnerabilities to gain privileged execution in the REE, and use this access to attack the separate eSE. Such an example attack on the REE and eSE could potentially open up for a brute force attack on the CE storage unlock scheme from Figure 2.3.

AFU State

The AFU state is the only state where the CE storage encryption key material is already unlocked. Thus the goal can be any attack that gives privileged execution, or execution in the same user context as the CE credentials. The attack requires the device to stay in the AFU state, so an attack that requires a restart of the device would be unsuccessful. Similar attacks on physical and logical interfaces from Section 2.4 could give such access. This access could then be used to access CE storage directly and acquire file contents. A different approach could be to acquire RAM from the device, potentially revealing the CE and DE storage key material. However, this approach would

require some off-device, or on-device, method to utilise this key material. An off-device approach could be to do chip-off of flash storage and attempt to decrypt the FBE storage directly, without involving the rest of the device.

So seizing devices in an AFU state is a viable option, but most vendors have implemented a time-out of user screen lock credentials entered, requiring a re-authentication with the user screen lock credentials within a given time, like 48 hours, effectively removing in-memory encryption key material and unencrypted data, putting the device in a BFU state. This time-out also invalidates most of the additional authentication methods, like fingerprint and face recognition (Section 1.3).

2.5 Security Vulnerability Identification and Exploitation

The attack paths described in the previous sections require the identification of exploitable security vulnerabilities. As the nature of a security vulnerability is that it should not exist, a generic process of discovering them can be challenging to define. However, some basic steps we tried to follow throughout this research are listed below. These basic steps were later used as inspiration for developing a methodical approach to identify and use security vulnerabilities in DF (Paper IV).

- **Reconnaissance and information gathering:** Any source that can be used to gather information on targeted technology and to identify potential attack surfaces. Like documentation, literature, source code, executable binaries, etc.
- **Study attack surfaces:** Before attempting to locate security vulnerabilities, an attempt to isolate and study potential attack surfaces.
- **Vulnerability research:** This task is challenging, with a high presence of creativity, experimenting and experience.
- **Exploitation development:** The creation of a working method and tool to exploit a security vulnerability. Many bugs and security vulnerabilities might be hard to exploit, but this step might include repeatable tasks and techniques.

Chapter 3

Ethical Considerations of *0-day* discovery in Digital Forensics

The discovery and usage of security vulnerabilities "for good" by law enforcement might be a debated and controversial subject. We will supplement with our attitude towards the ethical discussion this raises[1]. This will be a subjective argumentation, trying to set the topic in an ethical context.

3.1 Introduction

In Chapter 1 we described a case where the FBI asked Apple to deliberately introduce a security vulnerability in a specific iPhone, seized by the FBI. As Apple refused to comply, the FBI is believed to have acquired data from the device through the use of a security vulnerability already present on the device. This is very interesting from an ethical point of view; did the FBI acquire knowledge of a security vulnerability in iPhone devices, unknown to Apple? If so, are they obliged to tell Apple, so this vulnerability can be fixed, and thereby protect all of Apple's iPhone users? Doing so would prevent the FBI from using this vulnerability to acquire data from a different device, in a future criminal investigation.

Generalising this. Such security vulnerabilities [8], suspected to be used by the FBI, have existed since the very beginning of software development and will probably exist in the future. The question is if such vulnerabilities can be used for the good of mankind, and if so, how law enforcement should handle knowledge of such security vulnerabilities. We claim that using such vulnerabilities by law enforcement is *acceptable*, and we also claim that such vulnerabilities, *discovered* by law enforcement, should be *kept from disclosure* to e.g. vendors, to prevent the vulnerabilities from

[1]This chapter is based on an essay written by the thesis author as part of the PhD course "MNSES9100 – Science, Ethics and Society" at the University of Oslo.

being fixed (*patched* in computer security terminology). We will also argue *for* the sharing of such knowledge with a sub-group of comparable law enforcement agencies in other countries in *bilateral* and *multilateral* relationships.

3.2 Good vs. Evil

Any intended (backdoor) or unintended bug/security vulnerability in any security feature can potentially be used by attackers to gain unauthorised access to user data. This abuse of security vulnerabilities, often referred to as "hacking" or "exploiting" [103] in media and literature, is often considered bad and unethical, and is often referred to in negative terms, like *cyber attacks*, *ransom-ware attacks* and *cyber warfare*. But if the "attacker" is law enforcement, does that change this perception of using security vulnerabilities, now for "good"? Can there be acceptable situations for using security vulnerabilities to bypass security as means to reach justice? A security vulnerability unknown to the public and/or the vendor is known as a *0-day* and a published vulnerability is known as an *n-day*, referring to the number of days since publicly known [21].

We shall try to evaluate some ethical questions surrounding these dilemmas.

3.3 EQ1

Can security vulnerabilities be used "for good" in digital forensics? If so, should law enforcement engage in the research and discovery of such publicly unknown vulnerabilities (*0-days*) or should law enforcement only use published vulnerabilities (*n-days*) in digital forensics?

3.4 EQ2

If we accept there are such situations, where law enforcement can use security vulnerabilities to bypass security schemes to enable successful acquisition of user data, does this come with more ethical challenges? If the successful acquisition of a suspect's phone in a criminal investigation uses a *0-day* vulnerability, unknown to the vendor, are law enforcement obliged to inform the vendor of this security vulnerability, so it can be fixed, thereby protecting all users of the same phone model, but consequently denying law enforcement to reuse the security vulnerability in later criminal investigations? If this is indeed required of law enforcement, should

they report the security vulnerability *before* using it in *any* criminal case?

3.5 EQ3

Should law enforcement in one country, which are against e.g. death sentence, share methods with law enforcement in countries that have death sentences? Should law enforcement in one country, like Norway, discovering and developing digital forensic methods based on security vulnerabilities, share these with countries that are less democratic or where law enforcement can be suspected to be e.g. corrupt?

3.6 Fighting Evil with Evil

As stated in the introduction, we will argue for the claim that law enforcement should be able to use security vulnerabilities, both *0-days* and *n-days*, in digital forensics. We will also argue that law enforcement, engaging in the discovery of such vulnerabilities, should keep them from disclosure to vendors for patching.

Whether law enforcement should be able to use security vulnerabilities in digital forensics (Section 3.3) is fundamentally a question of ethics. Are security vulnerabilities "bad" in nature? Are they morally wrong to use, as the intention of a security vulnerability is not to enable security bypass? But can we discuss an *intention* of a security vulnerability at all, when it's not made on purpose, but simply is a result of an action intended for something else? We can argue that most security vulnerabilities are a result of something going wrong with an action with good intentions. The developer intended to improve the quality of some feature in some vendor's code and probably had no intention of creating a way to bypass a security feature. So any use of code that exposes a security vulnerability will be *abuse* of the intended code.

To help us evaluate the ethics, we'll turn to two different normative ethics[2]; *deontology* [104] and *consequentialism* [105]. *Deontology* is based on rules regarding an action. An action can be seen as *good* or *bad*, based on e.g. universal, religious, cultural or personal rules. Any action can then be classified free from context and the situation it can be performed in. An extreme example can be that taking someone's life is always considered *bad*, even if this could save others (like killing a terrorist to prevent a terrorist attack). In contrast, *consequentialism* evaluates an action on the consequences and outcome. So taking someone's life can be accepted in certain

[2]Normative ethics is the study of a ethical action; how one ought to act.

cases, as the consequence might be that other lives can be spared. Although this is a very simplified description of these two normative ethics, it should be sufficient to illustrate our ethical dilemmas.

Based on a *deontology* view of the world, we could then argue that it's always ethically wrong to use security vulnerabilities, no matter what the intended outcome is. It's simply wrong to abuse someone else's wrongdoing for a "destructive" goal of bypassing a security feature. In contrast, from a *consequentialist* point of view, the *outcome* or *consequence* of using security vulnerabilities in digital forensics is the important aspect. The goal is to acquire data for analysis in a criminal investigation, helping to serve justice, so using security vulnerabilities should then be considered ethically right. As law enforcement must do all in its power to enlighten any criminal investigation, should it ignore a possible data source that can help with this, simply because one needs to (ab)use a security vulnerability in the process? The intention of helping in the serving of justice is clearly a "good" intention, so this is ethically right, according to *consequentialism*. One important aspect of using security vulnerabilities in digital forensic, is that law enforcement does not *introduce* these in the vendor's code. Any given security vulnerabilities are present regardless of if someone finds it or not. So law enforcement is not changing anything in the target device *before* using a security vulnerability in digital forensics.

Since law enforcement is only using already present security vulnerabilities in digital forensics, one can compare it with a key under a doormat. Acquiring knowledge of the security vulnerability of keeping your key under the doormat, law enforcement should be able to use this key to get access through the locked door. What if a given victim or suspect had written down their user credentials on a post-it note? The security vulnerability of having a "backup" of your user credentials this way, clearly doesn't prevent law enforcement from using this to unlock any security mechanism with these user credentials. Even more closer to our situation; what if there's something wrong in the design of the security mechanism, the "door lock", that enables anyone with "lock picking" expertise to open the lock without the key? Should this prevent law enforcement from using this as a way to open the door? We argue that using security vulnerabilities in digital forensics is nothing more than using digital "lock picking" skills to bypass a security feature.

Continuing with the door lock analogy (Section 3.4). If law enforcement gains knowledge of a way to lockpick a secure door lock, is law enforcement required to tell the vendor of the lock vulnerability? The fate of any discovered (security) vulnerability would always be in the hands of the discoverer, leaving the vendors and their users with no influence. According to *deontology*, we must then tell the vendor, as it would be wrong to not

tell the vendor of the negative effect this has on the security of their lock, leaving all customers vulnerable to theft, as well as a potential harm to the vendor's reputation. The negative effect that law enforcement potentially loses a method to be used in criminal investigations is not of any concern. We could further argue that the knowledge of such vulnerabilities belong to the vendor, as it's their erroneous product. We could even argue that in a situation of law enforcement having knowledge of a security vulnerability unknown to the vendor, law enforcement knows of potential harm that can be done to both the vendor, and its customers, and by not informing the vendor could be seen as a passive acceptance of harm to be done to the vendor and its customers. Is law enforcement, or any other party that gains knowledge of a security vulnerability, in their right to evaluate such risk of harm, and take a decision based on a biased and subjective evaluation? It would be hard, if not impossible, for law enforcement to perform a sufficient risk analysis of the potential harm that a discovered security vulnerability could do to a vendor and its customers. And even if such risk analysis could be performed, is it up to the discoverer of a security vulnerability to decide the best outcome of informing the vendor or not?

Turning to a *consequentialism* view, we can include the positive sides of *not* informing a vendor of a security vulnerability. While the *deontology* view focuses on the act itself, *consequentialism* focuses on the effect of the act. Looking beyond the negative effects already discussed, the positive effects are also strong. Having the knowledge of a security vulnerability can be of great importance in criminal investigations, with the effect that justice might be served. One security vulnerability could potentially be used in numerous criminal investigations, with a potential to not only solve criminal cases post-mortem, but even prevent crime, e.g. terrorist attacks, from happening, and thereby saving lives. Of course the potential to save lives is a strong positive argument for not informing vendors of security vulnerabilities, but does it outweigh the negative risk of potential harm happening to the vendor and users? I would argue in favour, as justice and a sense of security is the very foundation of human civilisation. Without this, vendors and their users would probably have greater challenges than the potential harm from unknown security vulnerabilities. Keeping knowledge of security vulnerabilities from disclosure also implicates keeping them from disclosure to the legal court. If the court can decide if the used method is acceptable (*forensically sound* [106]) and the acquired data is exact and without doubt authentic, the court should not need to know the details of the digital forensic method used.

Having argued for the use and secrecy of security vulnerabilities by law enforcement, we can turn to the complicated question of the sharing policy of such knowledge (Section 3.5). In a world where law enforcement

in different countries share methods and intelligence in an effort to fight global crime and terrorism, should methods based on security vulnerabilities be shared as well? Norway as a democratic country, with its laws and culture, might have a policy for using such methods that differ significantly with other countries. According to Norwegian policy for case handling in extradition cases [107], Norway are not extraditing prisoners to a country where a death penalty might get enforced. What is the Norwegian policy of sharing digital forensic methods, based on security vulnerabilities, with such countries? Should e.g. Norway share methods, or help in an investigation, with law enforcement in a country where possible suspect might be sentenced to the death penalty? Can, and should, such methods undergo the same strict regime? If so, how can this be enforced if such methods are kept from the public and *non* law enforcement disclosure? This dilemma is not as simple as taking a standing for or against death penalty. There's not a clear view if sharing such methods is inherently *good* or *bad*, according to *deontology*. Sharing information with the goal of justice and prevention of terrorism is clearly good, but if it leads to the death penalty for individuals, it's clearly not in accordance with the Norwegian policy. Turning to *consequentialism* doesn't help much either, as the consequences of sharing are unknown. Sharing a method with country A for solving one specific case might be unproblematic, but the sharing can not be withdrawn, leaving the next case in country A possibly problematic. Country A might even share the method with country B, as country A might not have the same strict sharing policy as Norway. Such sharing of both methods and intelligence information is of course relevant to many other areas besides law enforcement, and often demands strict diplomatic bilateral and multilateral relations. The sharing and use of offensive techniques, like using security vulnerabilities, might even be considered as digital *weapons* and should then be treated according to export control of military equipment [108]. This leaves us with a answer of contextual application; we should share methods based on security vulnerabilities with countries with similar laws and culture, and not with countries with e.g. the death penalty. However, the practical challenges and implications of trying to achieve this is beyond this essay.

3.7 Summary and Future Work

The increased complexity, security and ever expanding implications of consumer technology will have a significant effect on how digital forensic, as a tool used by e.g. law enforcement, will evolve in the next few years. Can law enforcement ever conclude they have acquired sufficient amounts of

digital data to enlighten a criminal investigation? What are great sources of digital intelligence and evidence, and how can law enforcement get access to all these sources? Mandatory security and encryption of consumer products is quickly becoming a big obstacle for law enforcement, leaving numerous cases with insufficient data for legal court, leaving the criminal investigation without a clear conclusion from law enforcement. Can justice be served without the court having access to vital data regarding an incident or a suspect? Will this increase in inaccessible data sources end up in more cases being dismissed, and in the end hurting our justice system?

We have argued that law enforcement must do everything in its power to gain access to all possibly important data sources in a criminal investigation, even if this includes (ab)using security vulnerabilities. We have also argued that law enforcement should keep such security vulnerabilities from disclosure to the public, legal courts and the vendors. Further we have seen that the development of methods based on security vulnerabilities can be considered *weaponising* the security vulnerabilities. In the wrong hands such digital weapons can do potential harm to vendors and their customers, so great effort should be made to keep them under a strict government policy. This implies that any sharing with law enforcement in other countries should be through controlled diplomatic bilateral and multilateral channels. This is to ensure that the policy of sharing, and using, such methods are controlled and decided by the government.

We firmly believe that law enforcement should be able to use digital forensics methods based on security vulnerabilities, to bypass security mechanisms and acquire secured data.

This challenge is complex and our discussion is only brief, thus this is important future work, to further discuss practical solutions that can be accepted from ethical, legal and technology stand points.

Chapter 4

Summary of Work

This chapter summarises the published papers included in this thesis and how these relate to the research questions from Section 1.5. A summary of the relationship between contributions and research questions can be seen in Figure 4.1.

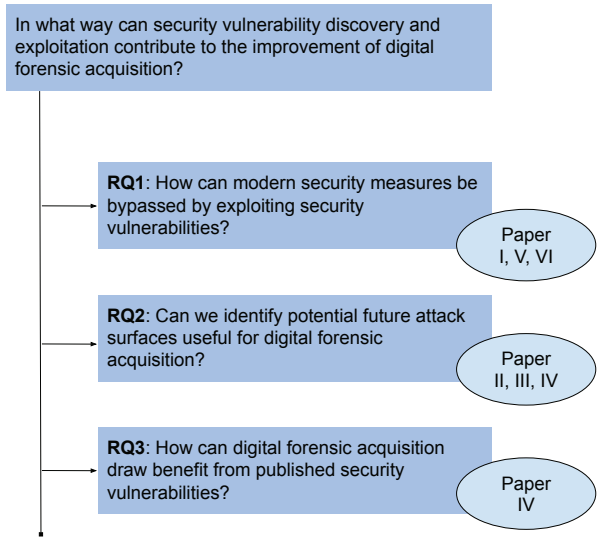


Figure 4.1: The relationship between research questions (RQ) and papers.

The contributions to answer the research questions are:

Paper I G. Alendal, G. O. Dyrkolbotn and S. Axelsson, 'Forensics acquisition - Analysis and circumvention of samsung secure boot enforced common criteria mode,' *Digital Investigation*, 2018 [1].
See appended paper in Part II, Paper I.

Paper II G. Alendal, S. Axelsson and G. O. Dyrkolbotn, 'Exploiting Vendor-Defined Messages in the USB Power Delivery Protocol,' in *Advances in Digital Forensics XV*, 2019 [2].
See appended paper in Part II, Paper II.

Paper III G. Alendal, S. Axelsson and G. O. Dyrkolbotn, 'Leveraging the USB Power Delivery Implementations for Digital Forensic Acquisition,' in *Advances in Digital Forensics XVII*, 2021 [3].
See appended paper in Part II, Paper III.

Paper IV G. Alendal, G. O. Dyrkolbotn and S. Axelsson, 'Digital Forensic Acquisition Kill Chain - Analysis and Demonstration,' in *Advances in Digital Forensics XVII*, 2021 [4].
See appended paper in Part II, Paper IV.

Paper V G. Alendal, S. Axelsson and G. O. Dyrkolbotn, 'Chip chop - smashing the mobile phone secure chip for fun and digital forensics,' *Forensic Science International: Digital Investigation*, 2021 [5].
See appended paper in Part II, Paper V.

Paper VI G. Alendal, 'Breaking Android Security by Abusing Implicit HW Trust,' In submission. [6].
See appended paper in Part II, Paper VI.

4.1 Main Research Question

In what way can security vulnerability discovery and exploitation contribute to the improvement of digital forensic acquisition?

As the main goal for Digital Forensic Acquisition (DFA) is to acquire data from any digital source, often with no knowledge of user credentials, any obstacle preventing this needs to be tackled. This involves interacting with the parts of a device that contains valuable data. This interaction

can be over physical connections to the device, like an USB interface or an Inter-Integrated Circuit (I^2C) interface on a chip after chip-off, or a wireless interface, like WiFi or Bluetooth. Often some form of logical communication protocol is used on top of such interfaces, like USB Power Delivery protocol over USB, or TCP/IP over WiFi. Our research focused on the concrete challenges with security schemes preventing DFA, potentially solvable using exploitation of security vulnerabilities.

The papers in this thesis address different aspects of the main challenges of accessing user data on mobile phones and contribute to answering the research questions in Section 1.5. We will discuss how our papers contribute to each research question in turn.

4.2 RQ1: How can modern security measures be bypassed by exploiting security vulnerabilities?

The main security mechanism protecting user data on modern mobile phones is encryption (Section 2.3.1). Thus bypassing encryption of user data is mandatory for successful DFA. Bypassing encryption often implies bypassing additional security measures, and RQ1.1 and RQ1.2 are tightly connected, as our results below show.

Generally speaking, there are two different scenarios where encryption can be bypassed: Before-First-Unlock (BFU) and After-First-Unlock (AFU) (Section 2.4). A DFA method for the BFU state is generally considered more valuable, as there's no requirement of the target device to be seized in a powered-on state, with at least one user unlock since the last device boot (Section 2.4).

Most contributions in this thesis are therefore focused on devices in the BFU state, including the powered off state. Our research contributes with several in-depth studies on some potential ways to achieve DFA on selected devices. This shows the complexity and experience needed for DFA to use security vulnerabilities, but that it's practically achievable, even with limited resources.

Breaking encryption in the BFU state can be done by gaining access to the Credential Encrypted (CE) encryption keys on the targeted devices or by recovering the user screen lock credentials. These encryption keys are the most valuable asset on the device and are often protected by many security features, often tied to the specific Hardware (HW), making chip-off useless.

There are numerous attack paths that could potentially lead to such a compromise of encrypted user data (Section 2.4) and to limit the scope of our research, we focused on logical security attack paths, like SW (in-

cluding FW) security vulnerabilities (Section 1.2). We focused on such SW security vulnerabilities, excluding HW based attacks and Side-channel Attack (SCA). SW attacks have the benefit of being easier to exploit when used in a DFA method, requiring less potentially damaging HW modifications. However, they might require more than one security vulnerability to be efficient.

This challenge often requires the execution of arbitrary code on the device, breaking the integrity of the device. Attacking the secure boot feature of modern mobile phones is one potential path to execute privileged / elevated attacker-provided code on the device, and we first researched this challenge on the most widespread mobile Operating System (OS), Android, on one of the most popular vendors, Samsung [55, 56]. In Paper I we discussed such protections that are present in the secure boot of Android devices, strengthening the device security. Secure boot makes sure the code integrity of the device is maintained throughout the boot cycle of a device and breaking this secure boot to introduce new code is a potential way to access user data. We demonstrated that we could fairly easy identify a security vulnerability, a design flaw, leading to the bypass of a security feature of this secure boot feature.

As the introduction of unauthorised code on a device in BFU state is not enough (Section 2.4), we studied how to bypass the security of the user screen lock, needed to unlock the CE storage. In Paper V and Paper VI we studied Embedded Secure Element (eSE) chips. These dedicated HW solutions are responsible for the protection of this crucial encryption key material, needed to decrypt user data. These stand-alone *system-within-the-system* chips are designed to withstand a fully compromised system and breaking the eSE security is needed to perform DFA without knowledge of user credentials. Paper V demonstrated how we could identify a new and previously unknown *0-day*, fully compromising the secure eSE chip protecting the crucial encryption keys. This attack demonstrated that a single researcher, with limited resources, could identify such a crucial security vulnerability and exploit it for successful DFA. The researched eSE was CC EAL 5+ certified, and our research shows a potential mismatch between *intended* and *achieved* security, encouraging security vulnerability research to develop new DFA methods. Our discovered *0-day* was reported to the vendor, Samsung, as part of the preparation for the publication of Paper V. Samsung acknowledged this serious issue and a corresponding security update, referenced as SVE-2020-18632/CVE-2020-28341, was released to remediate the flaw [109, 110].

Paper VI improved our attack from Paper V and demonstrated how a vulnerable security design, with implicit trust in eSE HW, opens up for a HW attack variant that removes the unauthorised and privileged code

execution assumption required in the attack in Paper V.

Our contribution in Paper I, Paper V and Paper VI demonstrates that a single researcher, with limited resources, can locate and exploit security vulnerabilities to develop new DFA methods on modern mobile phones.

4.3 RQ2: Can we identify potential future attack surfaces useful for digital forensic acquisition?

Discovery of future attack surfaces, especially in areas potentially not widely researched in the open security research community, can be very useful for DFA method development. A powerful actor, like LE, has access to resources and can perform physical attacks on devices, opening up attack surfaces not commonly available to attackers. Investing in resource demanding attacks and attacks requiring physical access can potentially be fruitful. The increasing complexity of device features, as well as the introduction of new features, all contribute to potential new attack surfaces being introduced. To illustrate this, we looked into potential attack surfaces with little or no public research available, like the USB Power Delivery (USB PD). We wanted to see if the lack of published security research on this subject was the result of this being a secure feature or simply an improbable attack surface.

Both Paper II and Paper III address RQ2.1 and RQ2.2, and demonstrated the USB PD as a potential attack surface, with different target architectures and with different security vulnerabilities. In Paper II we demonstrated access to debug features on selected Android devices that can be used to facilitate DFA. In Paper III we further researched this new attack surface on a different platform, iPhone devices by Apple. This paper demonstrated how a new attack surface could potentially be used to gain a foothold within the system, through Firmware (FW) and HW security vulnerabilities, with the potential to access and exploit the System-on-Chip (SoC), which again can give access to encryption keys. This shows the potential of seeking new paths to bypass the crucial encryption security feature, in this case the SoC. This paper demonstrates how a FW vulnerability could potentially give an attacker a foothold within the system, with the potential leverage of privileges by attacking the system from within. We believe we are the first to publish concrete research of security challenges with USB PD implementations.

Our contribution in Paper II and Paper III demonstrates that new attack surfaces can be discovered, useful for the development of new DFA methods on modern mobile phones.

4.4 RQ3: How can digital forensic acquisition benefit from published security vulnerabilities?

Much work has been published on preventing attacks on computer systems, as well as published security vulnerabilities, often with a published patch available for affected products. This raises the question if such *defensive* research and publicly available information could be used for *offensive* use in DFA, and if we could systematically use such information. Both *0-day* and such published *n-day* security vulnerabilities can be valuable sources for the development of new DFA methods, much because of the ability LE has to seize devices, preventing patches for future *n-day* security vulnerabilities to be applied to devices. These *n-day* security vulnerabilities can be equally efficient in specific cases, saving resources and time compared to the discovery of new *0-day* security vulnerabilities. We believe this is important to take advantage of.

To answer RQ3.1, Paper IV introduced a new methodology, Digital Forensic Acquisition Kill Chain (DFAKC), in an attempt to be more efficient in developing new digital forensic acquisition methods based on all security vulnerabilities, including published, *n-day*, vulnerabilities. One of the goals was to benefit from the continuous device updates and patching that takes place, potentially for challenges that are relevant for digital forensic acquisition. As a powerful user of digital forensic acquisition, like LE, has the ability to prevent a seized device from installing a published patch, the same patch can be reverse engineered to rediscover security vulnerabilities. These rediscovered security vulnerabilities can be used to develop a new digital forensic acquisition method for use on already seized, and thus vulnerable, devices. Paper IV also demonstrated how, using our proposed DFAKC, we could identify both the attack surface and an *n-day* security vulnerability by studying patches released for a particular router FW.

Our contribution in Paper IV takes advantage of combining the tracking of published *n-day* vulnerabilities and researching *0-day* vulnerabilities, balancing the use of limited LE resources.

4.5 Additional Work

4.5.1 Additional Research Presentations and Awards

Our research in Paper V was also accepted and presented at the Black Hat Briefings security conference [111]. This is a major security industry conference aimed at a general security professional audience and demonstrates the interest in our research from a community beyond digital forensics.

Our research in Paper V was in addition awarded "Best Client-Side Bug" at the Pwnie Awards [112]. The award acknowledges the importance and impact of our research results from a general security perspective.

4.5.2 Unpublished Work

During research lab work, new and unpublished *0-day* vulnerabilities were discovered. These were developed into new and unique DFA tools for LE. These vulnerabilities and tools are not made public and details are therefore redacted from this thesis. However, their success confirms the applicability for use by LE in the development of new DFA methods and thus to the answer of research questions in Section 1.5.

4.5.3 Preliminary Work

Leading up to and motivating the start of this thesis work, was earlier research we have done on self-encrypting hard drives [113]. This work was also presented at the Hardwear.io conference [114]. This research demonstrated many security vulnerabilities in the implementations of Western Digital HW encrypted drives, bypassing the encryption layer to secure user data without knowing the user credentials. The research discovered weak AES key generations, weak implementation of authentication schemes and even discovered backdoors to completely bypass authentication.

This work was a great motivation to kick off this thesis research as it demonstrates the often huge difference between *intended/claimed* security and *actual* security. HW encryption could simply be broken using security vulnerabilities and reverse engineering. This work also attracted some media attention [115–118] and was a good experience in how vendors, in this case Western Digital, handled such critical security vulnerabilities and "responsible disclosure" [119, 120].

Chapter 5

Discussion and Conclusion

This thesis contributes towards a more realistic view of the challenges Digital Forensic Acquisition (DFA) are facing with the increased mandatory security of embedded devices.

The contributions in this thesis can be identified in the following areas:

DFA Challenges: Study of concrete security measurements preventing successful DFA on modern embedded devices: confidentiality and code integrity. Contributing to a better understanding of concrete challenges with mobile device encryption on a.o. Android devices.

Vulnerability Discovery and Exploitation in DFA: Demonstrated that it is feasible for LE to discover new *0-day* security vulnerabilities and develop new DFA techniques based on the exploitation of these. Our results show that even with limited resources, within a PhD research scope, one can achieve such results, motivating further research into this area.

New Attack Surfaces for DFA: Contribution towards increasing the attack surface for DFA. Demonstrating that new functionality is likely to be a new opportunity for DFA.

Improving DFA Development and Success Rate: A new methodology, DFAKC, for LE to prioritise and systematise continuous development of DFA methods, utilising both *0-day* and *n-day* security vulnerabilities.

Ethical Discussions: Contribute to the discussion on the ethical dilemma with security vulnerabilities discovered by LE.

5.1 DFA Challenges

Following the *bottom-up* approach from Section 1.4 and Section 2.4, we conducted research into identifying actual security schemes that prevent successful DFA. With experimental cases and case study of actual implementations in Papers I, V and VI, we demonstrated and confirmed concrete examples where confidentiality and the encryption of user data is a big challenge for DFA. Other challenges, like code integrity and signed FW updates, add to this as secondary challenges when trying to bypass confidentiality of user data, further raising the bar. DFA can bypass encryption by securing unencrypted data from a device in an After-First-Unlock (AFU) state or by recovery of the user screen lock or encryption key material from a device in a Before-First-Unlock (BFU) state. Our results indicate that DFA methods targeted against devices in the BFU state are achievable and preferable, as they do not require the device to be seized in a powered on state, with at least one user unlock since last boot (AFU).

5.2 Vulnerability Discovery and Exploitation in DFA

Our contributions demonstrate how to discover and exploit both *0-day* and *n-day* security vulnerabilities to aid in DFA and show that exploitation of security vulnerabilities is feasible, even with the limited resources of this research project. We showed how to discover new *0-day*s in both the integrity (secure boot) of the device and the confidentiality, breaking the Embedded Secure Element (eSE) HW. As this was possible in our research with limited resources, we firmly believe that this is possible by dedicated authorities with far more resources available, like LE. We also developed the *0-day*s vulnerabilities into fully working exploits, ready to be used in DFA methods, again with the limited resources our research operates within.

To be able to locate security vulnerabilities and attack devices in BFU state, our research (Paper V) shows that bypassing encryption often involves bypassing integrity as well, as one requirement for our attack was to bypass the integrity of the device, to be able to run an attacking process in the REE, to communicate with and attack the eSE chip. Paper I exemplified this by attacking features that protect the integrity of the device. However, we believe such attacks on integrity might be removed as a requirement for DFA in some cases. In Paper VI we introduced a variant of our original attack on the eSE, that abuses the implicit trust relationship between the overall system and the trusted eSE HW. This removes the need for an additional attack on the integrity of the general device and demonstrates that we only need to locate *one* security vulnerability in the eSE to

fully bypass the encryption challenge on affected devices.

5.3 New Attack Surfaces for DFA

Our research into USB Power Delivery (USB PD) demonstrates how a resourceful attacker, with physical access to devices, can expand the attack surface by looking at additional ways to influence devices to attack the integrity and confidentiality. New attack surfaces might be introduced by new features and technology, like the USB PD. Much of the open security research community focuses on attack vectors with high spreading effect. A vulnerability in a device's way to handle e.g. an SMS or an email, distributed to the device remotely by any user/attacker, is far more serious, seen from a security perspective, than a local vulnerability in the USB bus, where the attacker needs physical access to the device. As LE has the ability to seize devices, all remote *and* local attack vectors are equally valuable and there might be more uncovered attack surfaces in the less researched physical attack surfaces. Paper II demonstrates one such new attack surface by locating hidden debug interfaces on selected Android devices, very useful for DFA, by manipulating the Vendor Defined Message (VDM) feature in the USB Power Delivery protocol. This protocol has not gained much attention in the security community and our research was, as far as we know, the first to do so. Paper III demonstrates the USB Power Delivery protocol on a different target, the Apple iPhone. In this paper we demonstrate how this attack vector represents a path to a possible exploitation of the SoC of the device, by locating security vulnerabilities in the FW and update mechanism. This enabled us to overtake the USB PD HW, with the possibility to abuse the implicit trust this chip has in the iPhone system or any interfaces, with potential data parsing security vulnerabilities, with other parts of the system.

We have demonstrated that new attack surfaces, like the USB Power Delivery, could reveal new attacks for use in DFA and should be researched further.

5.4 Improving DFA Method Development

The need for a methodical approach for efficient use of both *0-day* and *n-day* security vulnerabilities in the development of new DFA methods was emphasised when performing our research, as *n-day* security vulnerabilities can be very effective in combination with LE's ability to prevent patching of seized devices.

In addition, to be efficient in solving challenges with specific devices

in case-to-case work as well as more long term and case-independent trends in consumer adaptations of new technology, there is a need to use available resources in an efficient way. The amount of seized devices will probably increase and so will their security and complexity. Acquiring data from devices and services is simply getting more challenging and requires more (human) resources. In Paper IV we proposed a methodology, DFAKC, in an attempt to identify these challenges and to improve the use of resources to increase the success of developing new DFA methods based on security vulnerabilities. This challenge is not easily solved, as conflicting interests pull on the same resources. There is always an expectation that LE will use "every resource available" to solve any serious criminal case, and at the same time solve more long term challenges for future cases, like the encryption challenge of modern mobile phones. DFAKC tries to improve the prioritisation between such conflicting interests. Our method also suggests ways to improve the efficiency in discovering *n-day* security vulnerabilities, as these are often far less resource demanding to (re)discover than to discover new *0-day*s. Thus going for the "low hanging" *n-day* security vulnerability discovery is more fruitful because of the "patch preventing" capability of LE. We expect DFAKC to mature over time as more experience is gained through its use, but hope this is a positive beginning for improved use of security vulnerabilities in DFA method development.

5.5 Ethical Discussions

The "equity issue" and ethical dilemma of discovering *0-day* security vulnerabilities, as introduced in Chapter 1 and discussed in Chapter 3, is challenging for any government organisation, especially for LE. It is expected by the community that law enforcement protects its citizens from threats, also threats in the digital sphere. How LE handles the discovery of *0-day* security vulnerabilities is of course important for the technical challenge it helps to solve, such as bypassing a security scheme to acquire data from a suspect's phone. LE also has to handle the public opinion on their policy. If such *0-day*s are kept from disclosure to vendors for patching and this is publicly known at a later stage, the public might disagree with the policy. This calls for an open discussion on such matters, as this is an important political question in addition to the technical one.

5.6 Conclusions and Future Work

Our research has contributed to a better understanding of the growing challenges mandatory security of consumer devices has for the future of

DFA. Motivated by the increasing complexity of features and devices, as well as the seemingly growing number of discovered security vulnerabilities, we have demonstrated that the use of security vulnerabilities in DFA is a fruitful path forward.

As our research project only studied isolated challenges, with limited resources available, but still was able to locate and exploit *0-day* security vulnerabilities, should motivate further research. Our research discovered new vulnerabilities, developed working exploits as well as demonstrated how to use exploit results in an off-device brute force attack. We believe that further research could be performed in each of these different phases, from the discovery to the effective exploitation and use of security vulnerabilities.

More targeted research into existing attack surfaces, in addition to the discovery of new attack surfaces, is also needed. Expanding the attacks to include attacks based on HW and/or SCA should also increase the possibilities for new DFA methods. The challenge with implicit HW trust also seems like an area where security vulnerabilities can be discovered. Isolating all trust in a single eSE chip did not work out as intended for the devices in our research, but the amount of work and resources needed to uncover such vulnerable HW trust relations could be much higher for other devices. Our research into this problematic HW trust was also limited by the lack of HW resources to demonstrate the full attack. Thus more research might need to be done in unifying the SW, FW and HW expertise. Combining forces to develop new DFA methods based on attack chains utilising resources, experience and security vulnerabilities in all these attack domains, from logical design and implementation bugs in an application, down to physical gates in an Integrated circuit (IC).

Bibliography

[1] G. Alendal, G. O. Dyrkolbotn and S. Axelsson, 'Forensics acquisition - Analysis and circumvention of samsung secure boot enforced common criteria mode,' *Digital Investigation*, vol. 24, S60–S67, 2018, ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2018.01.008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1742287618300409.

[2] G. Alendal, S. Axelsson and G. O. Dyrkolbotn, 'Exploiting Vendor-Defined Messages in the USB Power Delivery Protocol,' in *Advances in Digital Forensics XV*, G. Peterson and S. Shenoi, Eds., Springer International Publishing, 2019, pp. 101–118, ISBN: 978-3-030-28752-8.

[3] G. Alendal, S. Axelsson and G. O. Dyrkolbotn, 'Leveraging the USB Power Delivery Implementations for Digital Forensic Acquisition,' in *Advances in Digital Forensics XVII*, G. Peterson and S. Shenoi, Eds., Springer International Publishing, 2021, pp. 111–133, ISBN: 978-3-030-88381-2.

[4] G. Alendal, G. O. Dyrkolbotn and S. Axelsson, 'Digital Forensic Acquisition Kill Chain - Analysis and Demonstration,' in *Advances in Digital Forensics XVII*, G. Peterson and S. Shenoi, Eds., Springer International Publishing, 2021, pp. 3–19, ISBN: 978-3-030-88381-2.

[5] G. Alendal, S. Axelsson and G. O. Dyrkolbotn, 'Chip chop - smashing the mobile phone secure chip for fun and digital forensics,' *Forensic Science International: Digital Investigation*, vol. 37, p. 301 191, 2021, ISSN: 2666-2817. DOI: https://doi.org/10.1016/j.fsidi.2021.301191. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666281721000998.

[6] G. Alendal, 'Breaking Android Security by Abusing Implicit HW Trust,' In submission.

[7] Venturebeat, *Apple vs. fbi: A timeline of the iphone encryption case*, [Online; accessed 20-June-2018], 2016. [Online]. Available: http://venturebeat.com/2016/02/19/apple-fbi-timeline/.

[8] The MITRE Corporation, *Glossary - vulnerability*, https://www.cve.org/ResourcesSupport/Glossary?activeTerm=glossaryVulnerability, [Online; accessed 19-November-2021], 2021.

[9] S. Skorobogatov and C. Woods, 'Breakthrough silicon scanning discovers backdoor in military chip,' in *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2012, pp. 23–40.

[10] D. J. Bernstein, T. Lange and R. Niederhagen, 'Dual ec: A standardized back door,' in *The New Codebreakers*, Springer, 2016, pp. 256–281.

[11] CBS News, *Cbs news poll: Americans split on unlocking san bernardino shooter's iphone*, [Online; accessed 20-June-2018], 2016. [Online]. Available: https://www.cbsnews.com/news/cbs-news-poll-americans-split-on-unlocking-san-bernardino-shooters-iphone/.

[12] CNN, *Apple opposes judge's order to hack san bernardino shooter's iphone*, [Online; accessed 20-June-2018], 2016. [Online]. Available: https://edition.cnn.com/2016/02/16/us/san-bernardino-shooter-phone-apple/.

[13] Wikipedia contributors, *Fbi apple encryption dispute*, [Online; accessed 20-June-2018], 2018. [Online]. Available: https://en.wikipedia.org/w/index.php?title=FBI%E2%80%93Apple_encryption_dispute&oldid=844488919.

[14] The Washington Post, *The fbi wanted to unlock the san bernardino shooter's iphone. it turned to a little-known australian firm.* [Online; accessed 19-May-2021], 2021. [Online]. Available: https://www.washingtonpost.com/technology/2021/04/14/azimuth-san-bernardino-apple-iphone-fbi/.

[15] H. F. Tayeb and C. Varol, 'Android mobile device forensics: A review,' in *2019 7th International Symposium on Digital Forensics and Security (ISDFS)*, 2019, pp. 1–7. DOI: 10.1109/ISDFS.2019.8757493.

[16] A. Al-Sabaawi and E. Foo, 'A comparison study of android mobile forensics for retrieving files system,' *International Journal of Computer Science and Security (IJCSS)*, vol. 13, no. 4, pp. 148–166, 2019.

[17] A. Ghosh, K. Majumder and D. De, 'Android forensics using sleuth kit autopsy,' in *Proceedings of the Sixth International Conference on Mathematics and Computing*, D. Giri, R. Buyya, S. Ponnusamy, D. De, A. Adamatzky and J. H. Abawajy, Eds., Singapore: Springer Singapore, 2021, pp. 297–308.

[18] H. H. Lwin, W. P. Aung and K. K. Lin, 'Comparative analysis of android mobile forensics tools,' in *2020 IEEE Conference on Computer Applications(ICCA)*, 2020, pp. 1–6. DOI: 10.1109/ICCA49400.2020.9022838.

[19] B.-J. Koops and E. Kosta, 'Looking for some light through the lens of "cryptowar" history: Policy options for law enforcement authorities against "going dark",' *Computer Law & Security Review*, vol. 34, no. 4, pp. 890–900, 2018, ISSN: 0267-3649. DOI: https://doi.org/10.1016/j

.clsr.2018.06.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0267364918302413.

[20] A. Ozment and S. E. Schechter, 'Milk or wine: Does software security improve with age?' In *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, ser. USENIX-SS'06, Vancouver, B.C., Canada: USENIX Association, 2006.

[21] J. Armin, P. Foti and M. Cremonini, '0-day vulnerabilities and cybercrime,' in *2015 10th International Conference on Availability, Reliability and Security*, IEEE, 2015, pp. 711–718.

[22] S. Skorobogatov, 'The bumpy road towards iphone 5c nand mirroring,' *arXiv preprint arXiv:1609.04327*, 2016.

[23] M. Fidler, 'Regulating the zero-day vulnerability trade: A preliminary analysis,' *ISJLP*, vol. 11, p. 405, 2015.

[24] P. N. Stockton and M. Golabek-Goldman, 'Curbing the market for cyber weapons,' *Yale L. & Pol'y Rev.*, vol. 32, p. 239, 2013.

[25] S. M. Bellovin, M. Blaze, S. Clark and S. Landau, 'Lawful hacking: Using existing vulnerabilities for wiretapping on the internet,' *Nw. J. Tech. & Intell. Prop.*, vol. 12, p. 1, 2014.

[26] M. Daniel, *Heartbleed: Understanding when we disclose cyber vulnerabilities*, https://obamawhitehouse.archives.gov/blog/2014/04/28/heartbleed-understanding-when-we-disclose-cyber-vulnerabilities, 2014.

[27] White House, *Vulnerabilities equities policy and process for the united states government*, https://www.whitehouse.gov/sites/whitehouse.gov/files/images/External-UnclassifiedVEPCharterFINAL.PDF, 2017.

[28] A. Friedman, T. Moore and A. Procaccia, 'Cyber-sword v. cyber-shield: The dynamics of us cybersecurity policy priorities,' *Under Review*, 2010.

[29] B. Schneier, *Disclosing vs. hoarding vulnerabilities." schneier on security, may 22, 2014*, https://www.schneier.com/blog/archives/2014/05/disclosing_vs_h.html, 2014.

[30] Council of the European Union, *Draft council resolution on encryption - security through encryption and security despite encryption*, https://files.orf.at/vietnam2/files/fm4/202045/783284_fh_st12143-re01en20_783284.pdf, 2020.

[31] W. Diffie and S. Landau, *Privacy on the line: The politics of wiretapping and encryption*. The MIT Press, 2010.

[32] GDPR.EU, *General data protection regulation (gdpr)*, https://gdpr.eu/tag/gdpr/ [Online; accessed 10-December-2021], 2021. [Online]. Available: https://gdpr.eu/tag/gdpr/.

[33] A. Fukami, R. Stoykova and Z. Geradts, 'A new model for forensic data extraction from encrypted mobile devices,' *Forensic Science International: Digital Investigation*, vol. 38, p. 301 169, 2021, ISSN: 2666-2817. DOI: https://doi.org/10.1016/j.fsidi.2021.301169. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666281721000779.

[34] C. Liguori, 'Exploring lawful hacking as a possible answer to the 'going dark' debate,' *Mich. Tech. L. Rev.*, vol. 26, p. 317, 2019.

[35] Regjeringen.no, *Dataavlesing*, https://www.regjeringen.no/no/dokumenter/prop.-68-l-20152016/id2479232/?ch=14 [Online; accessed 10-December-2021], 2021. [Online]. Available: https://www.regjeringen.no/no/dokumenter/prop.-68-l-20152016/id2479232/?ch=14.

[36] D. Craigen, N. Diakun-Thibault and R. Purse, 'Defining cybersecurity,' *Technology Innovation Management Review*, vol. 4, no. 10, 2014.

[37] M. La Polla, F. Martinelli and D. Sgandurra, 'A survey on security for mobile devices,' *IEEE Communications Surveys Tutorials*, vol. 15, no. 1, pp. 446–471, 2013. DOI: 10.1109/SURV.2012.013012.00028.

[38] P. Sun, L. Garcia, G. Salles-Loustau and S. Zonouz, 'Hybrid firmware analysis for known mobile and iot security vulnerabilities,' in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2020, pp. 373–384. DOI: 10.1109/DSN48063.2020.00053.

[39] R. Mayrhofer, 'An architecture for secure mobile devices,' *Security and Communication Networks*, vol. 8, no. 10, pp. 1958–1970, 2015.

[40] M. A. Almaiah, A. Al-Zahrani, O. Almomani and A. K. Alhwaitat, 'Classification of cyber security threats on mobile devices and applications,' in *Artificial Intelligence and Blockchain for Future Cybersecurity Applications*, Y. Maleh, Y. Baddi, M. Alazab, L. Tawalbeh and I. Romdhani, Eds. Cham: Springer International Publishing, 2021, pp. 107–123, ISBN: 978-3-030-74575-2. DOI: 10.1007/978-3-030-74575-2_6. [Online]. Available: https://doi.org/10.1007/978-3-030-74575-2_6.

[41] D. R. Thomas, A. R. Beresford and A. Rice, 'Security metrics for the android ecosystem,' in *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2015, pp. 87–98.

[42] J. Wu, S. Liu, S. Ji, M. Yang, T. Luo, Y. Wu and Y. Wang, 'Exception beyond exception: Crashing android system by trapping in "uncaught exception",' in *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, 2017, pp. 283–292. DOI: 10.1109/ICSE-SEIP.2017.12.

[43] A. Mazuera-Rozo, J. Bautista-Mora, M. Linares-Vásquez, S. Rueda and G. Bavota, 'The android os stack and its vulnerabilities: An empirical study,' *Empirical Software Engineering*, vol. 24, no. 4, pp. 2056–2101, 2019.

[44] K.-S. Lhee and S. J. Chapin, 'Buffer overflow and format string overflow vulnerabilities,' *Software: practice and experience*, vol. 33, no. 5, pp. 423–460, 2003.

[45] A. One, 'Smashing the stack for fun and profit,' *Phrack*, vol. 7, no. 49, Nov. 1996, http://www.phrack.com/issues.html?issue=49&id=14.

[46] I. Goicoechea-Telleria, J. Liu-Jimenez, R. Sanchez-Reillo and W. Ponce-Hemandez, 'Vulnerabilities of biometric systems integrated in mobile devices: An evaluation,' in *2016 IEEE International Carnahan Conference on Security Technology (ICCST)*, 2016, pp. 1–8. DOI: 10.1109/CCST.2016.7815677.

[47] Z. Zahid, A. Haider, N. Sabahat and A. Tanwir, 'Vulnerabilities in biometric authentication of smartphones,' in *2020 IEEE 23rd International Multitopic Conference (INMIC)*, 2020, pp. 1–5. DOI: 10.1109/INMIC50486.2020.9318094.

[48] R. Hay, 'Fastboot oem vuln: Android bootloader vulnerabilities in vendor customizations,' in *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17)*, 2017.

[49] M. Linares-Vásquez, G. Bavota and C. Escobar-Velásquez, 'An empirical study on android-related vulnerabilities,' in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, IEEE, 2017, pp. 2–13.

[50] G. Alendal, C. Kison and modg, *got HW crypto? On the (in)security of a Self-Encrypting Drive series*, Cryptology ePrint Archive, Report 2015/1002, http://eprint.iacr.org/2015/1002, 2015.

[51] P. Mishra, S. Bhunia and M. Tehranipoor, *Hardware IP security and trust*. Springer, 2017.

[52] *Common criteria*, https://www.commoncriteriaportal.org/.

[53] *Owasp*, https://www.owasp.org/.

[54] E. Eilam, *Reversing: secrets of reverse engineering*. John Wiley & Sons, 2011.

[55] Statcounter, *Mobile operating system market share worldwide*, https://gs.statcounter.com/os-market-share/mobile/worldwide [Online; accessed 10-December-2021], 2021. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide.

[56] NetApplications.com, *Operating system market share*, https://www.netmarketshare.com/operating-system-market-share.aspx?id=platformsMobile [Online; accessed 10-December-2021], 2021. [Online]. Available: https://www.netmarketshare.com/operating-system-market-share.aspx?id=platformsMobile.

[57] B. Carrier, 'Defining digital forensic examination and analysis tools,' *International Journal of Digital Evidence*, vol. 1, p. 2003, 2002.

[58] D. Quick and K.-K. R. Choo, 'Impacts of increasing volume of digital forensic data: A survey and future research challenges,' *Digital Investigation*, vol. 11, no. 4, pp. 273–294, 2014.

[59] D. Lillis, B. Becker, T. O'Sullivan and M. Scanlon, 'Current challenges and future research areas for digital forensic investigation,' *arXiv preprint arXiv:1604.03850*, 2016.

[60] D. Quick and K.-K. R. Choo, 'Big forensic data reduction: Digital forensic images and electronic evidence,' *Cluster Computing*, vol. 19, no. 2, pp. 723–740, 2016.

[61] L. Caviglione, S. Wendzel and W. Mazurczyk, 'The future of digital forensics: Challenges and the road ahead,' *IEEE Security & Privacy*, vol. 15, no. 6, pp. 12–17, 2017.

[62] R. Van Baar, H. M. van Beek and E. Van Eijk, 'Digital forensics as a service: A game changer,' *Digital Investigation*, vol. 11, S54–S62, 2014.

[63] S. L. Garfinkel, 'Digital forensics research: The next 10 years,' *Digit. Investig.*, vol. 7, S64–S73, Aug. 2010, ISSN: 1742-2876. DOI: 10.1016/j.diin.2010.05.009. [Online]. Available: http://dx.doi.org/10.1016/j.diin.2010.05.009.

[64] E. Casey and G. J. Stellatos, 'The impact of full disk encryption on digital forensics,' *ACM SIGOPS Operating Systems Review*, vol. 42, no. 3, pp. 93–98, 2008.

[65] E. Casey, G. Fellows, M. Geiger and G. Stellatos, 'The growing impact of full disk encryption on digital forensics,' *Digital Investigation*, vol. 8, no. 2, pp. 129–134, 2011.

[66] E. A. Vincze, 'Challenges in digital forensics,' *Police Practice and Research*, vol. 17, no. 2, pp. 183–194, 2016.

[67] R. Montasari and R. Hill, 'Next-generation digital forensics: Challenges and future paradigms,' in *2019 IEEE 12th International Conference on Global Security, Safety and Sustainability (ICGS3)*, IEEE, 2019, pp. 205–212.

[68] Google, *File-based encryption*, https://source.android.com/security/encryption/file-based [Online; accessed 21-September-2020], 2020. [Online]. Available: https://source.android.com/security/encryption/file-based.

[69] N. Scrivens and X. Lin, 'Android digital forensics: Data, extraction and analysis,' in *Proceedings of the ACM Turing 50th Celebration Conference - China*, ser. ACM TUR-C '17, Shanghai, China: ACM, 2017, 26:1–26:10, ISBN: 978-1-4503-4873-7. DOI: 10.1145/3063955.3063981. [Online]. Available: http://doi.acm.org/10.1145/3063955.3063981.

[70] S. J. Yang, J. H. Choi, K. B. Kim and T. Chang, 'New acquisition method based on firmware update protocols for android smartphones,' *Digital Investigation*, vol. 14, no. Supplement 1, S68–S76, 2015, The Proceedings of the Fifteenth Annual DFRWS Conference, ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2015.05.008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1742287615000535.

[71] S. J. Yang, J. H. Choi, K. B. Kim, R. Bhatia, B. Saltaformaggio and D. Xu, 'Live acquisition of main memory data from android smartphones and smartwatches,' *Digital Investigation*, 2017, ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2017.09.003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1742287617301317.

[72] A. Ali-Gombe, S. Sudhakaran, A. Case and G. G. R. III, 'Droidscraper: A tool for android in-memory object recovery and reconstruction,' in *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*, Chaoyang District, Beijing: USENIX Association, Sep. 2019, pp. 547–559, ISBN: 978-1-939133-07-6. [Online]. Available: https://www.usenix.org/conference/raid2019/presentation/ali-gombe.

[73] J. Sylve, A. Case, L. Marziale and G. G. Richard, 'Acquisition and analysis of volatile memory from android devices,' *Digital Investigation*, vol. 8, no. 3, pp. 175–184, 2012, ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2011.10.003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1742287611000879.

[74] M. Zalewski, *Memfetch - linux on-demand process image dump*, [Online; accessed 16-November-2020], 2014. [Online]. Available: https://github.com/citypw/lcamtuf-memfetch/.

[75] B. Taubmann, M. Huber, S. Wessel, L. Heim, H. P. Reiser and G. Sigl, 'A lightweight framework for cold boot based forensics on mobile devices,' in *2015 10th International Conference on Availability, Reliability and Security*, 2015, pp. 120–128. DOI: 10.1109/ARES.2015.47.

[76] B. Carrier, *Open source digital forensics tools: The legal argument*, 2002.

[77] S. Raghavan, 'Digital forensic research: Current state of the art,' *CSI Transactions on ICT*, vol. 1, no. 1, pp. 91–114, 2013, ISSN: 2277-9086. DOI: 10.1007/s40012-012-0008-7. [Online]. Available: http://dx.doi.org/10.1007/s40012-012-0008-7.

[78] R. Montasari and R. Hill, 'Next-generation digital forensics: Challenges and future paradigms,' in *2019 IEEE 12th International Conference on Global Security, Safety and Sustainability (ICGS3)*, 2019, pp. 205–212. DOI: 10.1109/ICGS3.2019.8688020.

[79] FORMOBILE, *From mobile phones to court*, [Online; accessed 13-November-2020], 2020. [Online]. Available: https://formobile-project.eu/.

[80] EXFILES, *Europe fights against crime and terrorism*, [Online; accessed 13-November-2020], 2020. [Online]. Available: https://exfiles.eu/about/.

[81] L. Hatton, 'Reexamining the fault density-component size connection,' *IEEE Softw.*, vol. 14, no. 2, pp. 89–97, Mar. 1997, https://ieeexplore.ieee.org/abstract/document/582978/, ISSN: 0740-7459. DOI: 10.1109/52.582978. [Online]. Available: https://doi.org/10.1109/52.582978.

[82] A. Tomlinson, 'Introduction to the tpm,' in *Smart Cards, Tokens, Security and Applications*, Springer, 2008, pp. 155–172.

[83] B. McGillion, T. Dettenborn, T. Nyman and N. Asokan, 'Open-tee–an open virtual trusted execution environment,' in *2015 IEEE Trustcom/BigDataSE/ISPA*, IEEE, vol. 1, 2015, pp. 400–407.

[84] V. Costan and S. Devadas, 'Intel sgx explained.,' *IACR Cryptology ePrint Archive*, vol. 2016, no. 086, pp. 1–118, 2016.

[85] ARM, *Trustzone model*, [Online; accessed 12-Jan-2021], 2009. [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0301h/Chdebaee.html.

[86] M. Vauclair, 'Secure element,' in *Encyclopedia of Cryptography and Security*. Boston, MA: Springer US, 2011, pp. 1115–1116, https://doi.org/10.1007/978-1-4419-5906-5_303 [Online; accessed 01-September-2020], ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_303. [Online]. Available: https://doi.org/10.1007/978-1-4419-5906-5_303.

[87] A. P. Fournaris and G. Keramidas, 'From hardware security tokens to trusted computing and trusted systems,' in *System-Level Design Methodologies for Telecommunication*. Cham: Springer International Publishing, 2014, pp. 99–117, https://doi.org/10.1007/978-3-319-00663-5_6, ISBN: 978-3-319-00663-5. DOI: 10.1007/978-3-319-00663-5_6. [Online]. Available: https://doi.org/10.1007/978-3-319-00663-5_6.

[88] R. Anderson, 'Cryptography and competition policy: Issues with 'trusted computing',' in *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, https://dl.acm.org/doi/abs/10.1145/872035.872036, 2003, pp. 3–10.

[89] Google, *Android code search*, https://cs.android.com/android/platform/superproject/+/master:frameworks/base/services/core/java/com/android/server/locksettings/SyntheticPasswordManager.java [Online; accessed 21-September-2020], 2020. [Online]. Available: https://cs.android.com/android/platform/superproject/+/master:frameworks/base/services/core/java/com/android/server/locksettings/SyntheticPasswordManager.java.

[90] R. Mayrhofer, J. V. Stoep, C. Brubaker and N. Kralevich, 'The android platform security model,' *arXiv preprint arXiv:1904.05572*, 2019, https://arxiv.org/abs/1904.05572 [Online; accessed 04-September-2020].

[91] ScienceDirect, *Cryptanalysis*, https://www.sciencedirect.com/topics/computer-science/cryptanalysis [Online; accessed 01-December-2021], 2021. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/cryptanalysis.

[92] S. Gunasekera, 'Rooting your android device,' in *Android Apps Security: Mitigate Hacking Attacks and Security Breaches*. Berkeley, CA: Apress, 2020, pp. 173–223, https://doi.org/10.1007/978-1-4842-1682-8_8, ISBN: 978-1-4842-1682-8. DOI: 10.1007/978-1-4842-1682-8_8. [Online]. Available: https://doi.org/10.1007/978-1-4842-1682-8_8.

[93] Google, *Android security features*, https://source.android.com/security/features [Online; accessed 18-November-2021], 2021. [Online]. Available: https://source.android.com/security/features.

[94] S.-T. Sun, A. Cuadros and K. Beznosov, 'Android rooting: Methods, detection, and evasion,' in *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2015, pp. 3–14.

[95] J. Yao and V. Zimmer, 'Os resiliency,' in *Building Secure Firmware: Armoring the Foundation of the Platform*. Berkeley, CA: Apress, 2020, pp. 185–196, ISBN: 978-1-4842-6106-4. DOI: 10.1007/978-1-4842-6106-4_6. [Online]. Available: https://doi.org/10.1007/978-1-4842-6106-4_6.

[96] D. Moghimi, B. Sunar, T. Eisenbarth and N. Heninger, 'TPM-FAIL:TPM meets timing and lattice attacks,' in *29th USENIX Security Symposium (USENIX Security 20)*, https://www.usenix.org/conference/usenixsecurity20/presentation/moghimi-tpm, 2020, pp. 2057–2073.

[97] R. Anderson, M. Bond, J. Clulow and S. Skorobogatov, 'Cryptographic processors - a survey,' *Proceedings of the IEEE*, vol. 94, no. 2, pp. 357–369, 2006, https://ieeexplore.ieee.org/document/1580505.

[98] M. Bond and R. Anderson, 'Api-level attacks on embedded systems,' *Computer*, vol. 34, no. 10, pp. 67–75, 2001, https://ieeexplore.ieee.org/abstract/document/955101/.

[99] J. S. Jang, S. Kong, M. Kim, D. Kim and B. B. Kang, 'Secret: Secure channel between rich execution environment and trusted execution environment.,' in *NDSS*, 2015.

[100] C. Miller, 'Exploring the nfc attack surface,' *Proceedings of Blackhat*, 2012.

[101] C.-Y. Chao, H. C. Su and C.-Y. Wu, 'Breaking samsung's root of trust: Exploiting samsung s10 secure boot,' *Black Hat USA*, 2020, https://www.blackhat.com/us-20/briefings/schedule/#breaking-samsungs-root-of-trust-exploiting-samsung-s-secure-boot-20290.

[102] N. Artenstein, 'Broadpwn: Remotely compromising android and ios via a bug in broadcom's wi-fi chipsets,' *Black Hat USA*, 2017.

[103] V. A. Padaryan, V. Kaushan and A. Fedotov, 'Automated exploit generation for stack buffer overflow vulnerabilities,' *Programming and Computer Software*, vol. 41, no. 6, pp. 373–380, 2015.

[104] Wikipedia contributors, *Deontological ethics — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Deontological_ethics&oldid=846118213, [Online; accessed 27-June-2018], 2018.

[105] Wikipedia contributors, *Consequentialism — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Consequentialism&oldid=844338720, [Online; accessed 27-June-2018], 2018.

[106] R. McKemmish, 'When is digital evidence forensically sound?' In *Advances in Digital Forensics IV*, I. Ray and S. Shenoi, Eds., Boston, MA: Springer US, 2008, pp. 3–15, ISBN: 978-0-387-84927-0.

[107] Regjeringen.no, *Utlevering fra norge til utlandet*, https://www.regjeringen.no/no/tema/lov-og-rett/innsikt/internasjonalt-samarbeid-pa-justisomradet/utlevering-av-lovbrytere/utlevering-fra-norge-til-utlandet/id2339827/, [Online; accessed 26-June-2018], 2016.

[108] Regjeringen.no, *Varelister*, https://www.regjeringen.no/no/tema/utenrikssaker/Eksportkontroll/om-eksportkontroll/varelister/id2008485/, [Online; accessed 26-June-2018], 2018.

[109] Samsung, *SVE-2020-18632*. November 2020, SVE-2020-18632. Nov. 2020. [Online]. Available: https://security.samsungmobile.com/securityUpdate.smsb (visited on 20/01/2021).

[110] MITRE, *CVE-2020-28341*. Available from MITRE, CVE-ID CVE-2020-28341. Dec. 2020. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-28341 (visited on 20/01/2021).

[111] G. Alendal, *Chip chop - smashing the mobile phone secure chip for fun and digital forensics*, https://www.blackhat.com/us-21/briefings/schedule/#chip-chop---smashing-the-mobile-phone-secure-chip-for-fun-and-digital-forensics-23566, [Online; accessed 29-September-2021], Aug. 2021.

[112] Pwnie Awards LLC, *Pwnie award winners 2021*, https://pwnies.com/winners/, [Online; accessed 29-September-2021], Aug. 2021.

[113] G. Alendal, C. Kison and modg, 'got HW crypto? On the (in) security of a Self-Encrypting Drive series.,' *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 1002, 2015.

[114] Hardwear, *Talks from hardwear.io 2015*, https://hardwear.io/archives/the-hague-2015/.

[115] Forbes, *'no excuses' as western digital leaves gaping crypto flaws in hard drives*, http://www.forbes.com/sites/thomasbrewster/2015/10/20/western-digital-security-sucking-bad/.

[116] The Register, *Western digital's hard drive encryption is useless. totally useless*, http://www.theregister.co.uk/2015/10/20/western_digital_bad_hard_drive_encryption/.

[117] PCWorld, *Western digital encrypted external hard drives have flaws that can expose data*, http://www.pcworld.com/article/2995539/western-digital-self-encrypting-external-hard-disk-drives-have-flaws-that-can-expose-data.html.

[118] Motherboard, *Some popular 'self encrypting' hard drives have really bad encryption*, http://motherboard.vice.com/read/some-popular-self-encrypting-hard-drives-have-really-bad-encryption.

[119] Bugcrowd, *What is responsible disclosure?* https://www.bugcrowd.com/resource/what-is-responsible-disclosure/.

[120] Google, *How google handles security vulnerabilities*, https://www.google.com/about/appsecurity/.

Part II

Included Publications

Paper I

Forensics Acquisition — Analysis and Circumvention of Samsung Secure Boot enforced Common Criteria Mode

G. Alendal, G. O. Dyrkolbotn and S. Axelsson, 'Forensics acquisition - analysis and circumvention of samsung secure boot enforced common criteria mode,' *Digital Investigation*, vol. 24, S60–S67, 2018, ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2018.01.008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1742287618300409

Abstract

The acquisition of data from mobile phones have been a mainstay of criminal digital forensics for a number of years now. However, this forensic acquisition is getting more and more difficult with the increasing security level and complexity of mobile phones (and other embedded devices). In addition, it is often difficult or impossible to get access to design specifications, documentation and source code. As a result, the forensic acquisition methods are also increasing in complexity, requiring an ever deeper understanding of the underlying technology and its security mechanisms. Forensic acquisition techniques are turning to more offensive solutions to bypass security mechanisms, through security vulnerabilities.

Common Criteria mode is a security feature that increases the security level of Samsung devices, and thus make forensic acquisition more difficult for law enforcement.

With no access to design documents or source code, we have reverse

engineered how the Common Criteria mode is actually implemented and protected by Samsung's secure bootloader. We present how this security mode is enforced, security vulnerabilities therein, and how the discovered security vulnerabilities can be used to circumvent Common Criteria mode for further forensic acquisition.

I.1 Introduction

Digital forensics is the recovery and investigation of data found in digital devices [2]. Garfinkel [3] discusses the difficulties that awaits digital forensics, what challenges exist in today's tools, research and knowledge and how digital forensic research should move forward to keep digital forensics a valid method for the years to come. The prediction is that both the recovery, forensic acquisition, and investigation will become increasingly harder as complexity and security mechanisms, like encryption, grow in use. Faced with this ever increasing security of Commercial of-the-shelf (COTS) products, law enforcement faces an increasing challenge when it comes to the ability to do forensic acquisition. Where before law enforcement could bypass security mechanisms by e.g. accessing data at a lower level, like forensic de-soldering (chip-off), to read content off data storage directly, today's, often mandatory, encryption of user data on mobile devices invalidates such methods. The ability to read stored data on the device's storage is simply not enough. Reading encrypted data has little value without the corresponding encryption key(s). The addition of security features like device-tied encryption keys, supported by hardware and a TrustZone, gaining access to such encryption keys is made even harder. This might then require law enforcement to power on the device, in order to try to extract keys or decrypted data through interaction with the security mechanisms protecting the user data. This type of interaction often means installing or modifying code on the device. Even though law enforcement have legitimate cause for their "hacking", this is activity that in other contexts would be regarded malicious and illegal, also known as an attack. Therefore, to protect against such attacks, most mobile device vendors protect code running on the devices, from the first code executed at power on and all the way through to a full operating system, like Android, is up and running. This is often referred to as a *Secure Boot*, and refers to the trust in code executed on the device. This code should only be certified and official code, made by the vendor, and properly signed to prove authenticity.

Law enforcement always strives to acquire as much data as possible to support any ongoing investigation. So bypassing such complex security

schemes, if possible, forces law enforcement to invest in deeper knowledge and costly equipment to perform advanced forensic acquisition, utilising such attacks[1]. Law enforcement is then investing in the discovery and use of security vulnerabilities, to bypass security mechanisms to acquire digital evidence.

On the other hand, seen from a user and enterprise perspective, with the growing use of these devices, both end users and enterprises are demanding more secure devices to help protect sensitive data. The need to secure mobile devices, especially in an enterprise context is important, as devices moving in and out of the enterprises network, unchallenged, introduces attractive attack vectors for cyber criminals and cyber espionage.

Mobile Device Management (MDM) solutions can enable the centralised control of devices that are used in the enterprise. Enterprises can then monitor, control and administrate devices in a systematic manner, across device vendors and service providers. Samsung supports such solutions by offering a.o. a feature they refer to as *Common Criteria mode* or simply *CC mode* [4]. CC mode is a security feature designed to increase the device's protection against unauthorised access and can therefore pose an additional challenge to law enforcement trying to acquire data from devices with CC mode enabled. A major challenge is that CC mode denies access to the device firmware update mechanism, a common method used by law enforcement to gain access to data.

This paper presents the reverse engineering results of CC mode and how discovered security vulnerabilities can be used to circumvent CC mode for further forensic acquisition.

The rest of the paper is organised as follows: Section 'Related Work and Contributions' discusses related work and how our contribution relates. Section 'Samsung Secure Boot Model' introduces the Samsung secure boot model. Section 'Samsung CC mode and SBOOT' describes the CC mode related parts of the Samsung secure boot and how this relates to the secure execution environment, TrustZone. Section 'Unauthorised disabling of CC mode' discusses attacks on the CC mode. In section 'Conclusion' we discuss the implications of our findings and offer our conclusions.

I.2 Related Work and Contributions

Recovering data from mobile devices can be achieved by reading data from storage or from volatile memory (RAM). The two sources of data differs in both how data is stored and how data can be retrieved. Data in long term

[1]With the word "attacks", in the context of this article, we mean: exploiting vulnerabilities for forensic data acquisition purposes by law enforcement agencies.

storage is often stored well structured in file systems, as it has to be able to be read by different operating systems, and other tools. Data structures in RAM are often less well documented, and the formats more volatile, as it needs only survive to the next restart of the device. RAM is repopulated each time the device is restarted.

Nathan Scrivens et al. [5] summarised many of the current options for forensic acquisition of storage on Android mobile devices. According to Scrivens et al., the main options are chip-off, de-soldering storage for off-device reading, JTAG (Joint Test Action Group) interface for in-circuit reading of storage, rooting and exploitation solutions for recovering data by breaking the security of the device, Android Debug Bridge (ADB) by utilising device debug capabilities for forensic acquisition, and finally backup solutions retrieving data through normal or rooted user access. These different methods have different requirements and weaknesses. Chip-off requires physical access to underlying storage media, and can not deal with the increasing use of encryption on storage devices. JTAG is a interface often used during development and testing of a device, and can be used to communicate directly with the underlying storage media. However, the JTAG test pins can be hard to find and access on different devices, and can also be secured against unauthorised access, and also disabled by the vendor before shipping. ADB is a powerful debug interface supported by Android, but it is not enabled by default on most Android devices, nor does it give root access. Finally, backup applications are rarely accessible to unauthenticated users and are often of limited use for forensics.

Seung Jei Yang et al. [6] demonstrated a different approach: doing forensic acquisition of storage media through the misuse of the device firmware update protocols. This will give access to the underlying storage and the ability to dump its content. Unfortunately this method will also be insufficient if the data stored is encrypted.

Seung Jei Yang et al. [7] recently demonstrated a different use for the device firmware update protocols. Instead of acquiring storage they have demonstrated how to acquire RAM through this update protocol. This can again be used to acquire encryption keys used to encrypt storage, in addition to save user data that resides in RAM at the time of RAM acquisition.

M. Guido et al. [8] demonstrated *hawkeye*, an agent to do rapid acquisition of Android devices. Although their goal is to reduce the amount of data needed to be transferred during the acquisition process, this is an example of a forensic agent that needs to be injected into the device to function as expected. This is done by installing a custom boot image on the device to facilitate hawkeye injection. Installing this custom image is done through the device firmware update protocol and access to firmware update mechanism is a requirement.

As we can see, access to a device's firmware update protocol can be vital for successful forensic acquisition. Any functionality denying this access is therefore limiting the possibilities for law enforcement to acquire data from a given device. CC mode is preventing law enforcement access to the firmware update mode on Samsung devices. Our contribution is to analyse and circumvent CC mode to gain access to the firmware update mode. For completeness, we have also included the discussion of a MDM setting, also affecting access to the firmware update mode.

Our reverse engineering of CC mode reveals security vulnerabilities in the design and implementation of these security mechanisms, and demonstrates how such security vulnerabilities can be discovered and used in digital forensic acquisitions.

Our contribution shows that law enforcement trying to acquire data from a device can disable CC mode and get access to firmware update mode, thus removing the extra layer of security enforced by CC mode. Disabling CC mode can then enable existing methods but also increases the attack surface in general, increasing the possibility to discover new vulnerabilities and methods.

I.3 CC Mode and Methodology

CC mode is built on top of the phone's Android security model and hardware, to increase enterprise security. Samsung has made available several guidance documents for Common Criteria evaluation for many of their different phone models [9].

Samsung provides a wide range of management APIs to control a Samsung device [10]. These APIs can be used in 3rd party MDM solutions. To further promote the use of CC mode in MDM solutions, Samsung has made available a Common Criteria mode APK [4]. This Android application package (APK) is installed on the evaluated device and sets a number of default policies and security settings. This APK is intended for evaluators and IT admins, to test the features of Samsung's CC mode. Samsung provides a long list of compatible phones, e.g. the Samsung Galaxy S6, with model name `SM-G920F` and the Samsung Galaxy S7 Edge, with model name `SM-G935F`. It is unknown to the authors what requirements are needed for a particular model to be compatible, but for blocking the access to the firmware update mode, the bootloader of compatible models must have code to handle this blocking. It is the bootloader that implements the firmware update mode.

The policy and settings set by the CC mode APK is the basis for the testing done in this paper. When we refer to CC mode, we refer to settings

set by the CC mode APK. Our main test device was a Samsung Galaxy S7 Edge (SM-G935F) running firmware version `G935FXXS1DQGA_G935FNEE1DQF3_NEE`[2].

One crucial feature of CC mode is the ability to only allow Firmware Over the Air (FOTA) firmware updates. This is to protect against an attacker with physical access to the device, trying to install unauthorised firmware on the device. Updating firmware in this way on Samsung devices is done through what is called *ODIN* mode. CC mode will both block ODIN mode and any attempt to boot an unofficial boot image already stored on the device.

Other features of Samsung CC mode and Common Criteria in general will not be discussed further as these features does not influence the blocking of ODIN mode.

Samsung devices come in different hardware configurations, where system-on-a-chip (SoC) implementations from Qualcomm (e.g. Snapdragon) and Samsung (Exynos) are the most common. Although the phone models share the same name, like *Samsung Galaxy S7*, they are very different in e.g. hardware components and bootloader code. In this paper we only focus on Samsung devices based on the Exynos SoC variants. Examples of devices with Exynos SoCs are Samsung Galaxy S6 (models `SM-G920F / SM-G925F`) and Samsung Galaxy S7 (models `SM-G930F / SM-G935F`).

Access to ODIN mode is enforced by the Samsung bootloader. The bootloader is part of the secure start-up of the device and is native code responsible for starting the device. On the studied models with the Exynos SoC, the bootloader responsible for ODIN mode is often referred to as *SBOOT*. SBOOT is built from Samsung proprietary code, and documentation and source code are not publicly available. We have analysed how the secure bootloader knows that CC mode is enabled and how this is used to limit access to certain features. We have also analysed the security of the storage of this CC mode configuration, as well as how SBOOT can change the configuration or simply disable CC mode. This leaves SBOOT not only responsible for enforcing the configuration, but also changing the setting.

Both design and implementation details of many security features are generally not available, and hence many such features may be left unexplored by the research community. To be able to analyse the enforcement of CC mode and how ODIN mode is blocked, we reverse engineered SBOOT with both static and dynamic analysis techniques. With access to the firmware for our test device, we reverse engineered the binary SBOOT code. Most of the static reverse engineering effort was done using the tool *IDA*

[2] `G935FXXS1DQGA_G935FNEE1DQF3_NEE.zip`
`SHA-1:67CA63BCAF53C9D48A5D5DF43A8F5E56544081AC`

Pro from Hex-Rays [11]. We also developed our own exploit based on the SBOOT vulnerability disclosed in a security blog by Nitay Artenstein [12]. We used this exploit as a tool to perform more dynamic analysis of how CC mode is enforced and used to protect against unauthorised firmware updates. Our exploit is fully developed using the Python scripting language, with the aid of the Keystone assembler framework [13] for creating binary ARM code to be executed as part of the exploitation. Our goal was to be able to evade or disable CC mode, to get access to ODIN mode.

I.4 Samsung Secure Boot Model

The code that is implementing ODIN mode, and thereby flashing firmware on Samsung devices, is located in the bootloader of Samsung devices. Therefore the bootloader must be able to turn off access to ODIN mode when CC mode is enabled on the device, in order for CC mode to disable firmware updates, e.g. designed to sidestep centralised control. To better understand how this mechanism works, some background on the Samsung secure boot model, applicable for devices using the Exynos SoC, follows.

To maintain security and trust in code running on devices, Samsung utilises a secure boot model [14], where all code running from power on until a complete Android system is running, is signed. This includes the integrity of the TrustZone and the baseband processor, that handles most of the radio functions. The security of bootloaders is therefore crucial for the integrity of the device. Nilo Redini et al. [15] explored vulnerabilities in both design and implementation of bootloaders for a range of devices, and emphasised on the importance of a secure boot by demonstrating several attacks. A simplified boot model used by Samsung Exynos devices is show in Figure I.1. This shows how execution is started at the BootROM and carries on through the boot process through to the Android kernel.

Samsung provides a generic description of their platform security [14]. This describes that the signature chain is rooted in the *Samsung Secure Boot Key*, SSBK, used to sign Samsung approved executable boot components. The public part of this key is stored in the phone's hardware at manufacture and will not change during the device lifetime. This is used by the BootROM when the device powers on. As seen in Figure I.1, the BootROM makes sure all executable code fetched from storage during boot is signed by Samsung. Booting a device with this model starts with the primary bootloader, loaded from Read-Only Memory (ROM). This primary bootloader loads the next bootloaders, Boot loader 1 (BL1) and Boot Loader 2 (BL2) from storage, e.g. flash, to RAM, checks the signature and advances execution to BL1. BL1 will carry out its tasks, often related

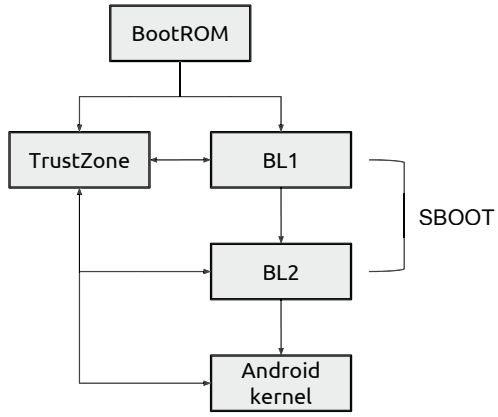


Figure I.1: Overview of the Samsung Secure Boot model from BootROM to an Android kernel.

to hardware initialisation, and advance execution to BL2. BL2 is a more complex bootloader, with larger code base, which in turn loads and checks the signature of the Android kernel before advancing execution to it. As a final stage, the Android kernel boots and loads the full operating system which enables all the device's features.

If all these bootloader stages maintain the signed integrity from the SSBK key, Samsung refers to this as *secure boot*. Note that Samsung distinguishes this from *trusted boot*, which also includes *Rollback Protection (RP)*, preventing the "downgrading" of any executable code to official, but vulnerable, older versions of the bootloaders or Android OS.

Note that Samsung does not forbid installing unofficial Android kernels, not signed by Samsung. This is however considered *tampering* with the device, and consequently a one-time programmable tamper fuse (eFuse) will be set. This fuse cannot be unset and the device is from here on marked as "Warranty void". This fuse is often referred to as the *KNOX Warranty Bit* [16] and is a Samsung proprietary way of marking the device as having been tampered with. Samsung can prevent the installation of such unofficial Android kernels if *Factory Reset Protection (FRP)* is set on the device. FRP is a setting that is enabled e.g. if the user adds a Google account to the system. FRP will prevent the installation of unofficial firmware updates, but will not deny access to ODIN mode.

In this paper we will refer to BL1 and BL2 as *SBOOT*. SBOOT is the first code loaded from writable storage and can therefore be upgraded as part of what is often referred to as *firmware upgrades*. Firmware upgrades for Samsung devices are often big archives that can upgrade different parts

of the Samsung code environment, like the Android OS, the baseband processor or the SBOOT bootloaders. Any upgrade to the SBOOT will be included in a file called *sboot.bin*, so in order to analyse executable code belonging to SBOOT, we need to analyse the *sboot.bin* file. As seen from Figure I.1, SBOOT is a crucial part of the Samsung Secure Boot model.

Kanonov et al. [16] have analysed and found weaknesses in the security of the Samsung KNOX secure containers, and also described the Secure Boot process, related to the security of KNOX containers. They also discuss other important security features, like runtime protections, named TIMA, and e.g. it's use in attestation of a device. Device attestation is to test the authenticity and integrity of the security measures and policies.

SBOOT is responsible for a range of tasks before it loads and executes the Android kernel. These tasks are part of the Secure/Trusted boot and includes loading
TrustZone dedicated applications, also known as *trustlets*. A trustlet is a small and dedicated application created to solve a specific, often sensitive task, like digital rights management (DRM). Trustlets run in the TrustZone.

The TrustZone is a separate execution environment, supported by the hardware, that divides each processor core into two separate "worlds". Often we refer to these different execution environments as the *normal* world and the *secure* world; the TrustZone. SBOOT is not part of the TrustZone and is running in *normal* world. When SBOOT is done executing its needed boot routines, it will load and execute the Android kernel. The Android kernel also runs in the same *normal* world. The TrustZone does not influence the enforcing of CC mode during boot and are therefore left out of further discussions in this paper.

Looking into the different steps performed by SBOOT, we can analyse the interaction with the CC mode configuration and how this is enforced. This will be explained in the next section.

I.5 Samsung CC mode and SBOOT

The Samsung Common Criteria Administrator Guidance, section 4.3.2.2 [17], states that to place the device in the evaluated configuration, *CC mode* must be enabled on the phone. This mode will, once enabled, enforce FIPS-validated crypto, disable USB connectivity in recovery mode and only allow Firmware Over the Air (FOTA) updates to the system. The Samsung Common Criteria mode APK [4] will enable CC mode on any supported model, for testing purposes. We will only focus on the parts of CC mode that affects SBOOT. The CC mode setting that affects SBOOT is either *on* or *off*.

After installing the APK and enabling CC mode, we can start to investigate how this affects SBOOT and how SBOOT enforces the blocking of ODIN mode. There are two ways of updating the firmware on our test devices; over-the-air through FOTA or with physical access through ODIN. Blocking ODIN mode is a crucial part of CC mode, since we'll see later that if we are given access to ODIN mode we can simply disable CC mode altogether. It is expected to be more difficult for an attacker to install unauthorised firmware updates through FOTA, as this is an online feature with secure communication to Samsung firmware servers.

With a combination of static reverse engineering of SBOOT and dynamic reverse engineering using an SBOOT exploit, we have analysed how SBOOT is affected by enabling CC mode, how this setting is stored, and how to attack it to disable CC mode. Our analysis shows that the CC mode setting is stored in flash, on a data partition called *PARAM*.

I.5.1 The PARAM Partition

An Android device's storage is divided into several logical partitions, where *system* and *userdata* are the most important ones. The first contains the Android operating system files (OS) and the latter contains most of the user data. Another partition, the *PARAM* partition, is a rather small logical partition that contains a few JPG pictures used by SBOOT, e.g. the Samsung Galaxy boot logo displayed when the device is powered on. In addition, there is a file, *adv-env.img* (See Table I.1), that a.o. contains parameters submitted to the Android kernel when SBOOT passes the execution to the Android kernel after SBOOT has loaded and checked the signature of the Android kernel. The PARAM partition is upgraded through firmware updates, with updates in the file *param.bin*, which is part of the firmware archives.

PARAM is however also storage for some other important settings. Our analysis show that in the last few 512-byte blocks of the PARAM partition, Samsung stores important settings like CC mode, MDM settings, the current system status ("Samsung Official" or "Custom") and flags named AFW and UCS. The way SBOOT address these settings, is to count backwards in number of 512-blocks from the end of the PARAM partition. See Figure I.2.

To access the CC mode setting on our primary test device, SBOOT references the block
`PARTITIONSIZEinBLOCKs(PARAM)-4` in the PARAM partition. This is a static offset in the SBOOT code, but can change with different versions of the SBOOT binary. The CC mode configuration is 64 bytes of encrypted data stored at the start of the referenced block. These 64 bytes are read

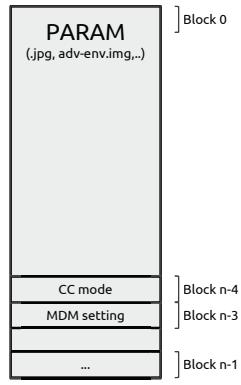


Figure I.2: PARAM Partition

and decrypted with a function we call `S_CC_decrypt`[3]. As the function `S_CC_decrypt` only takes two arguments; the input buffer with the 64 encrypted CC mode bytes, and a zero-initialised output buffer to receive the decrypted content, we assume the key material needed for the decryption is contained in, or retrieved by, the function itself. Looking closer at the `S_CC_decrypt`, we can see it uses a *whitebox* AES Cipher, where the AES key is not exposed during encryption or decryption [18]. This means that the function `S_CC_decrypt` can decrypt the CC mode data without exposing the key in static code or dynamic runtime analysis. The function is a decryption oracle. SBOOT also contains the corresponding `S_CC_encrypt`, though our analysis does not find this function to be called by the SBOOT binary. As it turns out, a native Android library, `/system/lib64/libSecurityManagerNative.so` matches the two WAES encrypt/decrypt oracles, discovered also by André Moulu [19]. Since SBOOT does not seem to call `S_CC_encrypt`, this leads us to think that the CC mode configuration is only written by the Android environment and not by the SBOOT bootloader. SBOOT simply queries the configuration. Given these WAES oracles, we can freely read and write the CC config from both the SBOOT and the Android environment, if we control execution. This will also be the case if we have other means of write access to the PARAM partition.

The decrypted CC mode data contains the magic characters `timg` in bytes `0-3` and the characters `NOCC` (`CCON` read little-endian) or `FFOC` (`COFF` read little-endian) in bytes `4-7`, signalling CC mode *on* or *off* respectively.

[3] All functions named by the authors are prefixed with `S_`. Function names comes from educated guesses made from error message strings referenced inside functions.

The CC mode setting is read early in SBOOT's execution and a global flag variable is set to signal the CC mode configuration. This flag can be queried through a function we call `S_CC_MODE_isSet`, which returns *true* if CC mode is enabled.

Another setting assumed to be important for MDM managed devices, is the `MDM setting`. This setting is not set by the Samsung CC mode APK. However, we assume that MDM solutions use this setting actively, and we therefore include this setting and its effect on CC mode in our analysis. The MDM setting is stored in block
`PARTITIONSIZEinBLOCKs(PARAM)-3` in the PARAM partition. It is an unencrypted setting in the first 32 bytes in the corresponding block. These bytes are read in during boot, as with CC mode, and sets a global variable corresponding to the MDM setting in PARAM. The MDM global variable can have three different values, where `1` and `2` seems to mean that MDM is in use.

The following pseudo code describes the different byte values in the MDM block and the corresponding MDM setting:

The MDM setting is set to `1` if

```
block[30] == 2 && block[31] == 6 &&
block[3] == 8 && block[7] == 8.
```

The MDM setting is set to `2` if

```
block[30] == 2 && block[31] == 6 &&
block[3] != 8 && block[7] == 8.
```

The MDM setting is set to `3` if

```
block[30] == 2 && block[31] == 6 &&
block[3] == 8 && block[7] != 8.
```

So if the global MDM variable in SBOOT is set to `1` or `2`, `MDM mode` is considered enabled and this will also affect how SBOOT permits access to ODIN mode.

I.5.2 SBOOT enforcing CC mode

`S_CC_MODE_isSet` is called at three different locations in SBOOT; when the phone is trying to enter ODIN mode, when the phone tries to boot a kernel with no or invalid signature, and when the SBOOT sends status variables to the TrustZone. It's the first of these three that is crucial for denying access to ODIN mode.

The decision of *when* during the boot process to check for CC mode is crucial for the success of denying access to ODIN mode. As the bootloader is responsible for initialising the device as well as firmware up-

dates to the system, it must also check for potential errors. These checks for errors are intermingled with security related checks, such as checking for CC mode. This intermingling makes the security checks vulnerable to changes in the execution path caused by errors, as specific errors can make the execution path change, such that certain security checks never takes place. Our analysis shows that this situation can arise for CC mode. Figure I.3 shows the pseudo code for the function responsible for enabling ODIN mode, `S_boot_enter_download_mode`. The function `S_boot_enter_download_mode` is called from various locations in the SBOOT boot process. It will check if it should go to ODIN mode. If so, it will call a function, `S_USB_mode_enter`, which will apply the configuration to the device and switch to ODIN mode. `S_USB_mode_enter` can be called with one parameter, where `0` means *ODIN/download mode*.

`S_boot_enter_download_mode` itself receives one parameter, *reason* (Fig. I.3, line 1), which indicates the reason for entering ODIN mode. The function acts accordingly based on the value of this parameter. The pseudo code shows that `S_USB_mode_enter` is called from two different locations. The call in line `31` is only reached if the call to the function `S_CC_MODE_isSet` in line `25` returns `false`. If `S_CC_MODE_isSet` returns `true`, SBOOT will print "DOWNLOAD IS BLOCKED BY CC MODE" to the device screen and eventually power off the device. The other call to `S_USB_mode_enter` is found at line `14`, and this call is not preceded by a call to `S_CC_MODE_isSet`, meaning this call seems to ignore if any CC mode is set. So if `S_boot_enter_download_mode` is called with parameter `6`, the device will enter ODIN mode, even if CC mode is set. Backtracking callers to `S_boot_enter_download_mode` identifies the situation in where this happens.

SBOOT has a table of environment variables, stored in the *adv-env.img* file of the PARAM partition. These values are listed in Table I.1. These values are ways of influencing the execution of SBOOT and since they are stored in PARAM, they survive a device reboot. As already mentioned, some of these are also passed on as parameters to the Android kernel, but the discussion of these are outside the scope of this article.

One example is the `REBOOT_MODE`, signalling SBOOT which boot mode to use, where *normal boot* `(0)`, *ODIN/download mode* `(1)`, *upload mode* `(2)` and *recovery mode* `(4)` are example settings. See Table I.2 for a full listing of values for `REBOOT_MODE`.

Many of these environment variables are checked during boot, and one of them is interesting with respect to CC mode; `DN_ERROR`. During normal boot, SBOOT calls a rather complex function, we call `S_boot_set_boot_mode`, that decides which boot mode to choose for the device. This is based on numerous checks on hardware, battery state, environment variables and so on. During these tests there's a call to a function, `S_s5p`

Table I.1: SBOOT environment variables, stored in *adv-env.img*

Index	Name	Example setting
0	REBOOT_MODE	0
1	SWITCH_SEL	3
2	DEBUG_LEVEL	18505
3	SUD_MODE	0
4	DN_ERROR	0
5	CHECKSUM	3
6	ODIN_DOWNLOAD	1
7	SALES_CODE	0
8	SECURITY_MODE	1526595585
9	NORMAL_BOOT	0
10	CP_DEBUG_LEVEL	22015
11	USERBOOT_MODE	0
12	DIAG_MODE	0
13	CHARGING_MODE	48
14	INT_RSVD14	0
15	LCD_RES	1
16	CMDLINE	console=ram loglevel=4
17	BARCODE_INFO	(null)
18	KEEP_LOG	(null)

```
S_boot_enter_download_mode(int reason){
  if (reason == 1) {
    S_draw_image("warning_L.jpg");
      if (S_user_cancel()){
        S_reboot_device();
      }
      else {
        S_draw_image("download_L.jpg");
      }
  }
  else {
    if (reason == 6) {
      S_draw_image("download_error_L.jpg");
      S_USB_mode_enter(0);
    }
    S_draw_image("download_L.jpg");
    if (reason == 3) {
      // SUD mode ..
    }
    if (reason != 4) {
      goto error();
    }
  }
  //...
  if (S_CC_MODE_isSet()) {
    S_screen_print(
        "DOWNLOAD IS BLOCKED BY CC MODE");
    S_sleep(1000);
    S_power_off_device();
  }
  S_USB_mode_enter(0);
}
```

Figure I.3: Simplified pseudo code of `S_boot_enter_download_mode`

`_check_download`, which will return a value different from `0` if it should go to ODIN/download mode and returns `0` if it should *not* go to download mode. Taking the path in which `S_boot_set_boot_mode` returns `0`, *no* ODIN/download mode, takes us to a call to a function we call `S_env_get`, called with the integer value of `DN_ERROR` from Table I.1. This call returns the integer value of the environment variable `DN_ERROR`. If this is set, there is a call to the already discussed function `S_boot_enter_download_mode`, with the parameter value of `6`. As we have already analysed `S_boot_enter_download_mode` and located a bypass of the CC mode check if the input parameter is equal to `6`, we now have a way to bypass the CC mode check. We can simply set the environment value `DN_ERROR` to a

non-zero value and reboot the device. The device will enter ODIN mode, even if CC mode is enabled. This seems to us like an *emergency* ODIN mode, where the bootloader is requiring a firmware update as the result of a failed firmware update. Figure I.4 shows the screen shown on a Galaxy S7 Edge (`SM-G935F`) when in this mode.

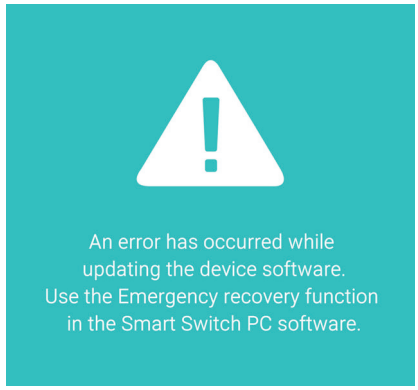


Figure I.4: *emergency* ODIN/download mode

Table I.2: REBOOT_MODE variable values

Name	value
REBOOT_MODE_NONE	0
REBOOT_MODE_DOWNLOAD	1
REBOOT_MODE_UPLOAD	2
REBOOT_MODE_CHARGING	3
REBOOT_MODE_RECOVERY	4
REBOOT_MODE_FOTA	5
REBOOT_MODE_FOTA_BL	6
REBOOT_MODE_SECURE	7
REBOOT_MODE_FWUP	9
REBOOT_MODE_EM_FUSE	10

I.5.3 MDM mode

We include a discussion on the MDM mode for completeness as this setting is expected to be used by some MDM solutions that supports Samsung Exynos devices. MDM mode also affects how SBOOT prevents ac-

cess to ODIN mode. SBOOT access this setting through a function we call `S_MDM_MODE_isSet`. This function will return *true* (`1`) if the global MDM variable is either `1` *or* `2`. It does not seem to make a distinction between the two. SBOOT calls this function from three different functions; when *in* ODIN mode, when booting, and for providing the MDM setting to the TrustZone. In ODIN mode SBOOT is checking for MDM mode and will prevent firmware upgrade to any partition other than the SBOOT ("BOOTLOADER") itself, if MDM mode is set. So we can see that while CC mode prevents access to ODIN mode itself, MDM mode will be checked *in* ODIN mode as well.

SBOOT also checks for MDM mode during boot. These checks are done *before* the CC mode check done in `S_boot_enter_download_mode`, already discussed in Section 'Samsung CC mode and SBOOT'. Checks are done inside the function called `s5p_check_download`, where the SBOOT decides *if* it should call `S_boot_enter_download_mode`. Note that the call to `s5p_check_download` is done *before* the check for the environment variable `DN_ERROR` is done, which is only checked if `s5p_check_download` returns a non-zero value, meaning *no* ODIN mode. We have already seen in Section 'Samsung CC mode and SBOOT' that setting the environment variable `DN_ERROR` to a non-zero value will put the device in an emergency ODIN mode. This will then be the same situation even if MDM mode is set. One important difference is that ODIN mode with MDM mode set will only allow firmware updates of the SBOOT partition.

As MDM mode is an unencrypted setting stored in the PARAM partition, any erasing or overwriting of the MDM setting block will reset and disable the MDM setting on the device, and therefore not prevent access to ODIN mode anymore.

I.6 Unauthorised disabling of CC mode

Based on our analysis of SBOOT, and the way the bootloader enforces CC mode to prevent ODIN/download mode, we have found three different attacks to disable CC mode[4]. All attacks have been verified through successful tests on our test device. We will also discuss the effect of an optional MDM setting, although this was not enabled by the Samsung CC mode APK.

[4]We have not considered if or how any commercial forensic tools support bypassing CC mode.

I.6.1 Modifying the PARAM partition

As discussed in section 'The PARAM Partition', we can modify the CC mode setting if we have *write* access to the PARAM partition. This can be either by physical access to the underlying flash storage, or through ODIN/download mode. Given PARAM write access, we can simply change the CC mode setting and encrypt the setting with the WAES encryption oracles from either SBOOT, or `/system/lib64/libSecurityManagerNative.so`. We can also simply overwrite the CC mode data bytes with zero bytes, with the same effect. So simply flashing a stock `param.bin` from a corresponding firmware upgrade archive will disable CC mode.

If MDM mode is also present, flashing a stock PARAM through ODIN mode is denied. This is not the case with physical access to the underlying flash storage, as overwriting the MDM setting block will disable MDM mode.

I.6.2 SBOOT exploitation

Since the bootloader is responsible for reading and enforcing the CC mode setting in the PARAM partition, any attack on the execution flow of SBOOT will have the potential to bypass CC mode and enable ODIN/-download mode. Based on a vulnerability discovered by Nitay Artenstein [12], we have developed a fully functional exploit to make SBOOT ignore the CC mode settings. One way of doing this is to patch the code flow of SBOOT to call `S_boot_enter_download_mode` with the parameter `6` and then overwrite the PARAM partition in the same way as detailed in section 'Modifying the PARAM partition'. Another way could be to patch the `S_CC_MODE_isSet` function to always return *false* by either patching the return code to `0` or by changing the global variable it references to `0`. This way we can bypass the blocking of booting unofficial and unsigned kernels, and the CC mode enabled setting is not reported to TrustZone before booting the kernel, in effect disabling CC mode on the booted Android system.

If MDM mode is also activated, this can also be bypassed simply by setting the global SBOOT MDM mode setting to 0, resulting in the function `S_MDM_MODE_isSet` always returning *false*. This is expected, as any arbitrary changes to the SBOOT code and execution flow will leave all security checks done by SBOOT ineffective.

I.6.3 Setting `DN_ERROR`

We have seen the effect of setting the `DN_ERROR` environment variable to a *non-zero* value in section 'SBOOT enforcing CC mode'. This has been tested through a console interface provided by SBOOT. This console can

be reached with the aid of a custom USB connector and a simple RS232-to-USB serial converter [12]. After entering the SBOOT console, one can simply type `setenv DN_ERROR 2` followed by a `saveenv` and `reset`. This will try to reboot the device with a normal boot, but the non-zero `DN_ERROR` environment variable will force the device into an emergency ODIN/download mode. From here we can modify the PARAM partition like in section 'Modifying the PARAM partition'.

If MDM mode is also active, the device will still enter ODIN mode. However, since the MDM mode is also checked by ODIN mode when flashing firmware, only
changes to the bootloader, SBOOT, are allowed. This will prevent this attack.

I.7 Conclusion

In this paper we have successfully demonstrated how to disable Common Criteria (CC) mode on selected Samsung devices. The effect of disabling the CC mode increases the device's attack surface and can further be used in forensic acquisition. This will open up the device for misuse of the firmware update protocol for direct storage or RAM acquisition, in addition to both signed and unsigned firmware updates through ODIN/download mode, depending on the *Factory Reset Protection* and *Rollback Protection* settings on the device. If one uses the SBOOT exploit attack from section 'SBOOT exploitation' we can easily avoid both of these defences as well. This is because these security settings are also enforced by the SBOOT bootloader and can therefore easily be changed/disabled.

We have found and tested the effect of several weaknesses in the enforcing of CC mode on our tested device. Using exploits to attack SBOOT will break the chain-of-trust anchored in the boot process. This will break the trust in all code running in the same *normal* world on the application processor on this device. With such a powerful attack we can replace or adapt to most of the security features of SBOOT. This is not unexpected, but emphasises the need for a secure bootloader and chain-of-trust.

As future work we suggest testing these attacks on actual MDM solutions utilising the Samsung CC mode feature and/or the MDM setting. The effect on a MDM solution after disabling CC mode by changing or erasing the CC mode setting and/or the MDM setting in the PARAM partition has not been tested and could lead to other attack scenarios of forensic interest.

Acknowledgements

The research leading to these results has received funding from the Research Council of Norway programme IKTPLUSS, under the R&D project Ars Forensica grant agreement 248094/O70.

I.8 References

[1] G. Alendal, G. O. Dyrkolbotn and S. Axelsson, 'Forensics acquisition - analysis and circumvention of samsung secure boot enforced common criteria mode,' *Digital Investigation*, vol. 24, S60–S67, 2018, ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2018.01.008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1742287618300409.

[2] B. Carrier, 'Defining digital forensic examination and analysis tools,' *International Journal of Digital Evidence*, vol. 1, p. 2003, 2002.

[3] S. L. Garfinkel, 'Digital forensics research: The next 10 years,' *Digit. Investig.*, vol. 7, S64–S73, Aug. 2010, ISSN: 1742-2876. DOI: 10.1016/j.diin.2010.05.009. [Online]. Available: http://dx.doi.org/10.1016/j.diin.2010.05.009.

[4] Samsung, *Common criteria mode apk*, https://www.samsungknox.com/en/article/common-criteria-mode, [Online; accessed 28-July-2017], 2017.

[5] N. Scrivens and X. Lin, 'Android digital forensics: Data, extraction and analysis,' in *Proceedings of the ACM Turing 50th Celebration Conference - China*, ser. ACM TUR-C '17, Shanghai, China: ACM, 2017, 26:1–26:10, ISBN: 978-1-4503-4873-7. DOI: 10.1145/3063955.3063981. [Online]. Available: http://doi.acm.org/10.1145/3063955.3063981.

[6] S. J. Yang, J. H. Choi, K. B. Kim and T. Chang, 'New acquisition method based on firmware update protocols for android smartphones,' *Digital Investigation*, vol. 14, no. Supplement 1, S68–S76, 2015, The Proceedings of the Fifteenth Annual DFRWS Conference, ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2015.05.008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1742287615000535.

[7] S. J. Yang, J. H. Choi, K. B. Kim, R. Bhatia, B. Saltaformaggio and D. Xu, 'Live acquisition of main memory data from android smartphones and smartwatches,' *Digital Investigation*, 2017, ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2017.09.003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1742287617301317.

[8] M. Guido, J. Buttner and J. Grover, 'Rapid differential forensic imaging of mobile devices,' *Digital Investigation*, vol. 18, no. Supplement, S46–S54, 2016, ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2016.04.012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1742287616300457.

[9] Samsung, *Guidance documents for common criteria evaluation*, https://support.samsungknox.com/hc/en-us/articles/115015114407, [Online; accessed 05-Jan-2018], 2017.

[10] Samsung, *Developer tools overview*, https://seap.samsung.com/sdk, [Online; accessed 27-July-2017], 2017.

[11] Ilfak Guilfanov, *Ida - disassembler & decompiler*, https://hex-rays.com/, [Online; accessed 27-July-2017], 2017.

[12] Nitay Artenstein, *Exploiting android s-boot: Getting arbitrary code exec in the samsung bootloader (1/2)*, http://hexdetective.blogspot.no/2017/02/exploiting-android-s-boot-getting.html, [Online; accessed 27-July-2017], 2017.

[13] Keystone team, *Keystone - the ultimate assembler*, http://www.keystone-engine.org/, [Online; accessed 28-September-2017], 2017.

[14] Samsung, *Platform security*, http://developer.samsung.com/tech-insights/knox/platform-security, [Online; accessed 28-July-2017], 2017.

[15] N. Redini, A. Machiry, D. Das, Y. Fratantonio, A. Bianchi, E. Gustafson, Y. Shoshitaishvili, C. Kruegel and G. Vigna, 'Bootstomp: On the security of bootloaders in mobile devices,' in *26th USENIX Security Symposium (USENIX Security 17)*, Vancouver, BC: USENIX Association, 2017, pp. 781–798, ISBN: 978-1-931971-40-9. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/redini.

[16] U. Kanonov and A. Wool, 'Secure containers in android: The samsung knox case study,' in *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*, ser. SPSM '16, Vienna, Austria: ACM, 2016, pp. 3–12, ISBN: 978-1-4503-4564-4. DOI: 10.1145/2994459.2994470. [Online]. Available: http://doi.acm.org/10.1145/2994459.2994470.

[17] Samsung, *Samsung android 7 on galaxy devices guidance documentation*, https://docs.samsungknox.com/CCMode/Adminguidev3_0.pdf, [Online; accessed 07-September-2017], Jun. 2017.

[18] S. Chow, P. A. Eisen, H. Johnson and P. C. v. Oorschot, 'White-box cryptography and an aes implementation,' in *Revised Papers from the 9th Annual International Workshop on Selected Areas in Cryptography*, ser. SAC '02, London, UK, UK: Springer-Verlag, 2003, pp. 250–270, ISBN: 3-540-

00622-2. [Online]. Available: http://dl.acm.org/citation.cfm?id=646558.694920.

[19] André Moulu, *How to lock the samsung download mode using an undocumented feature of aboot*, https://ge0n0sis.github.io/posts/2016/05/how-to-lock-the-samsung-\download-mode-using-an-undocumented-feature-of-aboot/, [Online; accessed 30-July-2017], May 2016.

Paper II

Exploiting Vendor-Defined Messages in the USB Power Delivery Protocol

G. Alendal, S. Axelsson and G. O. Dyrkolbotn, 'Exploiting Vendor-Defined Messages in the USB Power Delivery Protocol,' in *Advances in Digital Forensics XV*, G. Peterson and S. Shenoi, Eds., Springer International Publishing, 2019, pp. 101–118, ISBN: 978-3-030-28752-8

Abstract

The USB Power Delivery protocol enables USB-connected devices to negotiate power delivery and exchange data over a single connection such as a USB Type-C cable. The protocol incorporates standard commands; however, it also enables vendors to add non-standard commands called vendor-defined messages. These messages are similar to the vendor-specific commands in the SCSI protocol, which enable vendors to specify undocumented commands to implement functionality that meets their needs. Such commands can be employed to enable firmware updates, memory dumps and even backdoors.

This chapter analyzes vendor-defined message support in devices that employ the USB Power Delivery protocol, the ultimate goal being to identify messages that could be leveraged in digital forensic investigations to acquire data stored in the devices.

II.1 Introduction

An important goal of mobile device forensics is to acquire data. Mobile phones typically have two key data sources: (i) volatile memory (RAM); and (ii) long-term storage (typically, flash memory). These two sources differ in content and acquisition methods. RAM is often proprietary, short-term storage that is not intended for interpretation by applications other than the one that stored the data. In contrast, long-term storage such as flash memory contains well-structured data, usually in a filesystem, that is meant to be re-read, often by the operating system. Nevertheless, both types of storage maintain data that is important in digital forensic investigations.

Security mechanisms in commercial products are hindering the forensic acquisition of data. Data encryption in flash memory has invalidated methods such as desoldering (i.e., chip-off) that enable data to be read directly from a chip. Encryption prevents the extracted data from being interpreted without the decryption keys. The keys are often protected by additional encryption keys that tie the data to the specific device that encrypted the data in long-term storage. Therefore, transplanting a flash memory chip to a different, but identical, device would not decrypt the stored data. Device-tied encryption keys are also protected by security features such as TrustZone that rely on tamper-proof hardware. Therefore, in order to access data from a secured device, it is necessary to exploit security vulnerabilities in the device itself, or leverage undocumented features such as backdoors or indirectly increase the attack surface of the device.

The general approach is that any data extraction technique should be researched extensively, including any and all means it uses to communicate with other devices. The USB Power Delivery protocol is a communications mode that has the potential to increase the device attack surface. The idea is that, if undocumented means exist to communicate with the device, then hidden features and security vulnerabilities could be identified and exploited to facilitate data acquisition.

The USB Power Delivery protocol provides a uniform means for vendors to implement power negotiation between power sources and devices such as mobile phones and personal computers in order to maximize the charging current. The power source can support different power configurations, one power profile for a mobile phone and a different profile for a personal computer, to enable the devices to obtain the appropriate currents and voltages. Devices can also use the protocol to request higher currents and voltages from power sources. In the case of two non-power-source devices (e.g., two mobile phones), the devices can negotiate a power delivery profile so that one device can charge the other. Another

example is a monitor connected to a personal computer where the protocol enables the monitor to draw power from the personal computer if it is not connected to an external power source. If the monitor is connected to an external power source, then it could provide power to the personal computer. All these negotiations occur over the same USB cable unbeknownst to the user.

The USB Power Delivery protocol is of interest from a digital forensics perspective because it supports inter-device communications. These communications could be exploited to expand the attack surface of one or both devices, enabling the acquisition of data that is otherwise inaccessible. The focus is on vendor-defined messages in the USB Power Delivery protocol. Undocumented messages discovered in other protocols have been demonstrated to enable firmware updates, memory dumps and even backdoors. This chapter presents a black-box testing approach for revealing proprietary messages supported by the USB Power Delivery protocol that could be leveraged in digital forensic investigations to acquire data stored in devices that support the protocol.

II.2 Related Work

Allowing vendors to incorporate proprietary vendor-defined messages or commands in protocols to provide custom functionality has led to the release of numerous consumer devices that potentially respond to undocumented commands with unknown behavior. This can have devastating security implications. As demonstrated by Alendal et al. [2], vendor-specified SCSI commands can be used to bypass authentication on self-encrypting hard drives. Whether this research represents the best-case scenario for law enforcement or the worst-case scenario for the vendor, one cannot ignore the fact that the existence of hidden commands must be tested carefully. Indeed, as devices and firmware change over time, such testing should be performed regularly by law enforcement and security researchers.

Testing the USB Power Delivery protocol for hidden commands requires a means for emulating the protocol. Reydarns et al. [3] have demonstrated the use of USB Power Delivery protocol emulation in testing different power configurations for a power source. However, there is little, if any, research on the security of the USB Power Delivery protocol and nothing related to digital forensics. This research is important because it comprehensively analyzes the USB Power Delivery protocol and attempts to discover how vendor-defined protocol messages could be leveraged to assist digital forensic examinations of devices that support the protocol.

II.3 USB Power Delivery Protocol

Revision 1.0 (version 1.0) of the USB Power Delivery protocol specification was released in 2012; several revisions have been released since, the most recent being Revision 2.0 (version 1.3) and Revision 3.0 (version 1.2) [4]. The protocol provides a uniform means for devices to negotiate power supply configurations across vendors. It is typically supported by devices with a USB Type-C port/connector with dedicated CC1 and CC2 links (Figure II.1). The USB Type-C connection is reversible, enabling devices to communicate on either CC line.

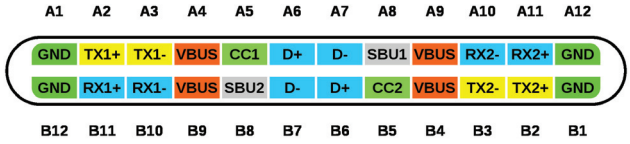


Figure II.1: USB Type-C pinout [5].

The message-based USB Power Delivery protocol has three types of messages: (i) control messages; (ii) data messages; and (iii) extended messages. Control messages are short messages that typically require no data exchange. Data messages contain data objects that are transmitted between devices. Extended messages are essentially data messages with larger data payloads. The USB Power Delivery protocol leverages the three message types to define a wide range of standard messages, which enable devices to communicate and negotiate power source configurations.

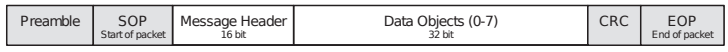


Figure II.2: Data message packet.

Figure II.2 shows a data message packet comprising a preamble for synchronization, start of packet (SOP), message header, up to eight data objects of 32-bits each, CRC and end of packet (EOP). The preamble, SOP, CRC and EOP are part of the physical transport layer; they are common to all three types of messages. The message header and the optional data objects are only found in data messages.

Table II.1 lists example control and data messages in the USB Power Delivery protocol.

The USB Power Delivery protocol supports different standard message sets as indicated by the protocol specification revisions, currently Revision 2.0 and Revision 3.0. Interested readers are referred to the protocol specifications [4] for information pertaining to the differences between the

message sets. Revision 3.0 is functionally the same as Revision 2.0, except for new features such as USB authentication.

The USB Power Delivery protocol also enables cables to take part in communications; a device can communicate with a cable directly using the start of packet. Such electronically-marked cables (EMCA) enable devices to ensure that the cable supports high voltage/current power source configurations. According to the protocol specification, devices can negotiate direct current levels up to 5 A, corresponding to a maximum of 100 W at 20 V between devices connected via an EMCA cable. Passive (non-EMCA) cables are rated for a maximum direct current of 3 A, which corresponds to 15 W at 5 V, 36 W at 12 V or 60 W at 20 V.

Figure II.3 shows a typical protocol negotiation – referred to as an explicit contract between two devices or port pairs. According to the standard, all port pairs are required to make an explicit contract. In a contract, the device (port) that consumes power is called the sink and the device (port) that provides power is called the source.

Vendors may implement novel functionality using proprietary vendor-defined messages, a subgroup of data messages in the USB Power Delivery protocol. Similar features are found in other protocols, such as vendor-specific commands in the SCSI protocol [6]. These commands are implemented and used only by vendors for internal purposes such as debugging, factory setup and proprietary communications with vendor software; the commands are not used in normal device operations. Vendor commands are rarely documented because they are reserved for internal use.

Figure II.4 shows a vendor-defined message (VDM) packet in the USB Power Delivery protocol. Vendor-defined messages are of two types: (i) structured; and (ii) unstructured. Structured vendor-defined message commands are defined in the USB Power Delivery protocol standard whereas unstructured vendor-defined message commands are implemented by vendors on an *ad hoc* basis. Note that a "command" is a subgroup of "message," which is either a structured vendor-defined message or an unstructured vendor-defined message. Thus, while structured vendor-defined messages have predefined command sets in the protocol specification, unstructured vendor-defined messages can correspond to commands defined by vendors.

Because vendor-defined messages are a type of data message, there is a size limitation on the amount of data a message can contain – this corresponds to the size of six vendor data objects (VDOs) plus the 32-bit vendor-defined message header. A vendor data object contains a 32-bit value (data). To prevent vendors from implementing conflicting messages, the protocol requires either the standard vendor ID (SVID) defined

Table II.1: Control and data messages in Revision 3.0 (Version 1.2).

Control Messages	*Data Messages*
GoodCRC	**Source_Capabilities**
GotoMin	**Request**
Accept	**BIST**
Reject	**Sink_Capabilities**
Ping	**Battery_Status**
PS_RDY	**Alert**
Get_Source_Cap	**Get_Country_Info**
Get_Sink_Cap	**Vendor_Defined**
DR_Swap	
PR_Swap	
VCONN_Swap	
Wait	
Soft_Reset	
Not_Supported	
Get_Source_Cap_Extended	
Get_Status	
FR_Swap	
Get_PPS_Status	
Get_Country_Codes	

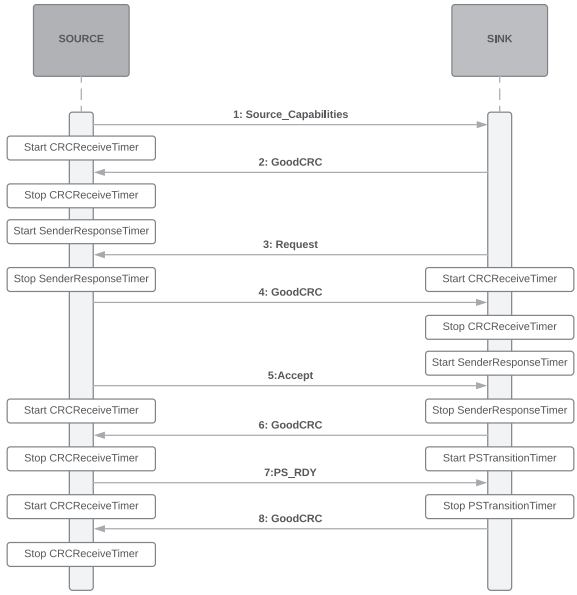


Figure II.3: Simplified explicit contract negotiation.

Figure II.4: Vendor-defined message packet.

in the protocol specification or a vendor ID (VID) to be part of the vendor-defined message header. This means that a vendor must use one of its 16-bit USB Implementers Forum (USB-IF) assigned vendor IDs [7] in any vendor-defined message it implements.

Example vendor IDs are `0x05ac` (Apple) and `0x04e8` (Samsung). As shown in Figures II.5 and II.6, the structured vendor ID and vendor ID are required to be part of the corresponding vendor-defined message headers. Thus, a vendor with a valid USB-IF-assigned vendor ID can implement any command that contains up to six additional vendor data objects in one vendor-defined message. The command is the second part of the vendor-defined message header that can be any 15-bit value in the case of an unstructured vendor-defined message. Table II.2 shows example structured vendor-defined message commands.

Figure II.5: Unstructured vendor-defined message header.

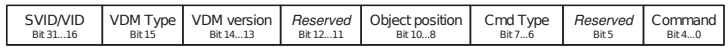


Figure II.6: Structured vendor-defined message header.

II.4 Methodology

Devices come in different architectures from numerous vendors and without source code or firmware that implement the USB Power Delivery protocol. Therefore, a black-box method was attempted to test the existence of vendor-defined messages in the protocol. One approach is to analyze protocol communications between devices from the same vendor and determine if vendor-defined messages are employed. This assumes that, if such messages exist, the connected devices initiate their use by default.

Instead, a more active approach that directly communicates with a test device was employed. Since no solution was available to communicate with devices via the USB Power Delivery protocol, a home-grown approach was employed. A detailed description of this approach is beyond the scope of this chapter. However, the concept is simple – set up a device to act

Table II.2: Structured commands in Revision 3.0 (version 1.2).

Structured Vendor-Defined Message Commands
Discover Identity
Discover SVIDs
Discover Modes
Enter Mode
Exit Mode
Attention
SVID Specific Commands (defined by the SVID)

as the source, establish a connection with the test device and check for vendor-defined messages.

Testing for vendor-defined messages sounds simple, but the reality is quite different. Because the protocol specification states that any vendor-defined message must include a vendor ID, it is necessary to know or guess the expected vendor ID of the device of interest. This is important because a device would not respond to a vendor-defined message containing a correctly-guessed command but an incorrect vendor ID in the header.

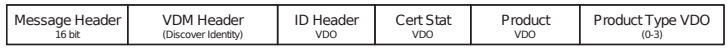


Figure II.7: Discover Identity reply packet.

Fortunately, it is possible to leverage the Discover Identity command in the structured vendor-defined message command set shown in Table II.2. This command is required by the USB Power Delivery protocol, so all devices should support this command. The command, which enables devices and cables to identify other end points, has a predefined reply packet format with a fixed number of vendor data objects and their content (Figure II.7). The ID header of the 32-bit vendor data object has bits 0–15 reserved for the device USB-IF vendor ID. A connected device reveals its vendor ID upon receiving a Discover Identity command.

The protocol specification also states that structured vendor-defined messages shall only be used when an explicit contract is in place (except for a small number of cables that are not relevant in this context). The same holds true for unstructured vendor-defined messages. Thus, a device will not reply to a vendor-defined message until an explicit contract is in place (i.e., a power source configuration has been negotiated). Therefore, it is required to simulate a complete explicit contract negotiation with a test device before a vendor-defined message can be received.

This makes it necessary to simulate many messages (Figure II.3) with

corresponding time-outs such as CRCReceiveTimer (maximum 1.1 ms), SenderResponseTimer (maximum 30 ms) and PSTransitionTimer (maximum 550 ms). Since the protocol defines the time-out values, the reply to a packet must be provided in time or the device will time out. Many of these requirements are strict, so the simulator must have a quick response, which, in turn, may render a pure software solution infeasible.

By negotiating an explicit contract with a device, it is possible to explore the existence of unstructured vendor-defined commands. Using the vendor ID captured from the response of a device to a Discover Identity command, different unstructured vendor-defined commands could be sent to the device and the responses, if any, could be examined. This can be done by brute forcing the lower 15 vendor use bits of the unstructured vendor-defined message header (Figure II.5) with a fixed vendor ID for each device.

Two approaches are possible. The first is to attempt to measure the skews in the timing of device responses. The second is to test for device responses other than the expected GoodCRC message. Testing for timing skews could indicate that the device spent additional time to process a correctly-guessed unstructured vendor-defined command. However, this approach requires high resolution timers. Unfortunately, the experimental setup could only measure the time elapsed from when a packet was sent to when the response was received, which was much too inaccurate. Therefore, the second approach involving device responses other than the expected GoodCRC message was employed in the experiments.

II.5 Experimental Results

Not every device with a USB Type-C connector is enabled for the USB Power Delivery protocol. If a test device with a USB Type-C connector does not respond with a GoodCRC message to the Source_Capabilities message in an explicit contract negotiation (Figure II.3), then the device can be assumed to be non-protocol-enabled.

According to Section 6.2.1.1.5 of USB Power Delivery Protocol Specification Revision 3.0 (v.1.2) [4], the source shall set its highest supported specification revision in the specification revision field of the Source_Capabilities message and the sink shall reply with its highest supported specification revision in the specification revision field of the Request message (Figure II.3). Because the specification states that the specification revision field value should be backwards compatible, this means the highest version can always be simulated in the first Source_Capabilities message acting as the source and the Request response from the device can

Table II.3: Test devices with USB Type-C connectors and protocol support.

Device/ Model	Firmware version	Spec. Rev.	Exposed VID
HTC 10/ 2PS6200	1.90.401.5	2.0	0x0bb4 (HTC)
HTC U11/ 2PZC100	1.13.401.1	3.0	0x05c4 (Qualcomm)
Huawei Mate 10 Pro/ BLA-L29	8.0.0.137(C432)	2.0	0x12d1 (Huawei)
LG G5/ LG-H850	V10i-EUR-XX MMB29M	2.0	0x0000 (Unknown)
Nokia 8 Sirocco/ TA-1005	00WW_3_10F	2.0	0x05c6 (Qualcomm)
Samsung Galaxy S9/ G960F	G960FXXU2BRH7	3.0	0x04e8 (Samsung)

then be checked.

After negotiating a complete explicit contract (Figure II.3) with a test device, a Discover Identity message was sent to the device to obtain the USB-IF vendor ID from the device. Table II.3 shows the test devices with USB Type-C connectors that were determined via this technique to support the USB Power Delivery protocol.

With an explicit contract in place with a test device with protocol support and its USB-IF vendor ID known, the next step was to send arbitrary protocol messages to the device and test the responses. Specifically, unstructured vendor-defined messages were sent with the vendor ID set to the appropriate value, type set to 0 (i.e., unstructured) and vendor use set to different values corresponding to commands (Figure II.5). The responses were analyzed and any response other than the expected GoodCRC was assumed to be an attempt by the test device to reply to the random "command" it received.

A commercial USB Power Delivery protocol recorder was used to capture communications with the test devices. Table II.4 shows an example capture of messages to and from the Huawei test device that was configured as the sink. The message capture shows the entire explicit contract negotiation (message IDs 286–309) and the USB-IF vendor ID discovery (message IDs 312–327), which are followed by two unstructured vendordefined message brute force attempts (message IDs 330–334 and message IDs 337–341). Note that the Huawei device did not respond to the two unstructured vendor-defined message tests with anything other than the expected GoodCRC message.

Very few test devices responded to the brute force test. In fact, only the Samsung device replied with anything other than a GoodCRC message,

Table II.4: Huawei Mate 10 Pro (BLA-L29) message capture.

Index	m:s.ms.us	Role	Message	Data
284	0:41.044.922		Hard Reset	
286	0:43.577.218	Source:DFP	[0]Source_Cap	A1 11 F0 90 01 08 FE CA B7 52
290	0:43.577.879	Sink:UFP	[0]GoodCRC	41 00 BB 6C BB A8
293	0:43.580.754	Sink:UFP	[0]Request	42 10 C8 20 03 13 52 0F 95 B7
297	0:43.581.374	Source:DFP	[0]GoodCRC	A1 01 C1 AF C2 81
300	0:43.582.060	Source:DFP	[1]Accept	63 03 21 7B 00 96
303	0:43.582.586	Sink:UFP	[1]GoodCRC	41 02 97 0D B5 46
306	0:43.583.283	Source:DFP	[2]PS_RDY	A6 05 1F FD EE C9
309	0:43.583.915	Sink:UFP	[2]GoodCRC	41 04 A2 A8 D6 AF
312	0:43.737.641	Source:DFP	[0]VDM:DiscIdentity	6F 11 01 80 00 FF 76 31 6B 61
316	0:43.738.185	Sink:UFP	[0]GoodCRC	41 00 BB 6C BB A8
319	0:43.744.295	Sink:UFP	[1]VDM:DiscIdentity	4F 52 41 80 00 FF D1 12 00 EC 00 00 00 00 00 00 7E 10 01 00 00 11 80 C1 C7 56
327	0:43.745.502	Source:DFP	[1]GoodCRC	61 03 A3 19 36 A4
330	0:44.918.448	Source:DFP	[1]VDM:Unstructured	6F 13 01 00 D1 12 0D 13 06 BC
334	0:44.919.214	Sink:UFP	[1]GoodCRC	41 02 97 0D B5 46
337	0:46.507.375	Source:DFP	[2]VDM:Unstructured	6F 15 02 00 D1 12 43 49 F3 21
341	0:46.507.960	Sink:UFP	[2]GoodCRC	41 04 A2 A8 D6 AF

and only for some messages.

Table II.5 shows an example capture of messages to and from the Samsung Galaxy S9 test device that was configured as the sink. Once again, the message capture shows the entire explicit contract negotiation (message IDs 5442–5465) and the USB-IF vendor VID discovery (message IDs 5468–5482). These are followed by the first unstructured vendor-defined message test (message ID 5485). The sent message has an unstructured vendor-defined message header of `0x04e80001`, which is decoded according to Figure 5 as vendor ID: `0x04e8`, type: `0` and vendor use: `0x0001` (15-bit value).

Note that this unstructured vendor-defined message received a response other that the GoodCRC (message ID 5492). The response has an unstructured vendor-defined message header of `0x04e80041`, which is decoded according to Figure II.5 as vendor ID: `0x04e8`, type: `0` and vendor use: `0x0041`. This message appears to be a reply with no additional data (i.e., vendor data objects).

A similar situation is seen for message 5499 with vendor use: `0x0002`, whose response (message ID 5506) has vendor use: `0x0042` and four additional vendor data objects: `0x00000000 0x00000000 0x00000000` and `0x00000000`.

The two vendor use command/reply pairs of `0x0001/0x0041` and `0x0002/0x0042` imply that bit 6 (`0x0040`) may be an ACK bit. If the unstructured headers are interpreted as structured headers (Figure II.6), then bits 6–7 correspond to type where `0x1` (bit 6 set) corresponds to an ACK. Of course, the real situation is not clear, but it does appear that the vendor

Table II.5: Samsung Galaxy S9 (G960F) message capture.

Index	m:s.ms.us	Role	Message	Data
5440	14:36.248.230		Hard Reset	
5442	14:39.309.886	Source:DFP	[0]Source_Cap	A1 11 F0 90 01 08 FE CA B7 52
5446	14:39.310.395	Sink:UFP	[0]GoodCRC	41 00 BB 6C BB A8
5449	14:39.311.982	Sink:UFP	[0]Request	82 10 F0 C0 03 13 08 11 00 3A
5453	14:39.312.708	Source:DFP	[0]GoodCRC	A1 01 C1 AF C2 81
5456	14:39.313.284	Source:DFP	[1]Accept	63 03 21 7B 00 96
5459	14:39.313.979	Sink:UFP	[1]GoodCRC	41 02 97 0D B5 46
5462	14:39.314.462	Source:DFP	[2]PS_RDY	A6 05 1F FD EE C9
5465	14:39.315.049	Sink:UFP	[2]GoodCRC	41 04 A2 A8 D6 AF
5468	14:39.471.248	Source:DFP	[0]VDM:DiscIdentity	6F 11 01 80 00 FF 76 31 6B 61
5472	14:39.471.866	Sink:UFP	[0]GoodCRC	41 00 BB 6C BB A8
5475	14:39.476.288	Sink:UFP	[1]VDM:DiscIdentity	8F 42 41 80 00 FF E8 04 00 D1 00 00 00 00 00 00 60 68 C2 B2 A2 9E
5482	14:39.477.131	Source:DFP	[1]GoodCRC	61 03 A3 19 36 A4
5485	14:40.650.372	Source:DFP	[1]VDM:Unstructured	6F 13 01 00 E8 04 E6 2B 56 46
5489	14:40.651.199	Sink:UFP	[1]GoodCRC	41 02 97 0D B5 46
5492	14:40.654.796	Sink:UFP	[2]VDM:Unstructured	4F 14 41 00 E8 04 FD AA CE 68
5496	14:40.655.473	Source:DFP	[2]GoodCRC	61 05 96 BC 55 4D
5499	14:41.828.228	Source:DFP	[2]VDM:Unstructured	6F 15 02 00 E8 04 A8 71 A3 DB
5503	14:41.829.056	Sink:UFP	[2]GoodCRC	41 04 A2 A8 D6 AF
5506	14:41.833.325	Sink:UFP	[3]VDM:Unstructured	4F 56 42 00 E8 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 34 A1 0A 25
5514	14:41.834.581	Source:DFP	[3]GoodCRC	61 07 BA DD 5B A3
5517	14:43.008.455	Source:DFP	[3]VDM:Unstructured	6F 17 02 00 E8 04 C8 22 63 A1
5521	14:43.009.071	Sink:UFP	[3]GoodCRC	41 06 8E C9 D8 41
5524	14:43.013.435	Sink:UFP	[4]VDM:Unstructured	4F 58 42 00 E8 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 84 AD C5 F6
5532	14:43.014.693	Source:DFP	[4]GoodCRC	61 09 BD F0 E3 44
5535	14:44.180.619	Source:DFP	[4]VDM:Unstructured	6F 19 03 00 E8 04 CC FB EF A6
5539	14:44.181.134	Sink:UFP	[4]GoodCRC	41 08 89 E4 60 A6
5542	14:45.761.683	Source:DFP	[5]VDM:Unstructured	6F 1B 02 00 E8 04 C9 CF 93 64
5546	14:45.762.289	Sink:UFP	[5]GoodCRC	41 0A A5 85 6E 48
5549	14:45.766.649	Sink:UFP	[5]VDM:Unstructured	4F 5A 42 00 E8 04 0D DA 95 63 4A 97 17 B5 F5 34 11 47 53 7E C9 E9 8C 35 3F 0E
5557	14:45.767.917	Source:DFP	[5]GoodCRC	61 0B 91 91 ED AA
5560	14:46.933.424	Source:DFP	[6]VDM:Unstructured	6F 1D 01 00 E8 04 87 95 66 F9
5564	14:46.934.042	Sink:UFP	[6]GoodCRC	41 0C 90 20 0D A1
5567	14:46.937.851	Sink:UFP	[6]VDM:Unstructured	4F 1C 41 00 E8 04 3C E1 BE 58
5571	14:46.938.566	Source:DFP	[6]GoodCRC	61 0D A4 34 8E 43
5574	14:48.114.825	Source:DFP	[7]VDM:Unstructured	6F 1F 02 00 E8 04 09 69 13 91
5578	14:48.115.442	Sink:UFP	[7]GoodCRC	41 0E BC 41 03 4F
5581	14:48.119.820	Sink:UFP	[7]VDM:Unstructured	4F 5E 42 00 E8 04 0D DA 95 63 4A 97 17 B5 F5 34 11 47 53 7E C9 E9 37 31 C6 1C
5589	14:48.121.075	Source:DFP	[7]GoodCRC	61 0F 88 55 80 AD
5592	14:49.303.445	Source:DFP	[0]VDM:Unstructured	6F 11 03 00 E8 04 0D B0 9F 96
5596	14:49.304.274	Sink:UFP	[0]GoodCRC	41 00 BB 6C BB A8
5599	14:50.881.168	Source:DFP	[1]VDM:Unstructured	6F 13 02 00 E8 04 08 84 E3 54
5603	14:50.881.789	Sink:UFP	[1]GoodCRC	41 02 97 0D B5 46
5606	14:50.886.156	Sink:UFP	[0]VDM:Unstructured	4F 50 42 00 E8 04 60 B3 A9 5A 65 3F 48 3C 3A D6 13 DC 2D 32 8D 16 F6 75 A3 FE
5614	14:50.887.366	Source:DFP	[0]GoodCRC	61 01 8F 78 38 4A

may have mixed the two types of vendor-defined message headers.

Investigating further, the response (message ID 5506) with vendor use set to `0x0042` also has four additional four vendor data objects: texttt0x00000000 0x00000000 0x00000000 and `0x00000000`. This appears to be data sent back to the source side from the sink. All the vendor data objects contain zeroes in the replies to two consecutive messages with vendor use set to `0x0002` (message IDs 5499 and 5517).

However, when a different message (message ID 5535) is sent to the device with vendor use set to `0x0003`, then a completely different reply is received with vendor use set to `0x0002` (message ID 5542) and four vendor data objects: `0x6395da0d 0xb517974a 0x471134f5` and `0xe9c97e53` (message ID 5549). Sending message 5535 again (message ID 5574) yields the same four vendor data objects (message ID 5581). However, another message with vendor use set to `0x0003` (message ID 5592) once again changes the vendor data objects for vendor use set to `0x0002`. Specifically, the four vendor data objects are: `0x5aa9b360 0x3c483f65 0xdc13d63a` and `0x168d322d` (message ID 5606).

It appears that data in the form of vendor data objects is received from the device and different data is received when sending a specific message with vendor use set to 0x0003. The four vendor data objects appear to change in pseudorandom order. Another observation is that, when a message is sent with vendor use set to `0x0002` along with four random vendor data objects (`0xabb3471c, 0xae7bf32e, 0x827909f9, 0xbbc63b02`), a reply is received with the same vendor data objects (Table II.6). This implies that a message with vendor use set to `0x0002` corresponds to an initialization command. Repeating the messages with vendor use set to `0x0003` and `0x0002` gives different vendor data objects, which may correspond to some form of encryption or obfuscation.

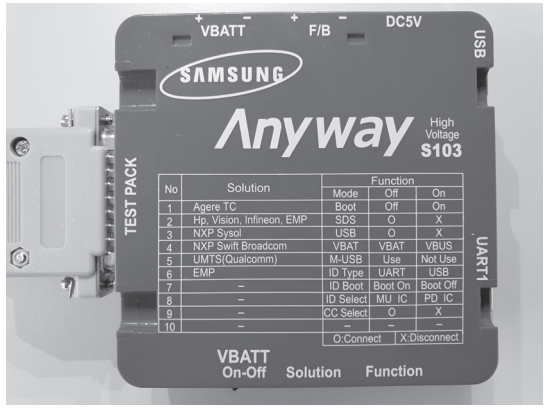
Table II.6: Samsung Galaxy S9 (G960F) message capture.

Index	m:s.ms.us	Role	Message	Data
162	0:06.589.154	Source:DFP	[1]VDM:Unstructured	6F 13 01 00 E8 04 E6 2B 56 46
166	0:06.589.982	Sink:UFP	[1]GoodCRC	41 02 97 0D B5 46
169	0:06.594.059	Sink:UFP	[1]VDM:Unstructured	4F 12 41 00 E8 04 5D 5F 8E E7
173	0:06.594.675	Source:DFP	[1]GoodCRC	61 03 A3 19 36 A4
176	0:06.629.222	Source:DFP	[2]VDM:Unstructured	6F 55 02 00 E8 04 1C 47 B3 AB 2E F3 7B AE F9 09 79 82 02 3B C6 BB 1A D4 E8 41
184	0:06.630.376	Sink:UFP	[2]GoodCRC	41 04 A2 A8 D6 AF
187	0:06.635.264	Sink:UFP	[2]VDM:Unstructured	4F 54 42 00 E8 04 1C 47 B3 AB 2E F3 7B AE F9 09 79 82 02 3B C6 BB 51 65 55 63
195	0:06.636.524	Source:DFP	[2]GoodCRC	61 05 96 BC 55 4D

Sending two identical runs of the messages in Table II.5 gives the same results and any randomization of the four vendor data objects sent with vendor use set to `0x0002` yields seemingly random reply vendor data

objects when intermingled with messages with vendor use set to `0x0003`. This strengthens the belief that encryption is in place and that the message with vendor use set to `0x0002` is either transmitting a key or an initialization vector for a symmetric cipher.

Because the results indicate that Samsung devices respond to vendor-defined messages in the USB Power Delivery protocol, additional experiments were conducted to confirm the results. The experiments employed a special factory test device called the Samsung Anyway S103 (Figure II.8). This device enables a console interface provided by the device bootloader, which is useful for debug logging and other activities. The same console can be reached via a custom USB connector and a simple RS232-to-USB serial converter on older devices with micro-USB connectors [8]. Alendal et al. [9] employed this type of connection to demonstrate an exploit targeting Samsung devices with a certain security vulnerability. The exploit assisted in bypassing a certain security feature in the devices. This demonstrates the importance of expanding the attack surface of a device by enabling the factory test feature.



No	Solution	Function		
		Mode	Off	On
1	Agere TC	Boot	Off	On
2	Hp, Vision, Infineon, EMP	SDS	O	X
3	NXP Sysol	USB	O	X
4	NXP Swift Broadcom	VBAT	VBAT	VBUS
5	UMTS(Qualcomm)	M-USB	Use	Not Use
6	EMP	ID Type	UART	USB
7	–	ID Boot	Boot On	Boot Off
8	–	ID Select	MU IC	PD IC
9	–	CC Select	O	X
10	–	–	–	–

Figure II.8: Samsung Anyway S103.

The special factory device was hard to obtain as it is usually provided to Samsung device repair shops and similar outlets. However, a factory device was procured to communicate with the Samsung test device using the USB Power Delivery protocol. Table II.7 shows a message capture with the Samsung Anyway S103 and Samsung Galaxy S9 configured as the source and sink, respectively (the vendor data objects are partially redacted). Note that the communications in the message capture did not involve an explicit contract negotiation as required in the protocol specification. Instead, immediate vendor-defined message communications were

conducted using the discovered vendor-defined messages. The capture corresponds to a vendor-defined message with vendor use set to `0x0001`, followed by a vendor-defined message with vendor use set to `0x0002` that provides four pseudorandom vendor data objects. These are followed by several vendor-defined messages with vendor use set to `0x0003`, each containing four vendor data objects with seemingly pseudorandom data.

Table II.7: Samsung Anyway S103 and Samsung Galaxy S9 message capture.

Index	Time	Role	Message	Data
1	0:03.900.730	Source:DFP	[0]VDM:DiscIdentity	6F 11 01 80 00 FF 76 31 6B 61
5	0:03.901.546	Sink:UFP	[0]GoodCRC	41 00 BB 6C BB A8
8	0:03.905.272	Sink:UFP	[0]VDM:DiscIdentity	8F 40 41 80 00 FF E8 04 00 D1 00 00 00 00 00 00 60 68 05 22 9E 4A
15	0:03.906.336	Source:DFP	[0]GoodCRC	61 01 8F 78 38 4A
18	0:03.906.881	Source:DFP	[1]VDM:Unstructured	6F 13 01 00 E8 04 E6 2B 56 46
22	0:03.907.590	Sink:UFP	[1]GoodCRC	41 02 97 0D B5 46
25	0:03.912.440	Sink:UFP	[1]VDM:Unstructured	4F 12 41 00 E8 04 5D 5F 8E E7
29	0:03.913.109	Source:DFP	[1]GoodCRC	61 03 A3 19 36 A4
32	0:03.913.649	Source:DFP	[2]VDM:Unstructured	6F 55 02 00 E8 04 0C DD BB FF <REDACTED>
40	0:03.914.888	Sink:UFP	[2]GoodCRC	41 04 A2 A8 D6 AF
43	0:03.919.998	Sink:UFP	[2]VDM:Unstructured	4F 54 42 00 E8 04 0C DD BB FF <REDACTED>
51	0:03.921.093	Source:DFP	[2]GoodCRC	61 05 96 BC 55 4D
54	0:03.922.149	Source:DFP	[3]VDM:Unstructured	6F 57 03 00 E8 04 E6 A9 7F 72 94 CE B1 B6 54 BA B7 75 6A F1 89 B8 01 65 20 E8
62	0:03.923.388	Sink:UFP	[3]GoodCRC	41 06 8E C9 D8 41
65	0:03.931.556	Sink:UFP	[3]VDM:Unstructured	4F 56 43 00 E8 04 9F B2 F5 F9 F1 68 E2 AF E5 AA 22 73 D0 77 6A 2E B6 3A A9 FB
73	0:03.932.759	Source:DFP	[3]GoodCRC	61 07 BA DD 5B A3
76	0:03.934.596	Source:DFP	[4]VDM:Unstructured	6F 59 03 00 E8 04 F7 96 A6 2A 08 BB A9 6E 38 40 E4 AF 33 43 7A 23 E6 D7 A8 E9
84	0:03.935.837	Sink:UFP	[4]GoodCRC	41 08 89 E4 60 A6
87	0:03.942.701	Sink:UFP	[4]VDM:Unstructured	4F 58 43 00 E8 04 9A 01 DB AE 9A 39 26 77 B0 A8 2D 11 A2 C1 76 80 1E 08 1E C2
95	0:03.943.902	Source:DFP	[4]GoodCRC	61 09 BD F0 E3 44

Next, the Samsung Anyway S103 factory device was removed as the source and a blind replay from the source side of the communications was attempted. The idea was that, if the source messages from the Samsung Anyway S103 device were replayed and the same sink messages were received from the test device, then the Samsung Anyway S103 device was essentially being emulated. This test was an immediate success. The key result is that the same console reached on micro-USB Samsung devices was enabled without the assistance of the Samsung Anyway S103 factory device.

The successful message replay strengthens the belief that encryption is involved and that the first four vendor data objects in the vendor-defined message with vendor use set to `0x0002` are crucial to initialization. These vendor data objects could correspond to an initialization vector or perhaps even the key to a symmetric cipher. However, experiments with several symmetric ciphers using the four vendor data objects as the key to decrypt

vendor data objects in messages with the vendor use set to `0x0003` did not yield positive results.

II.6 Conclusions

The principal contribution of this research is a black-box testing methodology and implementation for revealing and analyzing proprietary USB Power Delivery protocol messages. The experimental results demonstrate that at least one common mobile device, a Samsung Galaxy S9, is amenable to the testing methodology. In particular, the device responds to certain vendor-defined messages and the responses indicate the use of encryption, which raises the possibility of capturing initialization vectors and keys for symmetric ciphers. Another important result is the ability to enable factory device features in a test device in order to obtain valuable log data from the device and to widen its attack surface.

Future research will continue the investigation of vendor-defined messages in the USB Power Delivery protocol. Since vendors may also implement hidden features in other parts of the protocol, a promising approach is to investigate the role of the sink device that consumes power. Connecting two devices that typically serve as sinks – like two mobile phones – causes one device to assume the source role and provide power to the other device. This source-sink relationship could potentially be exploited to expand the attack surface or even to directly acquire data.

Future research will also investigate potential security vulnerabilities. This is challenging because it is not known how to instrument a USB Power Delivery chip for feedback (e.g., if it crashes or demonstrates anomalous behavior). An alternative approach is to conduct a source code review or extract the chip firmware and apply reverse engineering techniques. Another approach is to analyze device-side communications with the USB Power Delivery chip, which could reveal interesting features or vulnerabilities in the chip logic as well in the operating system.

The popularity of USB Type-C connectors is increasing and large numbers of consumer devices will support the USB Power Delivery protocol. It is hoped that this work will stimulate research on the protocol and its implementations to advance device security and forensics.

Acknowledgements

This research was sponsored by the Norwegian Research Council IKTPLUSS Program under the Ars Forensica Project No. 248094/O70.

II.7 References

[1] G. Alendal, S. Axelsson and G. O. Dyrkolbotn, 'Exploiting Vendor-Defined Messages in the USB Power Delivery Protocol,' in *Advances in Digital Forensics XV*, G. Peterson and S. Shenoi, Eds., Springer International Publishing, 2019, pp. 101–118, ISBN: 978-3-030-28752-8.

[2] G. Alendal, C. Kison and modg, *got HW crypto? On the (in)security of a Self-Encrypting Drive series*, Cryptology ePrint Archive, Report 2015/1002, http://eprint.iacr.org/2015/1002, 2015.

[3] H. Reydarns, V. Lauwereys, D. Haeseldonckx, P. van Willigenburg, J. Woudstra and S. D. Jonge, 'The development of a proof of concept for a smart dc/dc power plug based on usb power delivery,' in *Twenty-Second Domestic Use of Energy*, Apr. 2014, pp. 1–4. DOI: 10.1109/DUE.2014.6827761.

[4] USB-IF, *Usb power delivery*, https://www.usb.org/document-library/usb-power-delivery, [Online; accessed 20-September-2018], 2018.

[5] Chindi.ap, *Usb type-c pinout drawing*, https://commons.wikimedia.org/wiki/User:Chindi.ap, 2019.

[6] Technical Committee T10, *Scsi operation codes*, http://www.t10.org/lists/op-num.htm, [Online; accessed 20-September-2018], 2018.

[7] USB-IF, *Getting a vendor id*, https://www.usb.org/getting-vendor-id, [Online; accessed 20-September-2018], 2018.

[8] Nitay Artenstein, *Exploiting android s-boot: Getting arbitrary code exec in the samsung bootloader (1/2)*, http://hexdetective.blogspot.no/2017/02/exploiting-android-s-boot-getting.html, [Online; accessed 27-July-2017], 2017.

[9] G. Alendal, G. O. Dyrkolbotn and S. Axelsson, 'Forensics acquisition - Analysis and circumvention of samsung secure boot enforced common criteria mode,' *Digital Investigation*, vol. 24, S60–S67, 2018, ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2018.01.008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1742287618300409.

Paper III

Leveraging the USB Power Delivery Implementation for Digital Forensic Acquisition

G. Alendal, S. Axelsson and G. O. Dyrkolbotn, 'Leveraging the USB Power Delivery Implementations for Digital Forensic Acquisition,' in *Advances in Digital Forensics XVII*, G. Peterson and S. Shenoi, Eds., Springer International Publishing, 2021, pp. 111–133, ISBN: 978-3-030-88381-2

Abstract

Modern consumer devices present major challenges in digital forensic investigations due to security mechanisms that protect user data. The entire physical attack surface of a seized device such as a mobile phone must be considered in an effort to acquire data of forensic value.

Several USB protocols have been introduced in recent years, including Power Delivery, which enables negotiations of power delivery to or from attached devices. A key feature is that the protocol is handled by dedicated hardware that is beyond control of the device operating systems. This self-contained design is a security liability with its own attack surface and undocumented trust relationships with other peripherals and the main system-on-chips.

This chapter presents a methodology for vulnerability discovery in a black box USB Power Delivery implementation for Apple devices. The protocol and Apple-specific communications are reverse engineered, along with the firmware of the dedicated USB Power Delivery hardware. The investigation of the attack surface and potential security vulnerabilities can facilitate data acquisition in digital forensic investigations.

III.1 Introduction

Law enforcement has special opportunities to leverage security vulnerabilities in digital forensic acquisition. Since law enforcement can physically seize devices, any exposed physical interface on the devices is a potential attack vector for data acquisition.

Exposed interfaces on mobile devices that can be leveraged without physically opening the devices include SIM card slots, SD card slots, audio jacks and USB connectors. Interfaces that are exposed by physically opening devices are UART, JTAG and essentially any on-board peripherals that can be manipulated or replaced. The internal interfaces are often less accessible because opening a device like a mobile phone can be cumbersome and risky. A mobile phone is often glued shut and attempting to open it could disrupt normal operations, especially if the device must be powered on to exploit a specific vulnerability. This situation can occur if a phone is seized after the user has unlocked the device at least once since the device was powered on (i.e., after-first-unlock state), where user keys tied to user credentials are unlocked and more user data is potentially available. Thus, security vulnerabilities exposed via externally-accessible interfaces are preferred over internal interfaces.

The USB connector is one of the most common external interfaces in modern personal computers and embedded devices. As a result, the security of USB protocols is important from a digital forensic perspective. Wang et al. [2] have discussed USB attack strategies on the functional and physical layers, and several vulnerable scenarios for USB connected devices. However, they did not explore the security of the USB Power Delivery protocol.

Tian et al. [3] have investigated USB security. They have examined the security features provided by the USB Type-C connector, specifically, authentication that is included in recent USB Power Delivery revisions. They formally verified the authentication and identified USB attack vectors, but do not discuss implementation details. Examining actual implementations for verification of USB Power Delivery as an attack vector is, therefore, interesting and timely.

USB Power Delivery is a feature in newer devices that is available externally over standard USB physical interfaces such as a USB Type-C connector [4]. It is available on many modern personal computers and mobile phones in the after-first-unlock (AFU) and before-first-unlock (BFU) states. USB Power Delivery enables connected devices to negotiate the optimal power delivery (voltage and current), where one device acts as the source and the other as the consumer (sink).

Because devices can choose to swap the source and sink roles, the

USB Power Delivery protocol supports the negotiation of the direction of power flow. Additionally, the protocol supports the negotiations of multiple devices connected to a single power source as well as re-negotiations at any time if more power is required. The protocol specification allows direct current levels up to 5 A, corresponding to a maximum of 100 W at 20 V. Thus, even the cables need to communicate using the USB Power Delivery protocol to ensure that they support higher current levels. These cables are named "electronically marked cables" (EMCA) [5].

USB Power Delivery employs messages for communications [5]. Revision 2.0 of the protocol has control and data messages. Revision 3.0 specifies additional extended messages that support features such as firmware updates, battery information, manufacturer information and security messages. USB Power Delivery also supports a side-band channel for standard and non-standard vendor-specific communications.

Thus, the source, sink and cable can transmit and receive control, data and extended messages, as well as additional vendor-specific messages. The original and additional features of the protocol raise the question whether the protocol can be considered to be secure from a vulnerability perspective. The code implementing such a feature-rich protocol is large and complex, increasing the likelihood of faults, which include security vulnerabilities. Estimating the ratio of security vulnerabilities per line of code is difficult and cumbersome. Hatton [6] suggests a defect (bug) density of less than ten per thousand lines of code. Ozment and Schechter [7], who evaluated OpenBSD, suggest a vulnerability density three orders of magnitude less. Although the figures are not directly comparable, they indicate that more code increases the likelihood of security vulnerabilities.

The complexity of the USB Power Delivery protocol and its code base increase the likelihood that software security vulnerabilities could have been introduced during design and implementation. The protocol is also implemented in dedicated hardware, which raises questions about the state and integrity of the chip as well as the trust relationships with the rest of the system and the system-on-chip (SoC). This could expose the implementation to "evil maid" attacks [8] that replace the firmware in a USB Power Delivery chip or simply replace the entire chip.

The basis for any vulnerability research is the design and implementation details. Before applying any vulnerability discovery techniques [9, 10], access to code in any form and testing tools are extremely beneficial. Static vulnerability analysis benefits greatly from access to design and code details whereas fuzzing [11] requires simulation testing methods and tools.

Since most USB Power Delivery implementations are proprietary, the availability of source code is limited. Therefore, evaluating and estimat-

ing the likelihood of faults and security vulnerabilities based on lines of code is difficult. Additionally, since the protocol is implemented in dedicated hardware with undocumented interfaces, the ability to analyze and evaluate security via testing is limited. Few tools are available to perform black box testing or fuzzing of the protocol [12]. Extracting firmware from a USB Power Delivery chip and analyzing the firmware are also difficult tasks. But they are important because such proprietary, non-scrutinized code may have many unknown security vulnerabilities.

It appears that USB Power Delivery has potential vulnerabilities in the protocol [13, 14] and its implementations [9]. Other vulnerabilities may arise from hidden features [15], implicit trust relationships[16] and hardware exposures such as evil maid [8]. Clearly, the exploitation of USB Power Delivery should be investigated as a means to enable the forensic acquisition of data from devices that incorporate the hardware and implement the protocol.

This chapter presents a new methodology for evaluating the potential of USB Power Delivery as an attack vector. The focus on USB Power Delivery implementations can provide insights into vulnerabilities. Binary diffing [17] of firmware versions can reveal security patches that can be exploited. Indeed, this research is important to understanding and leveraging USB Power Delivery as an entirely new attack surface for forensic data acquisition.

III.2 USB Power Delivery Protocol

The USB Power Delivery protocol specifications were released in 2012 as Revision 1.0 (version 1.0). The most recent specifications are provided in Revision 2.0 (version 1.3) and Revision 3.0 (version 2.0) [5]. USB Power Delivery offers a uniform method for devices to negotiate power supply configurations across vendors. The protocol is often used by devices with USB Type-C connectors. A USB Type-C connector has dedicated lines, CC1 and CC2, that enable USB Power Delivery communications between devices. However, a USB Type-C connector is not required to support USB Power Delivery; for example, Apple's proprietary Lightning Connector [18] supports USB Power Delivery. In fact, a cable with Apple Lightning and USB Type-C connectors could be used between an Apple device and any other USB Power Delivery enabled device to facilitate communications. The Apple Lightning and USB Type-C connectors are reversible, so the orientations of the connectors are not important.

The message-based USB Power Delivery protocol employs three types of messages: control, data and extended messages. Control messages are

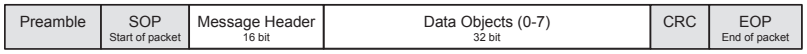


Figure III.1: USB Power Delivery data message packet.

short messages that typically require no data exchange. Data messages contain data objects that are exchanged between devices. Extended messages, introduced in Revision 3.0, are data messages with larger payloads.

Figure III.1 shows the format of a USB Power Delivery data message packet. The packet has a transport portion comprising the preamble, start of packet (SOP), cyclic redundancy check (CRC) and end of packet (EOP) fields, which encapsulates the message header and optional (up to seven) data objects. A data object has a fixed size of 32 bits, allowing for a maximum of 7×32 bits of data per message.

USB Power Delivery supports a wide range of standard messages to facilitate negotiations of power source configurations between devices. Table III.1 lists the messages supported by Revision 2.0 (Version 1.3) and/or Revision 3.0 (Version 2.0). The backward compatibility means that protocol complexity increases in new revisions as new messages are added but existing messages are not eliminated.

Some of the standard messages in Table III.1 have sub-types. For example, a Vendor_Defined message (VDM) can be structured or unstructured. Structured VDMs have commands defined in the standard (Table III.2). Unstructured VDM commands are defined by vendors and are undocumented. Vendors are free to implement proprietary communications using unstructured VDMs as demonstrated in [12]. Enabling vendors to add messages over and above the standard messages results in increased complexity and firmware code size.

To avoid conflicts when implementing proprietary vendor messages, VDMs require a standard vendor ID (SVID) defined in the specification or a vendor ID (VID) to be part of the VDM header. A VID is a unique 16-bit identifier assigned by the USB Implementers Forum [19]. A vendor with a valid VID is free to implement any VDMs needed to operate its USB Power Delivery enabled devices. Apple devices commonly use the VID `0x05ac` [20].

Connected devices negotiate power delivery via an explicit contract. Typically, this is initiated by the source device that sends a Source - Capabilities data message to which the sink replies with a GoodCRC message followed by a Request message (Figure III.2). These responses inform the source that the sink is USB Power Delivery enabled and the highest protocol revision it supports. Specification revision information is included in the message header of the Request message. The highest specification revision supported by the sink corresponds to the highest specification revision

Table III.1: Power Delivery protocol messages.

Control Messages	*Data Messages*	*Extended Messages*
Revision 2.0 and 3.0	*Revision 2.0 and 3.0*	*Revision 3.0 only*
GoodCRC	**Source_Capabilities**	**Source_Capabilities_Extended**
GotoMin	**Request**	**Status**
Accept	**BIST**	**Get_Battery_Cap**
Reject	**Sink_Capabilities**	**Get_Battery_Status**
Ping	**Vendor_Defined**	**Battery_Capabilities**
PS_RDY		**Get_Manufacturer_Info**
Get_Source_Cap	*Revision 3.0 only*	**Manufacturer_Info**
Get_Sink_Cap	**Enter_USB**	**Security_Request**
DR_Swap	**Battery_Status**	**Security_Response**
PR_Swap	**Alert**	**Firmware_Update_Request**
VCONN_Swap	**Get_Country_Info**	**Firmware_Update_Response**
Wait		**PPS_Status**
Soft_Reset		**Country_Info**
		Sink_Capabilities_Extended
Revision 3.0 only		**Country_Codes**
Data_Reset_Complete		
Not_Supported		
Get_Source_Cap_Extended		
Get_Status		
FR_Swap		
Get_PPS_Status		
Get_Country_Codes		
Get_Sink_Cap_Extended		
Data_Reset		

supported by the source, which is indicated in the Source_Capabilities message. Thus, the connected devices know the revision and the message sets that are mutually supported.

III.3 Research Methodology

The methodology for researching USB Power Delivery firmware involves information gathering, monitoring black box testing and simulation, and reverse engineering actual implementations (using binary code, documentation, source code, etc.). The individual methods often aid and overlap each other. For example, static reverse engineering of a binary is often assisted by monitoring and simulation. Even more powerful methods are instrumentation and debugging, which advance reverse engineering and vulnerability discovery.

Table III.2: Structured Vendor_Defined message commands.

Structured VDM Commands
Discover Identity
Discover SVIDs
Discover Modes
Enter Mode
Exit Mode
Attention
SVID Specific Commands

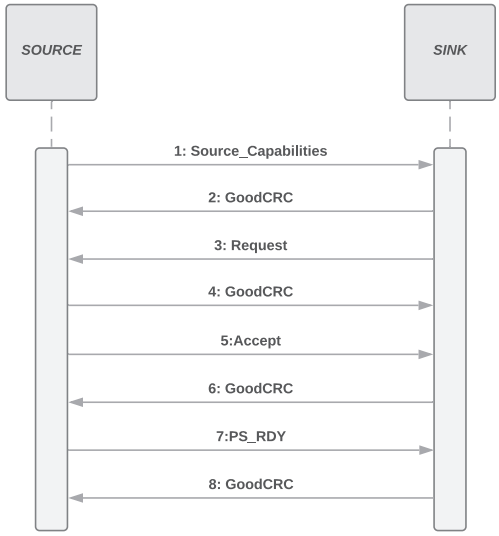


Figure III.2: Generic, source-initiated explicit contract negotiation.

Access to the source code of USB Power Delivery implementations (firmware) is difficult. This is because the source code is developed by the chip vendor and/or device vendor and are considered to be proprietary, business-confidential information. The documentation is also considered confidential and is kept in-house. As a result, the only option for researchers intending to study USB Power Delivery implementations is to extract and reverse engineer firmware.

Reverse engineering firmware involves the extraction of machine code (binary) for a specific chip and applying static and dynamic methods to produce human-readable assembly code [21, 22], following which decompilation is performed to obtain pseudo high-level source code [23].

Static reverse engineering analyzes machine code without executing or interacting with the code [22]. Dynamic reverse engineering analyzes machine code by interacting with and debugging executing code (e.g., using black box testing) [22].

Static and dynamic methods and tools are available for analyzing wellknown machine code structures such as PE [24] and ELF [25] binaries, but they are difficult to come by for hardware-specific and specialized firmware used by USB Power Delivery chips. Obtaining USB Power Delivery chip firmware is challenging. Vendors may include firmware in regular device updates as in the case of Apple iOS updates. However, USB Power Delivery chips may not receive any updates from vendors, requiring researchers to extract the firmware directly from the chips soldered on the targeted devices.

USB Power Delivery firmware can be retrieved from general iOS updates for Apple devices. In fact, the firmware for several USB Power Delivery chips can be obtained by unpacking and investigating the iOS updates that are often distributed in `.ipsw` archives.

Analyzing the firmware of USB Power Delivery chips is complex because the code is often based on specific, often unknown, hardware. In particular, little to nothing may be known about the architecture, memory layout and interfaces. Since the firmware does not directly interact with users, helpful, human-readable, informational/error messages are rarely embedded in the code. This renders static reverse engineering very difficult, making dynamic reverse engineering the only feasible option.

Dynamic reverse engineering involves the execution and observation of the behavior of firmware. A simulation tool can be used to evaluate code execution by communicating with the USB Power Delivery interface and, consequently, the firmware code. This can be accomplished using a proprietary USB Power Delivery simulation device as in [12]. USB Power Delivery messages are sent to the device to assist with static reverse engineering. Specifically, responses to the messages are matched in a trial

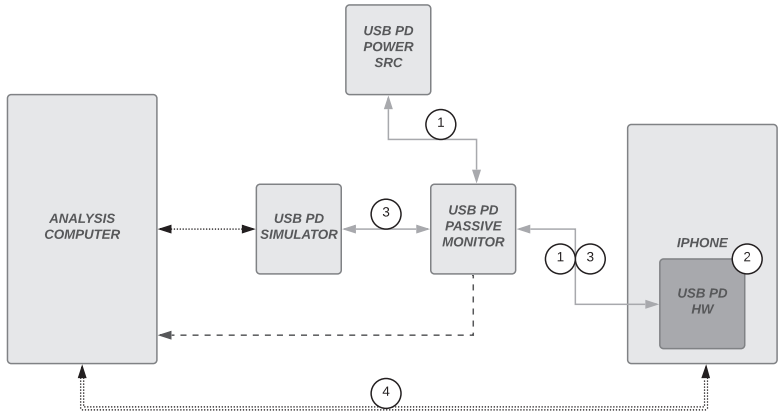


Figure III.3: Experimental setup.

and error manner to identity the corresponding firmware code. However, reversing the firmware and resulting assembly code are still very tedious. Also, the reverse engineering results may not fully match the full feature set of the original source code. Nevertheless, the results would help understand unknown parts of the protocol such as VDMs. The (re)produced pseudo code from the (re)produced assembly code could be used to estimate of the lines of code in the original source code and, thus, the complexity of the implementation.

The production of pseudo code from firmware binaries from different vendors is challenging and difficult to generalize. Therefore, this research opted to pursue a full reverse engineering effort for targeted devices such as Apple iPhones to help generalize the results to a wider selection of devices in the future.

Figure III.3 shows the experimental setup. It incorporates an analysis computer, target iPhone, USB Power Delivery monitor, USB Power Delivery simulator and stock USB Power Delivery power sources.

The general workflow is to first conduct information gathering from open sources. A generic USB PD passive monitoring tool (1 in Figure III.3) is employed to observe device functionality (especially beyond the specified behavior) when the device is connected to other Apple devices and devices from other vendors. This provides early indications of proprietary vendor code and supports subsequent reverse engineering.

Passive monitoring of USB Power Delivery communications only covers the use of a subset of the protocol and optional vendor-specific mes-

sages. Therefore, the USB Power Delivery chip firmware must be extracted from the selected Apple device (2 in Figure III.3) and static reverse engineering techniques employed to disassemble the firmware and reproduce the pseudo code via decompilation.

Next, dynamic reverse engineering techniques and trial-and-error probing of the running device with actual messages are performed (3 in Figure III.3). These augment the static reverse engineering efforts to provide a better understanding of the firmware. After important components of the firmware, especially undocumented vendor-specific functionality, are understood, attempts are made to implement and simulate the components to verify that the Apple device behaves and responds according to the reverse engineering results. A jailbreaking solution, `checkra1n` [26], is employed to facilitate on-device experiments. This provides root access to the test devices. Communications are performed over the normal USB interface (4 in Figure III.3). Dynamic reverse engineering efforts and simulation of USB Power Delivery communications are also useful when conducting injection tests to identify vulnerabilities.

III.4 Results

This section presents the results of applying the research methodology discussed above to USB Power Delivery implementations in Apple iPhone models.

III.4.1 Information Gathering

The USB Power Delivery hardware in iPhone X, iPhone 8 and iPhone 8 Plus models appears to employ a Cypress CYPD2104 embedded microcontroller [27]. General datasheets and hardware design guides are available at the vendor site [28]. While the documentation provides useful insights, it was not possible to obtain complete documentation for the hardware, which would have provided useful information about the memory mapping of peripherals.

A Cypress CYPD2104 embedded microcontroller has a 48 MHz ARM Cortex-M0 CPU with 32 KB of flash storage for firmware, 4 KB SROM for booting and configuration, and 4 KB of SRAM. The I/O subsystem includes two serial communications blocks supporting I2C, SPI and UART, as well as several GPIOs. The interfaces are used to communicate with other peripherals such as the Apple system-on-chip. Another feature very relevant to reverse engineering and vulnerability discovery is Serial Wire Debug (SWD) access. The access could be over a JTAG interface that would en-

able on-device debugging capabilities, very useful for reverse engineering and vulnerability discovery. However, no attempts were made to access the interface in this research.

III.4.2 Passive Monitoring

USB Power Delivery is not enabled on all Apple devices. Tests on Apple iPhones suggest that it is supported by Apple iPhone 8 and later models. Using a commercial USB Power Delivery analyzer [29] on a USB Power Delivery enabled power supply connected to an iPhone reveals if the device supports USB Power Delivery. A supported device also reveals its specification revision (and thus the supported messages). An attempt by a source to negotiate an explicit contract with an unsupported device fails to elicit a response.

Table III.3: Explicit Contract between source (non-Apple) power supply (Rev. 3.0) and sink iPhone X (iOS 13.2.2).

Spec	Index	Time	Role	Message	Data
v3.0	115	0:35.294.997	Source:DFP	[0]Source_Cap	A1 61 2C 91 01 0A 2C D1 02 00 F4 21 03 00 F4 C1 03 00 B1 B1 04 00 45 41 06 00 83 B5 F1 BC
	124	0:35.296.426	Sink:UFP	[0]GoodCRC	41 00 BB 6C BB A8
v2.0	127	0:35.297.707	Sink:UFP	[0]Request	42 10 2C B1 04 13 3D 9D 18 5D
	131	0:35.298.419	Source:DFP	[0]GoodCRC	61 01 8F 78 38 4A
v2.0	134	0:35.301.522	Source:DFP	[1]Accept	63 03 21 7B 00 96
	137	0:35.302.329	Sink:UFP	[1]GoodCRC	41 02 97 0D B5 46
v2.0	140	0:35.412.687	Source:DFP	[2]PS_RDY	66 05 51 2A 14 02
	143	0:35.413.232	Sink:UFP	[2]GoodCRC	41 04 A2 A8 D6 AF

Table III.3 shows a summary of the messages exchanged during an explicit contract between a source non-Apple power supply (Revision 3.0) and sink iPhone 10,6 (iOS 13.2.2; `iPhone10,3,iPhone10,6 13.2.2 17B102 Restore.ipsw` with SHA1 `9c50018b2ac7c2e3d667aa065aeda3a7ff80a4ef`. Table III.4 shows a summary of the messages exchanged during an explicit contract between a source Apple power supply (Revision 2.0) and sink iPhone 10,6. Note that the GoodCRC messages are removed.

The non-Apple power supply supports Revision 3.0 (Index 115 in Table III.3) and the test iPhone responds with Revision 2.0 (Index 127 in Table III.3). This identifies the latest revision supported by the test device.

Connecting an Apple power supply reveals additional, vendor-specific communications (Table III.4). Note that the communications are summarized and the GoodCRC messages are removed. The undocumented Apple device communications start with the first unstructured VDM with Index 299. This is accordance with the USB Power Delivery specifications, which state that proprietary communications should use unstruc-

Table III.4: Explicit contract between source Apple power supply (Rev. 2.0) and sink iPhone 10,6 (iOS 13.2.2)

Spec	Index	Time	Role	Message	Data
v2.0	182	0:21.801.787	Source:DFP	[3]Source_Cap	61 17 F0 90 01 08 EA 21 1F CC
v2.0	189	0:21.803.817	Sink:UFP	[0]Request	42 10 F0 C0 03 13 BC 0F E8 2B
v2.0	197	0:21.805.451	Source:DFP	[4]Accept	63 09 3F 92 D5 76
v2.0	203	0:21.834.345	Source:DFP	[5]PS_RDY	66 0B 56 07 AC E5
v2.0	238	0:24.825.555	Source:DFP	[1]VDM:DiscIdentity	6F 13 01 80 00 FF 16 62 AB 1B
v2.0	245	0:24.827.511	Sink:UFP	[2]VDM:DiscIdentity	4F 44 41 80 00 FF AC 05 00 54 00 00 00 00 00 21 7D 16 94 99 07 82
v2.0	255	0:24.831.372	Source:DFP	[2]VDM:DiscSVID	6F 15 02 80 00 FF 58 38 5E 86
v2.0	262	0:24.833.320	Sink:UFP	[3]VDM:DiscSVID	4F 26 42 80 00 FF 00 00 AC 05 F6 20 C2 26
v2.0	270	0:24.837.322	Source:DFP	[3]VDM:DiscMode	6F 17 03 80 AC 05 BA E4 F8 1B
v2.0	277	0:24.839.154	Sink:UFP	[4]VDM:DiscMode	4F 28 43 80 AC 05 02 00 00 00 72 AD 21 96
v2.0	285	0:24.844.365	Source:DFP	[4]VDM:EnterMode	6F 19 04 81 AC 05 55 08 DD 38
v2.0	292	0:24.846.477	Sink:UFP	[5]VDM:EnterMode	4F 1A 44 81 AC 05 8E 2F C5 E3
v2.0	299	0:24.850.260	Source:DFP	[5]VDM:Unstructured	6F 1B 05 00 AC 05 E7 4D 56 1A
v2.0	307	0:24.851.919	Sink:UFP	[6]VDM:Unstructured	4F 1C 15 00 AC 05 5E C3 C3 FF
v2.0	315	0:24.853.357	Sink:UFP	[7]VDM:Attention	4F 3E 06 81 AC 05 02 01 AC 05 00 00 00 00 DC 69 C4 D9
v2.0	324	0:24.856.927	Source:DFP	[6]VDM:Unstructured	6F 3D 02 01 AC 05 00 00 00 00 06 00 00 20 12 5E E4 81
v2.0	333	0:24.858.774	Sink:UFP	[0]VDM:Unstructured	4F 10 12 00 AC 05 E6 16 E4 A7
v2.0	340	0:24.860.209	Sink:UFP	[1]VDM:Attention	4F 32 06 81 AC 05 02 01 AC 05 04 00 00 00 70 47 1C CC
v2.0	349	0:24.863.748	Source:DFP	[7]VDM:Unstructured	6F 3F 02 01 AC 05 04 00 00 00 00 02 08 00 D1 C4 AA B0
v2.0	358	0:24.865.628	Sink:UFP	[2]VDM:Unstructured	4F 14 12 00 AC 05 26 B0 64 52

tured VDMs. The communications are initiated when the Apple power supply asks the iPhone for its device ID. The iPhone device responds with an Apple VID `0x05ac`, following which the Apple power source initializes and starts the Apple-device-specific protocol. This protocol is dissected in Section III.4.5.

III.4.3 Firmware Files

The USB Power Delivery firmware files were located in iOS updates [30]. These files are regularly released by Apple to update the iOS operating system and support on-board peripherals. The firmware files reside in an unpacked .ipsw file (directories: `048-90011-109/YukonB17B102.arm64CustomerRamDisk/usr/standalone/firmware/` or `048-90336-109/YukonB17B102.arm64CustomerRamDisk/usr/standalone/firmware/`) and are named `USB-C_HPM,x.bin`, where `x` varies based on the number of included firmware files.

The firmware files come in various versions for installation on device hardware (Section III.4.1). Many of the files are equal in size and have minor differences in their binaries. An important difference is that each has a different product ID (PID) that is reported by the corresponding firmware in response to a structured Discover Identity VDM (Table III.2).

Table III.5: Firmware files with their PIDs and test iPhone models

Filename	*sha1sum*	*PID*	*iPhone model*
USB-C_HPM,1.bin	83D9F3003DF9CC1915507BD090608AE0AA96CF5D	0x1654	–
USB-C_HPM,2.bin	87BF3EEDA8C98081657F13B5B547924893EF0ED3	0x165d	–
USB-C_HPM,3.bin	0314144681992F7A4FE8B0F69A7AB42CA159E76D	0x166c	iPhone 8
USB-C_HPM,4.bin	273A80375FE8FEC09D498221BE472958B818582F	0x167c	iPhone 8 Plus
USB-C_HPM,5.bin	ACC3DBE69E310E1FE3063725F5B436E807C83D94	0x167d	iPhone X
USB-C_HPM,6.bin	B7CE922CD8B3D0018E4861006A065C7C5FBD9D5B	0x1686	–
USB-C_HPM,7.bin	916D096C8939F4E108A269A854198B95BD5A7BEE	0x1687	–
USB-C_HPM,8.bin	4FC3EB5B1B0244C04F7D6BB6A917ACAAE9F7D56F	0x1688	–

Table III.5 lists the firmware files with their PIDs. The PID enables any connected device to identify the iPhone model via the USB Power Delivery protocol. An important observation is that the firmware files have approximately identical code across all the PIDs (and thus iPhone models) for a given iOS version. This means that any security issues discovered in firmware for a specific iPhone model would likely be present in multiple models, increasing the applicability of a forensic data acquisition method. Reverse engineering results for one device model could be reused across models, saving time and resources.

Research revealed that the different iOS 13.2.2 updates for iPhone 8, iPhone 8 Plus and iPhone X models included identical `USB-C_HPM,x.bin` files as shown in Table III.5.

The common firmware codebase also supports binary diffing. Security patches discovered in two versions of a `USB-C_HPM,x.bin` file can be assumed to be present in the firmware of different iPhone models. This greatly reduces the resources needed to discover potential security patches across device models.

For a given `USB-C_HPM,x.bin` file corresponding to an iPhone 8 Plus model (`USB-C_HPM,4.bin`), different iOS updates can be downloaded and analyzed to detect changes to the USB Power Delivery firmware. Comparing the sha1sum values for differences is adequate to indicate a patch because there does not appear to be any iOS-specific rebuilding or versioning changes embedded in the firmware, leaving it untouched between updates unless the actual USB Power Delivery firmware is updated.

Table III.6 shows several iOS updates for the iPhone 8 Plus model and the corresponding `sha1sum(USB-C_HPM,4.bin)` values. The trend is that the USB Power Delivery firmware is updated rarely. In fact, one patch was retained in iOS versions 13.3.1 to 13.4.

Table III.6: `USB-C_HPM,4.bin` in various iOS versions

iOS	*Filename*	*sha1sum(USB-C_HPM,4.bin)*
13.4	iPhone_5.5_P3_13.4_17E255_Restore.ipsw	9767A86F62ABDC8C1046F4D807CC30DAB99A4693
13.3.1	iPhone_5.5_P3_13.3.1_17D50_Restore.ipsw	273A80375FE8FEC09D498221BE472958B818582F
13.3	iPhone_5.5_P3_13.3_17C54_Restore.ipsw	273A80375FE8FEC09D498221BE472958B818582F
13.2.3	iPhone_5.5_P3_13.2.3_17B111_Restore.ipsw	273A80375FE8FEC09D498221BE472958B818582F
13.2.2	iPhone_5.5_P3_13.2.2_17B102_Restore.ipsw	273A80375FE8FEC09D498221BE472958B818582F
13.1.3	iPhone_5.5_P3_13.1.3_17A878_Restore	273A80375FE8FEC09D498221BE472958B818582F
12.4.1	iPhone_5.5_P3_12.4.1_16G102_Restore	79CB8220D2C6F5917C1C11ED7B4BF733E3C9B1C8
12.3	iPhone_5.5_P3_12.3_16F156_Restore.ipsw	79CB8220D2C6F5917C1C11ED7B4BF733E3C9B1C8
12.2	iPhone_5.5_P3_12.2_16E227_Restore.ipsw	79CB8220D2C6F5917C1C11ED7B4BF733E3C9B1C8
12.0	iPhone_5.5_P3_12.0_16A366_Restore.ipsw	B374072044A97669A688A49E1723C55E9973A851
11.4.1	iPhone_5.5_P3_11.0_11.4.1_15G77_Restore.ipsw	1E20D8B4D54D6C092DA9B668A53AAAE81ABFA3EE
11.0	iPhone10,5_11.0_15A372_Restore.ipsw	1E20D8B4D54D6C092DA9B668A53AAAE81ABFA3EE

Table III.7: `USB-C_HPM,4.bin` details for various iOS versions (see Table III.6)

iOS	*USB PD Rev.*	*functions*	*code bytes*	*opcodes*	*pseudo c code lines*
13.4	2.0	248	18310	8419	6247
13.2.2	2.0	249	18598	8552	6206

III.4.4 Firmware Reverse Engineering

Since most of the firmware files corresponding to different PIDs have few differences, reverse engineering can focus on just one of the `USB-C_HPM,x.bin` files in Table III.5. The machine architecture is ARM little endian and the code is in the ARM Thumb mode [31], which is a subset of the ARM instruction set that uses variable-length instructions, often for improved code density. The code is also what is often referred to as "bare metal" code, meaning it can execute without any other abstraction layer (e.g., underlying operating system). The code directly interacts with the Apple system-on-chip and other peripherals through an I/O subsystem, mapped at specific memory addresses. Without documentation about the underlying USB Power Delivery hardware, the addresses are hardware-specific and often unknown. Therefore, from a reverse engineering perspective, it is necessary to make assumptions when code uses such unknown, hard-coded (non-position independent) addresses.

Table III.7 shows the results of disassembling and decompiling the most recent versions of the firmware file `USB-C_HPM,4.bin` listed in Table III.6. The total numbers of lines of pseudo C code for the two files are slightly more than 6,200. The USB Power Delivery Revision 2.0 was previously confirmed via passive monitoring. Therefore, the code supports all the messages listed for Revision 2.0 (Table III.1). Code that implements ad-

ditional unstructured VDMs is also included. It is expected that the number of lines of code would grow significantly to support a later revision (e.g., Revision 3.0). This is because Revision 3.0 supports a large number of additional messages (Table III.1).

As described in Section III.4.2, all the Apple-specific messages were identified and reverse engineered. Therefore, all the unstructured VDMs supported by the firmware could be identified in the disassembled code and pseudo C code. In fact, all the Apple-specific unstructured VDMs are handled by the same handler function. This function processes user input and is, therefore, an attractive target for vulnerability analysis. Erroneous handling of data in any USB Power Delivery message is a potential attack vector that could lead to a compromise of the USB Power Delivery functionality.

The number of lines of pseudo C code lines is relatively small compared with larger source code trees [7]. Since the firmware is "bare metal" code, the code is less generic and more difficult to compare with other sources. Therefore, the likelihood of security vulnerabilities in the firmware is difficult to compare with other estimates. Nevertheless, the code is in a state that is amenable to the application of established security vulnerability discovery techniques [9–11].

III.4.5 Apple Vendor-Defined Protocol

The undocumented VDMs in Table III.4 indicate that a special protocol is used by Apple devices to exchange device-specific information. Two connected Apple devices engage in an explicit contract negotiation as seen in the messages with Index 182 through 203 in Table III.4. After this, the Apple-enabled power source requests the identity of the Apple iPhone X sink via a Discover Identity VDM with Index 238. Since the sink responds with a known Apple VID (`0x05ac`) and PID (`0x167d`), the two devices can engage in additional communications using messages with Index 255 through 292. The next message (Index 299) from the source to the sink is the first unstructured VDM and the first fully vendor-specific message. Upon dissecting the raw data in this message, bytes [0:2] were determined to correspond to the USB Power Delivery message header (Figure III.1), bytes `[2:6]` to the VDM header and bytes `[6:10]` to the message CRC. Further dissection of the VDM header bytes `[6:10]` (little endian) revealed a VID of `0x05ac`, VDM Type of `0` (unstructured message) and Vendor Use of `0x5` (Figure III.4). The unstructured VDM was determined to contain the expected Apple VID of `0x05ac` and an undocumented command `0x5`. For each undocumented command, a handler function can be identified in the associated firmware file `USB-C_HPM,4.bin` and disassembled.

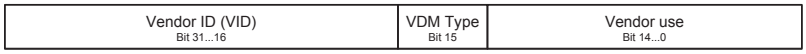


Figure III.4: Unstructured VDM header.

Further dissection of the communications in Table III.4 focused on the Attention VDM with Index 315 sent by the sink to the source (i.e., the iPhone asks the Apple power source for information). The response from the source has Index 324. This is interesting, because the iPhone only requests this type of information when it is connected to an Apple device. In fact, it turns out that the iPhone requests a range of data from the Apple power source (serial number, device name, manufacturer, etc.).

Root access to the iPhone was achieved using `checkra1n` [26]. This enabled the recovery of the data exchanged using the Apple-specific protocol. Next, the command `ioreg -f -i -l -w0 > /tmp/ioreg.txt` was used to obtain the content of the iPhone I/O Registry [32], which made it possible to interpret the exchanged data.

Table III.8 shows example data exchanged between the Apple power supply source and iPhone sink. Note that the communications are summarized and the GoodCRC messages are removed. The ASCII data `C04650505D5GW85A8` at Index 401 was located in the Apple I/O Registry [32] as the Apple power device serial number. The data can be located using the command `ioreg -f -i -l -w0 | grep C04650`, which yields `"SerialNumber"="C04650505D5GW85A8"` and `"SerialString"="C04650505D5GW85A8"`.

Table III.8: Data exchanged between source Apple power supply (Rev. 2.0) and sink iPhone 10,6 (iOS 13.2.2).

Spec	Index	time	Role	Message	Data	Ascii
v2.0	392	0:24.874.135	Sink:UFP	[5]VDM:Attention	4F 3A 06 81 AC 05 02 05 AC 05 30 00 00 00 F8 DF 19 1D	O:..... ..0.....
v2.0	401	0:24.877.949	Source:DFP	[1]VDM:Unstructured	6F 73 02 05 AC 05 30 00 00 00 43 30 34 36 35 30 35 30 35 44 35 47 57 38 35 41 38 00 00 00 55 8A 48 BE	os....0. ..C04650 505D5GW8 5A8...U. H.

By leveraging the handler functions in the firmware, it is possible to identify all the implemented vendor protocol messages and, thus, all the supported unstructured VDMs and the messages required by the USB Power Delivery protocol. Control over all the supported messages coupled with the ability to communicate with the iPhone hardware facilitates the discovery and exploitation of security vulnerabilities. These include direct code execution on the iPhone hardware and poor input validation

by peripherals/system-on-chip/kernel using the USB Power Delivery data (user input).

Table III.8 shows an example of an the Apple power supply sending its serial number. Because all the messages supported by the firmware (including undocumented VDMs) can be replicated, all the data exchanged by the undocumented protocol can be modified at will.

III.4.6 Firmware Modification and Rollback

Analysis reveals that the `USB-C_HPM,x.bin` firmware files are unsigned and are, therefore, neither verified at installation time nor at runtime. This is verified by modifying the PID in a `USB-C_HPM,5.bin` file (see Table III.5) and flashing it to the corresponding iPhone test device. With the aid of the `checkra1n` [26] jailbreaking solution, the Apple USB Power Delivery firmware flash executable `usbcfwflasher` included in the iOS firmware update file could be used to flash the modified `USB-C_HPM,5.bin` file. This can be performed on all `checkra1n`-supported Apple devices without requiring any user credentials.

The firmware modification is verified by monitoring a normal explicit contract with the additional Apple-specific VDM protocol between an Apple power supply and iPhone with the modified firmware. A successful firmware modification results in a different PID being returned from the iPhone in response to a structured Discover Identity VDM from the power supply.

Table III.9 shows that the returned PID in the message with Index 83 is `0x1337` instead of the expected PID `0x167d` in the message with Index 245 in Table III.4. Note that the communications are summarized and the GoodCRC messages are removed. The PIDs are the 16-bit little endian values at bytes `[16:18]` in both messages.

The result is that it is possible to fully modify the USB Power Delivery firmware. This includes the ability to perform a firmware rollback and install an older, potentially-vulnerable, firmware version on a patched device. Because this is a security vulnerability in itself, it is very useful for further vulnerability discover because researchers can implement any test code to expose, for example, further propagation in an iPhone or side-channel attack scenarios.

Table III.9: Discover Identity VDMs between source Apple power supply (Rev. 2.0) and sink iPhone 10,6 (iOS 13.2.2).

Spec	Index	time	Role	Message	Data
v2.0	76	0:44.204.575	Source:DFP	[7]VDM:DiscIdentity	6F 1F 01 80 00 FF 17 8F 5B DE
v2.0	83	0:44.206.298	Sink:UFP	[2]VDM:DiscIdentity	4F 44 41 80 00 FF AC 05 00 54 00 00 00 00 00 21 **37 13** 94 CA FB F8

III.5 Conclusions

The methodology for analyzing USB Power Delivery implementations facilitates the discovery of security vulnerabilities for exploiting USB Power Delivery hardware to acquire data in digital forensic investigations. The ultimate goal is to further leverage privileges within the system, potentially through a new set of security vulnerabilities that are identified using the hardware as a springboard. Examples of the new vulnerabilities include implicit trust relationships, other components in the USB Power Delivery hardware and system processes that parse data provided as inputs by Apple VDM commands. The step-by-step methodology, which is demonstrated to expose the implementation details of a USB Power Delivery device, is applicable to a wide range of USB Power Delivery implementations by diverse vendors.

The results of using the methodology on Apple iPhones can be summarized as follows. Gathering information about the underlying USB Power Delivery hardware assists firmware reverse engineering, side-channel analysis and attack development. The ability to monitor USB Power Delivery messages facilitates the analysis of messages supported by a given device and helps discern if a proprietary vendor protocol is employed. Sending and receiving arbitrary messages using a simulation tool advances black box testing, reverse engineering and exploitation.

Additionally, the reverse engineering of firmware to yield disassembled code and pseudo C code is very useful for manual and automated vulnerability analyses. Diffing tests can help reveal patches that can be checked for potential security vulnerabilities. Rollbacks of vulnerable firmware can be accomplished on jailbroken devices without requiring user credentials because firmware signatures and rollback protection mechanisms are not implemented. The lack of signatures also facilitates arbitrary modifications of firmware that expose USB Power Delivery to evil maid attacks.

Future research will attempt to discover additional vulnerabilities. It will also attempt to simulate and instrument/debug the extracted firmware, with the goal of advancing fuzzing techniques for vulnerability discovery. Other avenues of future research include debugging test devices and chips via JTAG and conducting simulation via emulation and symbolic execution [11, 33, 34].

Acknowledgements

This research was supported by the IKTPLUSS Program of the Norwegian Research Council under R&D Project Ars Forensica Grant Agreement 248094/O70. Apple was notified about this research in advance of publication.

III.6 References

[1] G. Alendal, S. Axelsson and G. O. Dyrkolbotn, 'Leveraging the USB Power Delivery Implementations for Digital Forensic Acquisition,' in *Advances in Digital Forensics XVII*, G. Peterson and S. Shenoi, Eds., Springer International Publishing, 2021, pp. 111–133, ISBN: 978-3-030-88381-2.

[2] Z. Wang and A. Stavrou, 'Exploiting smart-phone usb connectivity for fun and profit,' in *Proceedings of the 26th Annual Computer Security Applications Conference*, ser. ACSAC '10, Austin, Texas, USA: Association for Computing Machinery, 2010, pp. 357–366, ISBN: 9781450301336. DOI: 10.1145/1920261.1920314. [Online]. Available: https://doi.org/10.1145/1920261.1920314.

[3] J. Tian, N. Scaife, D. Kumar, M. Bailey, A. Bates and K. Butler, 'Sok:" plug & pray" today–understanding usb insecurity in versions 1 through c,' in *2018 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2018, pp. 1032–1047.

[4] USB-IF, *Universal serial bus type-c cable and connector specification*, https://www.usb.org/sites/default/files/USBType-CSpecR2.0-August2019.pdf, [Online; accessed 20-April-2020], 2020.

[5] USB-IF, *Usb power delivery*, https://www.usb.org/document-library/usb-power-delivery, [Online; accessed 20-March-2020], 2020.

[6] L. Hatton, 'Reexamining the fault density-component size connection,' *IEEE Softw.*, vol. 14, no. 2, pp. 89–97, Mar. 1997, ISSN: 0740-7459. DOI: 10.1109/52.582978. [Online]. Available: https://doi.org/10.1109/52.582978.

[7] A. Ozment and S. E. Schechter, 'Milk or wine: Does software security improve with age?' In *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, ser. USENIX-SS'06, Vancouver, B.C., Canada: USENIX Association, 2006.

[8] A. Tereshkin, 'Evil maid goes after pgp whole disk encryption,' in *Proceedings of the 3rd International Conference on Security of Information and Networks*, ser. SIN '10, Taganrog, Rostov-on-Don, Russian Federation: Association for Computing Machinery, 2010, p. 2, ISBN: 9781450302340.

DOI: 10.1145/1854099.1854103. [Online]. Available: https://doi.org/10.1145/1854099.1854103.

[9] A. Austin and L. Williams, 'One technique is not enough: A comparison of vulnerability discovery techniques,' in *2011 International Symposium on Empirical Software Engineering and Measurement*, 2011, pp. 97–106.

[10] B. Liu, L. Shi, Z. Cai and M. Li, 'Software vulnerability discovery techniques: A survey,' in *2012 Fourth International Conference on Multimedia Information Networking and Security*, 2012, pp. 152–156.

[11] N. Stephens, J. Grosen, C. Salls, A. Dutcher, R. Wang, J. Corbetta, Y. Shoshitaishvili, C. Kruegel and G. Vigna, 'Driller: Augmenting fuzzing through selective symbolic execution,' in *23nd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*, 2016. [Online]. Available: http://www.internetsociety.org/sites/default/files/blogs-media/driller-augmenting-fuzzing-through-selective-symbolic-execution.pdf.

[12] G. Alendal, S. Axelsson and G. O. Dyrkolbotn, 'Exploiting Vendor-Defined Messages in the USB Power Delivery Protocol,' in *Advances in Digital Forensics XV*, G. Peterson and S. Shenoi, Eds., Cham: Springer International Publishing, 2019, pp. 101–118, ISBN: 978-3-030-28752-8.

[13] F. Yang and S. Manoharan, 'A security analysis of the oauth protocol,' in *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, 2013, pp. 271–276.

[14] A. Sosnovich, O. Grumberg and G. Nakibly, 'Finding security vulnerabilities in a network protocol using parameterized systems,' in *Computer Aided Verification*, N. Sharygina and H. Veith, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 724–739, ISBN: 978-3-642-39799-8.

[15] W. Chen and J. Bhadra, 'Striking a balance between soc security and debug requirements,' in *2016 29th IEEE International System-on-Chip Conference (SOCC)*, 2016, pp. 368–373.

[16] G. Beniamini, *Over the air: Exploiting broadcom's wi-fi stack (part 2)*, [Online; accessed 20-April-2020], 2017. [Online]. Available: https://google-projectzero.blogspot.com/2017/04/over-air-exploiting-broadcoms-wi-fi_11.html.

[17] Y. Duan, X. Li, J. Wang and H. Yin, 'Deepbindiff: Learning program-wide code representations for binary diffing,' Jan. 2020. DOI: 10.14722/ndss.2020.24311.

[18] Apple, *Dual orientation connector with external contacts and conductive frame*, U.S. Patent, 13th May 2013.

[19] USB-IF, *Getting a vendor id*, https://www.usb.org/getting-vendor-id, [Online; accessed 30-January-2020], 2020.

[20] S. J. Gowdy, *The usb id repository*, [Online; accessed 17-May-2020], 2020. [Online]. Available: http://www.linux-usb.org/usb-ids.html.

[21] A. a. Maurushat, *Disclosure of Security Vulnerabilities Legal and Ethical Issues /*, 1st ed. 2013. London : Springer London : 2013, ch. 3.3, ISBN: 1-4471-5003-1.

[22] A. Harper, S. Harris, J. Ness, C. Eagle, G. Lenkey and T. Williams, *Gray Hat Hacking The Ethical Hackers Handbook*, 3rd. McGraw-Hill Osborne Media, 2011, ch. 20-22, ISBN: 0071742557.

[23] G. Chen, Z. Qi, S. Huang, K. Ni, Y. Zheng, W. Binder and H. Guan, 'A refined decompiler to generate c code with high readability,' *Software: Practice and Experience*, vol. 43, no. 11, pp. 1337–1358, 2013. DOI: 10.1002/spe.2138. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2138. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2138.

[24] M. Pietrek, 'Peering inside the pe: A tour of the win32 portable executable file format,' 1994.

[25] Y. Li and J. Yan, 'Elf-based computer virus prevention technologies,' in *Information Computing and Applications*, C. Liu, J. Chang and A. Yang, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 621–628, ISBN: 978-3-642-27452-7.

[26] A. Panhuyzen, *Checkra1n*, [Online; accessed 12-March-2020], 2020. [Online]. Available: https://https://checkra.in/.

[27] T. Inc., *Apple iphone x teardown*, [Online; accessed 12-May-2020], 2017. [Online]. Available: https://www.techinsights.com/blog/apple-iphone-x-teardown.

[28] C. S. Corporation, *Cypd2104-20fnxit*, [Online; accessed 12-May-2020], 2018. [Online]. Available: https://www.cypress.com/part/cypd2104-20fnxit.

[29] T. Phase, *Usb power delivery analyzer*, [Online; accessed 12-March-2020], 2020. [Online]. Available: https://www.totalphase.com/products/usb-power-delivery-analyzer/.

[30] Apple, *About ios 13 updates*, [Online; accessed 17-March-2020], 2020. [Online]. Available: https://support.apple.com/en-us/HT210393.

[31] A. Limited, *The thumb instruction set*, [Online; accessed 24-March-2020], 2020. [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0210c/CACBCAAE.html.

[32] Apple, *The i/o registry*, [Online; accessed 12-March-2020], 2020. [Online]. Available: https://developer.apple.com/library/archive/documentation/DeviceDrivers/Conceptual/IOKitFundamentals/TheRegistry/TheRegistry.html.

[33] E. J. Schwartz, T. Avgerinos and D. Brumley, 'All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask),' in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, ser. SP '10, Washington, DC, USA: IEEE Computer Society, 2010, pp. 317–331, ISBN: 978-0-7695-4035-1. DOI: 10.1109/SP.2010.26. [Online]. Available: http://dx.doi.org/10.1109/SP.2010.26.

[34] D. Engler and D. Dunbar, 'Under-constrained execution: Making automatic code destruction easy and scalable,' in *Proceedings of the 2007 International Symposium on Software Testing and Analysis*, ser. ISSTA '07, London, United Kingdom: ACM, 2007, pp. 1–4, ISBN: 978-1-59593-734-6. DOI: 10.1145/1273463.1273464. [Online]. Available: http://doi.acm.org/10.1145/1273463.1273464.

Paper IV

Digital Forensic Acquisition Kill Chain — Analysis and Demonstration

G. Alendal, G. O. Dyrkolbotn and S. Axelsson, 'Digital Forensic Acquisition Kill Chain - Analysis and Demonstration,' in *Advances in Digital Forensics XVII*, G. Peterson and S. Shenoi, Eds., Springer International Publishing, 2021, pp. 3–19, ISBN: 978-3-030-88381-2

Abstract

The increasing complexity and security of consumer products pose major challenges to digital forensics. Gaining access to encrypted user data without user credentials is a very difficult task. Such a situation may require law enforcement to leverage offensive techniques – such as vulnerability exploitation – to bypass security measures in order to retrieve data in digital forensic investigations.

This chapter proposes a digital forensic acquisition kill chain to assist law enforcement in acquiring forensic data using offensive techniques. The concept is discussed and examples are provided to illustrate the various kill chain phases. The anticipated results of applying the kill chain include improvements in performance and success rates in short-term, case-motivated, digital forensic acquisition scenarios as well as long-term, case-independent planning and research efforts focused on identifying vulnerabilities and leveraging them in digital forensic acquisition methods and tools.

IV.1 Introduction

Several digital forensic process models have been proposed in the literature [2]. Regardless, a generic digital forensic process can be viewed as comprising four phases: seizure, acquisition, analysis and reporting. The digital forensic acquisition phase covers the retrieval of digital forensic data from seized devices and other data sources. Its main goal is to gain access to data for forensic analysis. Clearly, digital forensic acquisition tasks are changing as technology advances, but the overall goal is the same – accessing data in a forensically-sound manner [3, 4].

Embedded devices and online services are important sources of digital data in criminal cases, which makes digital forensic acquisition a priority for law enforcement. In recent years, smartphone vendors such as Apple and Samsung have instituted mechanisms for securing user data. Data in their devices is often encrypted and secured against a variety of attacks, local as well as remote. Gaining access to encrypted user data without user credentials is a very difficult task.

Garfinkel et al. [5] mention encryption as posing major challenges to law enforcement as they conduct digital forensic investigations. Arshad et al. [6] discuss the impacts of mandatory encryption and increased focus on privacy on the effectiveness of digital forensics. Balogun et al. [7] estimate that encryption alone prevents the recovery of digital forensic data in as much as sixty percent of cases that involve full disk encryption. In the FBI-Apple encryption dispute of 2015-16, Apple denied the FBI's request to create special firmware that would enable the recovery of user credentials from an iPhone 5C seized in a terrorist investigation [6]. Apple considered product security and user privacy to be more important than supporting the terrorism investigation.

Since law enforcement cannot rely on assistance from vendors to bypass security mechanisms in their products, the best option is to leverage offensive techniques to retrieve protected data in digital forensic investigations. Specifically, it is necessary to apply sophisticated techniques to discover published (n-day) and unpublished (0-day) vulnerabilities in the targets, and exploit them to acquire forensic data.

The idea of law enforcement leveraging published vulnerabilities is a concern because law enforcement assumes the role of an attacker in order to pursue justice. However, discovering and holding on to undocumented vulnerabilities in order to bypass security mechanisms are even more concerning. New vulnerabilities should be promptly reported to the affected vendors to enable them to mitigate risks, but this would prevent the continued use of the vulnerabilities. The conflicting interests between offensive and defensive uses of security vulnerabilities are not new. Indeed, they

have been discussed publicly [8] and addressed by the U.S. Government [9]. Whether to restrict discovered vulnerabilities for offensive use or disclose them for defensive purposes is determined by a vulnerability equities process, where U.S. agency representatives gather to evaluate and decide the fate of new vulnerabilities discovered by government agencies [9]. This policy is understandably controversial [10, 11].

This research does not take a stand on the vulnerability equities dilemma. Rather, it seeks to inform law enforcement about the possibility of discovering vulnerabilities in electronic devices and leveraging them to acquire forensically-sound data in criminal investigations. It focuses on a methodical approach called the "digital forensic acquisition kill chain," which is based on the "intrusion kill chain" concept used in computer network defense [12]. The intrusion kill chain is a systematic process for targeting and engaging an adversary to achieve the desired security effects [12]. The digital forensic acquisition kill chain turns this around – it is a systematic process for law enforcement (acting as an adversary) to target electronic devices using offensive techniques to facilitate digital forensic acquisition.

Law enforcement has some advantages when developing and employing offensive techniques. These include access to resources as well as police authority (ability to seize devices). Unlike attackers, law enforcement may have the time to execute offensive actions and impose patch prevention. A seized device may be fully patched with no known vulnerabilities at the time of seizure. However, the same device becomes vulnerable in the future as n-day vulnerabilities are published and 0-day vulnerabilities are discovered. Since law enforcement can prevent seized devices from receiving updates, it can leverage both types of vulnerabilities in digital forensic acquisition.

IV.2 Related Work

Several digital forensic process models that focus on practitioners and the use of digital evidence in court have been proposed. The Advanced Data Acquisition Model [13] addresses the needs of practitioners and the expectations of courts for formal descriptions of the processes undertaken to acquire digital evidence. Montasari [14] has proposed a standardized model that enables digital forensic practitioners to follow a generic approach that can be applied to incident response as well as criminal and corporate investigations. In an attempt to further address the need for a generic digital forensic investigation process for use in the three domains, Montasari et al. [15] have proposed the Standardized Digital Forensic In-

vestigation Process Model that draws on existing models and renders them generic enough for wide applicability. However, although digital forensic investigative processes are discussed, neither the scope nor the details of key processes such as examination and analysis are provided.

The three models address the need for trustworthy and court-accepted methods and processes. The focus is on ensuring the reliability of digital evidence presented in court using formal, standardized processes. In contrast, the digital forensic model presented in this chapter differs substantially from the three models in that it concentrates on using offensive techniques for digital forensic acquisition. However, the proposed model will have to be augmented in the future to guide the development of trustworthy, court-accepted methods.

IV.3 Digital Forensic Acquisition Kill Chain

The primary goal of the proposed digital forensic acquisition kill chain is to articulate a structured process for developing new digital forensic acquisition methods based on offensive techniques. It is intended to improve performance and success rates during the time-constrained, case-motivated development of digital forensic acquisition methods as well as the long-term case-independent development of digital forensic acquisition methods that take into account trends in consumer adoption of technology.

IV.3.1 Background

Hutchins et al. [12] have specified a kill chain model that describes the network intrusion phases employed by advanced adversaries, often referred to as advanced persistent threats. Engaging a model that describes adversarial intrusion phases to inform defensive postures reduces the likelihood of success on the part of the attackers. Specifically, detecting patterns that are signs of a campaign supports proactive computer network defense. This is referred to as intelligence-driven computer network defense, where identifying intrusion patterns facilitates responses before compromise occurs. The kill chain phases specify the goals and content as an adversary goes from intelligence gathering on a potential target to achieving full compromise and the ultimate goal of penetrating the target (e.g., exfiltrating sensitive data). Such a model is required because advanced adversaries invest considerable intellectual and technical resources to penetrate high value targets. The kill chain paradigm has proven to be very valuable, and several new ideas and models have been proposed [16–20].

The intrusion kill chain of Hutchins et al. [12] is motivated by the U.S.

military targeting doctrine that encompasses six phases: find, fix, track, target, engage and assess. They adapted the targeting doctrine to computer network intrusions by introducing new phases. The resulting kill chain phases are: reconnaissance, weaponization, delivery, exploitation, installation, command and control, and actions on objectives. This methodical way of describing the expected adversarial phases views computer network defense from the adversaries' perspectives, facilitating detection by predicting the subsequent phases and the ability to execute proactive defensive operations. The research described in this chapter adapts the intrusion kill chain to facilitate offensive actions in digital forensic acquisition scenarios.

IV.3.2 Kill Chain Overview

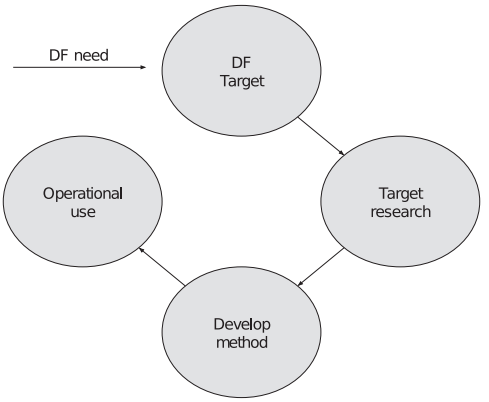


Figure IV.1: Generic digital forensic acquisition needs.

Figure IV.1 shows a simplified view of digital forensic acquisition using offensive techniques. The proposed digital forensic acquisition kill chain adapts the original kill chain to specify a methodology for using offensive techniques in digital forensic acquisition, where law enforcement assumes the role of the adversary and seized devices (evidence containers) are the targets. It brings an intelligence-driven perspective to applying forensic data acquisition methods as well as researching and developing new methods.

Figure IV.2 shows the nine phases of the proposed digital forensic acquisition kill chain. The phases are: reconnaissance, identification, surveillance and vulnerability research, weaponization, delivery, exploitation, installation, command and control, and actions on objectives.

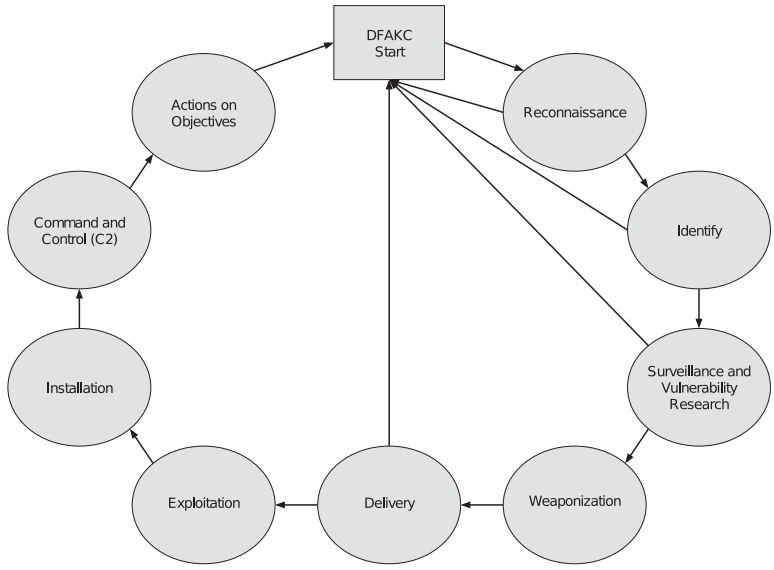


Figure IV.2: Digital forensic acquisition kill chain phases.

The nine phases are grouped and generalized according to the digital forensic acquisition needs in Figure IV.1. The initial reconnaissance phase considers the target of digital forensic acquisition. The next two phases, identification, and surveillance and vulnerability research, focus on the discovery of possible digital forensic acquisition solutions (vulnerabilities). The weaponization and delivery phases cover the development and realization of the discovered vulnerabilities. The last four phases, exploitation, installation, command and control, and actions on objectives, deal with operational issues.

A digital forensic acquisition kill chain is spawned in two general scenarios:

- **Case-Motivated Scenario:** This scenario is driven by a case-motivated need for a digital forensic acquisition method targeting a specific entity (e.g., device or service). Because digital forensic investigations are event-driven, law enforcement may not have applicable methods or be able to predict applicable methods for all possible scenarios. The kill chain focuses on solving the concrete challenge of acquiring forensically-sound data from the device or service, but it may spawn new kill chains to solve the sub-challenges that materialize. Several kill chains could be spawned in parallel and resources moved back and forth between them as the case foci and priorities

change. The overall goal of a case-motivated kill chain is to apply digital forensic acquisition to a specific device or service.

- **Case-Independent Scenario:** This scenario is driven by a case-independent, intelligence-driven need for a digital forensic acquisition method that addresses a class of challenges. As the results of several case-motivated kill chains are obtained, a trend in the challenges encountered, such as the encryption of user data, could spawn its own kill chain. A challenge in another kill chain phase, say exploitation, could spawn a separate kill chain that focuses entirely on the challenges encountered during exploitation. A challenge related to a class of devices (e.g., from a specific vendor) could spawn a vendor-specific kill chain. The vendor could be Apple or Samsung, and the targets could be smartphones, services or components such as processors and flash memory chips that are common to vendor products or services. The overall goal of a case-independent kill chain is to improve the performance of subsequent case-motivated kill chains by leveraging intelligence, knowledge, methods and tools.

Upon considering the general digital forensic acquisition needs in Figure IV.1, the completion of the reconnaissance, identification, surveillance and vulnerability research, or delivery phases could result in the kill chain being terminated. For example, as shown in Figure IV.2, a kill chain covering a trending device would terminate at the end of the delivery phase because no operational needs exist. Of course, the completion of a phase could initiate the next phase, or the phase could spawn a new kill chain.

The initial phases of reconnaissance and identification could be performed at the start of an investigation to set the direction of the investigation and prioritize resources. An initial kill chain could spawn several new (sub) kill chains that address specific devices and services. This would, of course, depend on the amount of resources available. Prioritization and resource management of the sub kill chains would be a continuous process as the investigation proceeds.

Kill chains can also be applied to trending challenges that are detached from concrete investigations. This is motivated by the fact that many current digital forensic acquisition challenges are too complex to be solved given the limited time and resources available in investigations. The kill chains would focus on longer term challenges that need dedicated resources and prioritization. The available resources would be put to best use at all times, even in the case of parallel kill chains where resources would be shifted between kill chains as priorities change and commonalities are discovered. The expected results are increased knowledge of trending challenges, increased security expertise and new digital forensic

acquisition methods.

IV.3.3 Kill Chain Phases

This section discusses the nine phases of the digital forensic kill chain in detail.

Reconnaissance

The reconnaissance phase focuses on the collection of information that would support the selection and prioritization of devices and services. This phase should be kept short if it is used as part of a specific case, where it would concentrate on selection and prioritization, and the estimation of the likelihood of success of a digital forensic acquisition method. In a case-independent scenario, the reconnaissance phase is more openly defined and may choose to focus on any target device or service of interest.

Multiple kill chains are expected to be initiated and terminated during the reconnaissance phase. Also, a single kill chain may spawn several kill chains for the identified devices and services. The basic idea is that the reconnaissance phase is based on the available information and information that is obtained easily.

Identification

The challenge to developing a new forensic acquisition method is approached in a bottom-up manner. The focus is on identifying forensic data of value and the layers of security features that may prevent its access (e.g., encryption could be the first layer to bypass).

Volatility of forensic data is always an issue. Embedded devices often keep log files and unencrypted app data in random access memory only. Thus, the digital forensic acquisition method must take into account the fact that a device cannot be power cycled. Addressing this challenge follows a different path in the remaining phases and would require a separate kill chain.

Note that two challenges – encryption and volatility – have been identified during this phase. Thus, two kill chains would be created and resource allocation decisions have to be made to best address the challenges.

Surveillance and Vulnerability Research

During the surveillance and vulnerability research phase, existing vulnerabilities, techniques, tools and services are investigated. Also, resources

are allocated to discover new vulnerabilities. Conducting activities in parallel can be efficient with regard to time. However, in order to optimize resources and not reinvent existing vulnerabilities and methods, the following two sub-phases are recommended:

- **Sub-Phase 1:** This short intelligence sub-phase focuses on gathering information about the identified challenges from open and closed sources. The goal is to discover published vulnerabilities that are potential candidates for direct use or are avenues for new vulnerability research. The sub-phase should not focus on the resource-intensive task of rediscovering low-level details about potential vulnerabilities, but only collect and prioritize potential vulnerabilities based on the available information.
- **Sub-Phase 2:** This sub-phase focuses on the active search for tools, services and vulnerabilities to address the identified challenges. It would also include a separate vulnerability research effort to discover new vulnerabilities.

The surveillance and vulnerability research phase is divided into two sub-phases in order to have a lightweight first sub-phase with a short time frame and low human resource needs. The results provide a basis for allocating resources to the much more intensive second sub-phase.

The second sub-phase has the most uncertainty with regard to resource needs and likelihood of attaining the end goal of a digital forensic acquisition method. However, in the event of success, a method that leverages a new vulnerability would have a longer life span than a method based on a published vulnerability. As multiple kill chains would be executed simultaneously during this sub-phase, efficient management of resources is required.

An example of a new kill chain is the discovery of new vulnerabilities and the acquisition of knowledge about existing vulnerabilities. Information about fixed vulnerabilities may be found on vendor web sites, change logs and published patches. Although the information about a patched vulnerability often lacks the detail needed to isolate and trigger the vulnerability, an experienced vulnerability researcher would be able to obtain the information in a reasonable period of time. This could be hours, days or months depending on the complexity of the technology and vulnerability. Additionally, since a vulnerability may not always be convertible to a successful exploit, it is necessary to research several vulnerabilities. Identifying and studying vulnerabilities, and developing exploits are time consuming; also, predicting the resource needs is difficult. Therefore, it is important to balance time, resources and success potential between discovering new vulnerabilities and rediscovering known vulnerabilities by studying

patches.

Weaponization

Weaponization involves the development of a working exploit from a new or existing vulnerability, a task that can be complex and potentially unrealizable. The weaponization of a vulnerability is hard to generalize, but it can be similar to the software development cycle. The steps proceed from developing a proof of concept to creating a production-quality exploit chain with quality assurance that minimizes the chance of failure when applied to digital forensic acquisition. A crucial step is to ensure that the method is forensically sound and complies with the law and established digital forensic standards [21].

Efforts in the weaponization phase also need to consider the users of the digital forensic acquisition method, especially their levels of expertise and access to specialized equipment and tools. Other considerations include ease of use, access to updates and support. Additionally, it is important to be aware that the type and sensitivity of the vulnerability may limit the number of users and cases where it can be applied.

Delivery

The delivery phase focuses on developing the channel or channels for executing the weaponized exploit. These could be physical interfaces such as USB, SPI, JTAG, UART and I2C or wireless channels such as Wi-Fi, Bluetooth and near-field communications (NFC). Even side channels that can be used to inject inputs into key components are potential delivery options.

Exploitation

During the exploitation phase, the focus is on applying the developed digital forensic acquisition method in a criminal case. Actions performed in this phase must adapt to the context of the device or service. Since the phase is operational in nature, it should consider all aspects of using the method, including device or service state, legality, special requirements, assumptions that do not hold (e.g., user credentials might be known), operational security and digital forensic principles. Special care should be taken if the exploitation is destructive (e.g., chip-off data acquisition), which would leave the device in a state where it cannot be returned to its owner after the investigation.

Installation

The installation phase is mostly concerned about the footprint required to achieve the goal of forensic data acquisition. A RAM-only installation is a good option when the goal is to acquire data from long-term storage and conform to forensically-sound principles [15]. Since a component installed on a device or service environment for forensic acquisition purposes may become a part of the acquired evidence, isolating and documenting the component and its behavior are vital in court proceedings. An alternative to installing a component is to enable device features to accomplish the same goal. For example, enabling adb and gaining root privileges on an Android smartphone would provide the required access. When executing custom code on a device, it is important that the footprint be as small as possible to reduce negative forensic impacts on volatile RAM storage. Alternatively, the available device debugging features could be leveraged.

Command and Control

In the command and control phase, control has already been gained over the execution and/or data on the target device or service. This could involve a generic interface such as a login shell with root access, arbitrary code execution or security feature (e.g., screen lock) bypass. Ideally, this phase should be detached from the earlier phases because it marks the start of the actual acquisition of digital forensic data. Activities could involve the use of special tools and commands that may not have been employed in the earlier device or service-specific exploitation and installation phases. The advantage of separating command and control from other phases is the reuse of knowledge, code and tools. A login shell with root access may apply the same tools to acquire data from diverse Android devices, but activities in the exploitation and installation phases for Android devices from different vendors could be totally different and leverage completely different vulnerabilities to reach the command and control phase.

Actions on Objectives

The last phase in the kill chain is to simply execute the final goal of performing the digital acquisition to obtain data of forensic value from the device or service.

IV.4 Case-Motivated Kill Chain Example

This section demonstrates the use of a digital forensic acquisition kill chain in a case-motivated scenario where law enforcement is interested in extracting data of forensic value from a broadband router seized at a crime scene. The data could constitute log files with network activity, including Wi-Fi logs pertaining to connected devices during a specific time period. The data could be used to gain information about the connected devices that would be identified by their MAC addresses. The device is a Zyxel router (model no. p8702n), which has a MIPS architecture and runs a uClinux-based operating system [22].

Reconnaissance

Open-source intelligence and reconnaissance activities for the Zyxel p8702n router focused on various discussion forums and on the availability of its firmware, which was eventually downloaded from a server located at `stup.telenor.net/firmwares/cpe-zyxel-p8702n`. Two firmware files, `100AAJX13D0.bin` and `100AAJX14D0.bin`, were obtained along with their `README` files.

Because change logs often contain valuable information about security patches, older files that were present on the server were also sought. The older files were downloaded from `web.archive.org`.

Thus, the reconnaissance phase yielded useful information from public forums along with publicly-available firmware files and their change logs.

Identification

Forensic data with the most value was expected to reside in the flash memory of the Zyxel p8702n router. However, like many low-end embedded devices, the Zyxel p8702n router stores much of its data, including logs, in RAM only. This means that valuable forensic data could be lost if the device were to be turned off. This discovery is important because it impacts how the device should be seized; specifically, the device should not be powered down before digital forensic acquisition. Addressing the RAM memory acquisition challenge requires a separate kill chain.

Thus, two directions have to be pursued and a decision must be made about where to focus the available resources. The RAM data was assumed to be more valuable, so the corresponding kill chain was pursued – gaining access to the Zyxel p8702n router RAM data without turning off or restarting the device.

Surveillance and Vulnerability Research

The shorter intelligence phase (sub-phase 1) sought to obtain information about acquiring RAM data, possibly by exploiting a vulnerability. In the case of the Zyxel p8702n router, a valuable source for vulnerability information was determined to be the vendor's patch reports. Because older firmware files and change logs were available, a reasonable approach was to examine the change logs for hints of security issues.

The examination revealed that firmware version `100AAJX7D0` had major security fixes. Therefore, the previous firmware version `100AAJX5D0` was inferred to have the security vulnerabilities.

The focus of the complex and resource-demanding sub-phase 2 was to rediscover the vulnerabilities patched in firmware version `100AAJX7D0`. This required firmware versions `100AAJX5D0` and `100AAJX7D0` to be unpacked and the differences between the two versions to be identified.

The analysis revealed a difference in the boot sequence, where a critical security vulnerability was exposed in the older version by a login shell on a serial console. The problem was that the login process /bin/smd had an `SIGTSTP` vulnerability – when `Ctrl-Z` was entered on the console, a `/bin/sh` shell was provided with the same credentials as the `/init` process. This enabled root access to the Zyxel p8702n router.

Thus, sub-phase 2 of the surveillance and vulnerability research phase resulted in the rediscovery of a vulnerability. However, the vulnerability still had to be triggered.

Weaponization

During the weaponization phase, it was determined that the vulnerability was not particularly difficult to exploit. The vulnerability was exploited by accessing the Zyxel p8702n router console and sending the `SIGTSTP` signal by entering `Ctrl-Z`. Thus, the goal of the weaponization phase was to discover an access method to the serial console of the Zyxel p8702n router; in this case, via the UART interface on the circuit board. The key result is that this could be done without powering off the Zyxel p8702n router.

Delivery

The delivery phase was also relatively simple. It involved sending `Ctrl-Z` over the attached serial console to the Zyxel p8702n router. The delivery was performed via the UART protocol using a standard RS232-to-USB serial converter and a `putty` terminal emulator.

Exploitation

Since the Zyxel p8702n router had to be powered on at all times, the digital forensic acquisition had to be performed without power-cycling the device. The considerations during the exploitation phase involved the ease of physical access to the device, speed of the operation (especially if it had to be covert), risk and likelihood of failure.

Important operational decisions had to made during the exploitation phase to prevent *ad hoc* decision making during the subsequent phases. Since the objective was to acquire data from RAM, any actions performed on the device (even as root) would affect the RAM (e.g., potentially overwriting valuable freed memory in RAM). Therefore, a bare minimum footprint had to be maintained.

Installation

The installation was restricted to digital forensic acquisition. Persistent access did not have to be maintained after the serial interface was detached. Therefore, no other tools were installed.

Command and Control

Root access to the Zyxel p8702n router rendered the digital forensic acquisition goal within reach. The command and control phase determined that only a few commands would be executed using on-device tools to preserve RAM content.

Actions on Objectives

At this point, all the digital forensic acquisition challenges were isolated and addressed. The final phase merely involved the final digital forensic acquisition of RAM data in the Zyxel p8702n router.

Note that the primary goal was to focus on the raw RAM in order to preserve freed memory data and structures. Since this goal was achieved, it was not necessary to pursue the lower priority goal focusing on temporary RAM-only filesystems that are common in many Linux distributions, or the even lower goal focusing on flash memory.

IV.5 Conclusions

Criminal investigations are increasingly hindered by strong security mechanisms that prevent forensically-relevant data from being acquired from

electronic devices and services. Absent technical assistance from vendors and service providers, the only option for law enforcement is to leverage offensive techniques such as vulnerability exploitation to bypass security measures and acquire evidentiary data. The notion of law enforcement becoming an attacker in order to pursue justice is controversial, but police authority and search and seizure laws and regulations may support such actions.

The digital forensic acquisition kill chain described in this chapter adapts the kill chain employed in computer network defense to articulate a systematic methodology for using offensive techniques in digital forensic acquisition, where law enforcement assumes the role of the adversary and the seized devices and services of interest (evidence containers) are the targets. Applying the digital forensic acquisition kill chain provides many benefits – improvements in performance and success rates in short-term, case-motivated, forensic data acquisition scenarios as well as in long-term case-independent, intelligence-driven planning and research scenarios focused on identifying vulnerabilities and leveraging them in the development of novel digital forensic acquisition methods and tools.

Future research will focus on validating the digital forensic acquisition kill chain. The case study described in this chapter focused on a single device. Realistic field evaluations with diverse and more complicated challenges will provide valuable guidance on adjusting the kill chain phases. At this time, a single kill chain model has been proposed for case-motivated and case-independent scenarios. These scenarios appear to pull the kill chain model in different directions. As a result, future research will focus on creating separate digital forensic acquisition kill chain models for the two types of scenarios.

Acknowledgements

This research was supported by the IKTPLUSS Program of the Norwegian Research Council under R&D Project Ars Forensica Grant Agreement 248094/O70.

IV.6 References

[1] G. Alendal, G. O. Dyrkolbotn and S. Axelsson, 'Digital Forensic Acquisition Kill Chain - Analysis and Demonstration,' in *Advances in Digital Forensics XVII*, G. Peterson and S. Shenoi, Eds., Springer International Publishing, 2021, pp. 3–19, ISBN: 978-3-030-88381-2.

[2] A. Al-Dhaqm, S. Abd Razak, R. A. Ikuesan, V. R. Kebande and K. Siddique, 'A review of mobile forensic investigation process models,' *IEEE Access*, vol. 8, pp. 173 359–173 375, 2020.

[3] E. Casey, *Digital evidence and computer crime: Forensic science, computers, and the internet*. Academic press, 2011, ch. 20.4.

[4] R. Ayers, S. Brothers and W. Jansen, 'Guidelines on mobile device forensics,' *NIST Special Publication*, vol. 1, no. 1, p. 85, 2014.

[5] S. L. Garfinkel, 'Digital forensics research: The next 10 years,' *Digit. Investig.*, vol. 7, S64–S73, Aug. 2010, ISSN: 1742-2876. DOI: 10.1016/j.diin.2010.05.009. [Online]. Available: http://dx.doi.org/10.1016/j.diin.2010.05.009.

[6] H. Arshad, A. B. Jantan and O. I. Abiodun, 'Digital forensics: Review of issues in scientific validation of digital evidence.,' *Journal of Information Processing Systems*, vol. 14, no. 2, 2018.

[7] A. M. Balogun and S. Y. Zhu, 'Privacy impacts of data encryption on the efficiency of digital forensics technology,' *CoRR*, vol. abs/1312.3183, 2013. arXiv: 1312.3183. [Online]. Available: http://arxiv.org/abs/1312.3183.

[8] M. Daniel, *Heartbleed: Understanding when we disclose cyber vulnerabilities*, https://obamawhitehouse.archives.gov/blog/2014/04/28/heartbleed-understanding-when-we-disclose-cyber-vulnerabilities, 2014.

[9] W. House, *Vulnerabilities equities policy and process for the united states government*, https://www.whitehouse.gov/sites/whitehouse.gov/files/images/External-UnclassifiedVEPCharterFINAL.PDF, 2017.

[10] T. Moore, A. Friedman and A. D. Procaccia, 'Would a'cyber warrior'protect us: Exploring trade-offs between attack and defense of information systems,' in *Proceedings of the 2010 New Security Paradigms Workshop*, 2010, pp. 85–94.

[11] B. Schneier, *Disclosing vs. hoarding vulnerabilities. schneier on security, may 22, 2014*, https://www.schneier.com/blog/archives/2014/05/disclosing_vs_h.html, 2014.

[12] E. M. Hutchins, M. J. Cloppert and R. M. Amin, 'Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains,' *Leading Issues in Information Warfare & Security Research*, vol. 1, p. 80, 2011.

[13] R. Adams, 'The advanced data acquisition model (adam): A process model for digital forensic practice,' Ph.D. dissertation, Murdoch University, 2012. [Online]. Available: https://researchrepository.murdoch.edu.au/id/eprint/14422/.

[14] R. Montasari, 'A standardised data acquisition process model for digital forensic investigations,' *Int. J. Inf. Comput. Secur.*, vol. 9, no. 3, pp. 229–249, Jan. 2017, ISSN: 1744-1765. DOI: 10.1504/IJICS.2017.085139. [Online]. Available: https://doi.org/10.1504/IJICS.2017.085139.

[15] R. Montasari, R. Hill, V. Carpenter and A. Hosseinian-Far, 'The standardised digital forensic investigation process model (sdfipm),' in *Blockchain and Clinical Trial: Securing Patient Data*, H. Jahankhani, S. Kendzierskyj, A. Jamal, G. Epiphaniou and H. Al-Khateeb, Eds. Cham: Springer International Publishing, 2019, pp. 169–209, ISBN: 978-3-030-11289-9. DOI: 10.1007/978-3-030-11289-9_8. [Online]. Available: https://doi.org/10.1007/978-3-030-11289-9_8.

[16] S. Caltagirone, A. Pendergast and C. Betz, 'The diamond model of intrusion analysis a summary by sergio caltagirone,' 2013.

[17] M. S. Khan, S. Siddiqui and K. Ferens, 'A cognitive and concurrent cyber kill chain model,' in *Computer and Network Security Essentials*, K. Daimi, Ed. Cham: Springer International Publishing, 2018, pp. 585–602, ISBN: 978-3-319-58424-9. DOI: 10.1007/978-3-319-58424-9_34. [Online]. Available: https://doi.org/10.1007/978-3-319-58424-9_34.

[18] B. I. Messaoud, K. Guennoun, M. Wahbi and M. Sadik, 'Advanced persistent threat: New analysis driven by life cycle phases and their challenges,' in *2016 International Conference on Advanced Communication Systems and Information Security (ACOSIS)*, IEEE, 2016, pp. 1–6.

[19] R. Luh, M. Temper, S. Tjoa and S. Schrittwieser, 'Apt rpg: Design of a gamified attacker/defender meta model.,' in *ICISSP*, 2018, pp. 526–537.

[20] G. Ioannou, P. Louvieris, N. Clewley and G. Powell, 'A markov multi-phase transferable belief model: An application for predicting data exfiltration apts,' in *Proceedings of the 16th International Conference on Information Fusion*, 2013, pp. 842–849.

[21] R. McKemmish, 'When is digital evidence forensically sound?' In *IFIP international conference on digital forensics*, Springer, 2008, pp. 3–15.

[22] D. McCullough, 'Uclinux for linux programmers,' *Linux Journal*, vol. 2004, no. 123, p. 7, 2004.

Paper V

Chip Chop — Smashing the Mobile Phone Secure Chip for Fun and Digital Forensics

G. Alendal, S. Axelsson and G. O. Dyrkolbotn, 'Chip chop - smashing the mobile phone secure chip for fun and digital forensics,' *Forensic Science International: Digital Investigation*, vol. 37, p. 301 191, 2021, ISSN: 2666-2817. DOI: https://doi.org/10.1016/j.fsidi.2021.301191. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666281721000998

Abstract

Performing mobile phone acquisition today requires breaking—often hardware assisted—security. In recent years, Embedded Secure Element (eSE) hardware has been introduced in mobile phones, with a view towards increasing the security of critical system features and encrypted user data. The idea being that the eSE should remain secure even if the rest of the system is compromised. The eSE is set to become crucial to modern mobile phone security, challenging Digital Forensics. The eSE is designed to withstand both logical and physical attacks, including side channel attacks, and to keep the attack surface towards the rest of the system/phone small, and complexity low to minimise the risk of implementation errors.

In this paper we adapt current state-of-the-art attacks to the eSE platform and present an attack on an eSE by Samsung, recently introduced in their premium mobile phones. We show how, with limited resources, our approach discovered a vulnerability that could be exploited, leading to a complete compromise of all the eSE security goals and a full loss of future

eSE trust, as mitigation of our attack in already fielded devices is challenging. This eSE is Common Criteria EAL 5+ certified and our attack exposes the gap between *intended* and *achieved* security, undermining the implied trust in such certifications.

We explain the eSE security design, the details of our attack, and discuss how a single vulnerability can have such devastating security results. The ultimate result of our research facilitates acquisition of affected devices, demonstrating use of offensive methods in advanced Digital Forensic Acquisition.

V.1 Introduction

The increased mandatory security and encryption of mobile phones is challenging digital forensics. This hindrance is discussed in the general media [2, 3] as well as research circles [4]. Security and encryption seem to be the major challenges in the years to come. Trusted computing (TC) in form of a stand-alone eSE HW, in addition to the existing TrustZone [5], is adding an extra layer of security that needs to be broken. All these security features motivate digital forensic acquisition (DFA) to turn to offensive techniques, like security vulnerability research and exploitation [6].

Trusted computing is the concept where a system is expected to behave as intended, withstanding outside influence, and enforced by trusted, stand-alone hardware and software. The concept is not without controversy and has caused discussion of its benefits, and risks [7, 8]. However, the idea is still implemented by many vendors, and to support trusted computing, several hardware (HW) solutions exist today. Intel Software Guard Extensions (SGX) [9], Trusted Platform Modules (TPM) [10], Trusted Execution Environment (TEE) [11], Hardware Security Modules (HSM) [12] and Secure Element (SE) [13] are all examples of technology providing physical / HW assisted separation *inside* a system to provide trusted, tamper-proof and secure environments for system critical security elements. One common design principle is the need for a separate root of trust, to prevent security breaches even if the overall system is compromised [14]. This isolated system-within-the-system is to be made secure by keeping complexity low, and implementation quality high. One advantage of lower complexity is that the probability of software bugs and side-channel attacks is reduced as a consequence of the smaller code size [15, 16]. Increased quality can be achieved by improving development methodology, e.g. by working according to certain standards, such as those meeting Common Criteria Evaluation Assurance Level (CC EAL) certification requirements [17]. The intention being that a higher CC EAL level increases

the reliability of the security features implemented.

In general terms, the eSE concept consists of specialised HW providing certain system critical security features to the host system without depending on that host system for any execution of code nor storage of data. This "black-box" principle means the eSE has full control of its own processor, RAM and storage. This setup is meant to prevent a compromised host system from reading the eSE embedded code and data, and to make it more difficult to perform side-channel attacks, like observing or influencing execution of eSE sensitive code.

An advantage of this physical separation is that development and production of eSE HW can be outsourced to specialised vendors with a secure production environment. The host system vendor need only to follow the documented eSE interface to incorporate it in end products. However, one major drawback is that this approach risks the introduction of a single point of failure. A failure in the eSE can have devastating effects on the operation of all systems using the eSE as a basis for their security. Another drawback is that the host system vendor needs some form of trust in the eSE HW, to certify that the eSE security features are securely implemented and working as intended. This is one of the intentions of performing a CC EAL certification.

A concept corresponding to eSE was presented in the Android Operating System (OS) version 9. Mayrhofer et al. [18] explain Google's views on the "The Android Platform Security Model", discussing a.o. the different threat model for mobile devices. They discuss the use of a *strongbox* that "...implements the Android keystore in separate tamper resistant hardware (TRH) for even better isolation. This mitigates [T1] and [T2] against strong adversaries..." [18, p. 8]. Their definition of threats [*T1*] is "Powered-off devices under complete physical control of an adversary (with potentially high sophistication up to nation state level attackers), e.g. border control or customs checks" and [*T2*] is "Screen locked devices under complete physical control of an adversary, e.g. thieves trying to exfiltrate data for additional identity theft." [18, p. 3]. T1 and T2 clearly identifies the most advanced and resourceful adversaries. We will use the term eSE in place of Google's term *TRH* for consistency throughout this paper.

In this paper we present a remote attack on a state-of-the-art eSE HW utilised by the major Android mobile phone vendor Samsung. The attack is *remote* as we attack the logical interface, as opposed to *local* attacks in need of physical access. Our attack bypasses the security of the eSE, protecting sensitive encryption key material, and facilitates digital forensic acquisition (DFA) of user data. This attack will work on powered off devices, known as the *before-first-unlock* (BFU) state, with no knowledge of user credentials. We show that although placing all trust in a single, well protected, entity

may be tempting, it also means the introduction of a single point of failure, and if done wrong the whole trusted computing design falls, leaving the system totally exposed. This eSE is present in Samsung's high-end mobile phone models and represents the state of the art in modern Android security. The eSE HW is CC EAL 5+ certified, and is thus expected to provide a very high level of security. Samsung uses CC EAL certifications to promote the security of their eSE [19] and also to justify the high security level of the Samsung Galaxy S20 mobile, needed in mobile eID solutions for use in Germany [20, 21]. CC EAL certifications have been proven problematic by other authors as well [22, 23], and our attack shows that such certifications are no guarantee the proper security level has been achieved. Our attack demonstrates the failure of all CC EAL5+ goals for the eSE HW. Further, our analysis shows that patching isolated eSE HW is challenging, making it hard to regain the expected CC EAL 5+ security level in already shipped mobiles.

Our contribution can be summarised as:

- The adaptation and improvement of state-of-the-art black-box attack techniques applied to the eSE HW platform. The stand-alone eSE significantly changes the attack path compared to conventional TEEs, like ARM TrustZone implementations.
- The discovery of previously unknown, *remotely exploitable*, security vulnerabilities that fully breaks the confidentiality and integrity of the CC EAL 5+ certified eSE HW.
- A demonstration of the gap between the *intended* and *achieved* security, and how certifications, like Common Criteria EAL, fails to deliver the needed trust in implemented solutions.
- A presentation of the full attack development and exploitation of the eSE, with example attacker use.
- Analysis of the effect of a vulnerability exploit in the eSE HW and the lack of eSE countermeasures.
- A demonstration of digital forensic goals: off-device brute force of user screen lock credential, necessary for digital forensic acquisition of encrypted user data.

The rest of this paper is organised as follows. In Section 'Background' we introduce needed background and the targeted eSE. Related work is discussed in Section 'Related Work' and Section 'The Attack' contains the attack steps performed and the technical details on the vulnerability and its exploitation. In Section 'Attack Implications' we present example implications of the attack. Finally we will present our discussion and conclusions in Sections 'Discussion' and 'Conclusions and Future Work'.

V.2 Background

In this section we introduce some needed background material. First an introduction to the specific eSE HW targeted by our attack and its CC EAL 5+ certification. The eSE threat model is then discussed to clarify our attack approach, communicating with the logical interface using a protocol based on the "Application Protocol Data Unit" (APDU). Refer to 'APDU primer' in Appendix for a brief APDU introduction.

V.2.1 Embedded Secure Element

The eSE HW under investigation is the Samsung
S3K250AF embedded Secure Element [19]. This eSE was introduced in February 2020 by Samsung, with the release of the Samsung Galaxy S20 product line. Our test devices were the Samsung Galaxy S20 Ultra 5G (SM-G988B), the Samsung Galaxy S20 (SM-G980F) and the Samsung Galaxy Note 20 Ultra 5G (SM-N986B). All these models use the Exynos SoC. The upcoming Galaxy S21 models with the Exynos SoC are also believed to include the S3K250AF, but this has not been confirmed at the time of writing, and was not part of our research.

We will mostly refer to the "S3K250AF eSE" simply as the "eSE" throughout the rest of the paper.

The S3K250AF eSE is a single chip solution, soldered to the printed circuit board (PCB) of the mobile. It has a small form factor, pictured in the Samsung promotion material [19]. The eSE processor is an ARM SecurCore SC000 [24], according to the NIST Cryptographic Algorithm Validation Program (CAVP) for the S3K250AF [25]. The architecture is ARM BE8 mode [26]. This architecture uses little-endian for code and big-endian for data and pointers. The S3K250AF contains 252 kilobytes (kB) on-board flash storage, according to the CC EAL documents [27]. Samsung promotes an eSE standard development kit (SDK) [28], but we have not evaluated this SDK as this entails us signing a non-disclosure agreement.

V.2.2 CC EAL

The S3K250A holds a CC EAL 5+ certification [29] from Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI) [30]. The certification is accompanied by two documents. The security target (ST) document [27] by Samsung describes the S3K250A and its security requirements, and the second document [31] by third parties describes the intended protection profile, which is generic and not specific to the S3K250A.

The main security goals for the eSE (SG1-SG3) ([27, p. 46]) are to

maintain *integrity* of user data (SG1), to maintain *confidentiality* of user data (SG2), and to maintain *correct operation* of the services provided by the eSE (SG3). So an attacker should not be able to change any stored eSE data, read any stored eSE data (without authorisation), and not influence the operation of any of the eSE features offered.

The CC EAL 5+ certification is an aid to achieving these goals, and it states "Certification does not in itself constitute a recommendation of the product by the National Information Systems Security Agency (ANSSI), and does not guarantee that the certified product is completely free from exploitable vulnerabilities."[1] [30, p. 2]. As such a guarantee is impossible to give, some effort has been done to lower the probability of the existence of such vulnerabilities. One such effort is the Common Criteria *Advanced methodical vulnerability analysis* (AVA_VAN) [32]. This vulnerability assessment aims to determine potential vulnerabilities. AVA_VAN is divided into levels ranging from 1 to 5 with " increasing rigour of vulnerability analysis by the evaluator and increased levels of attack potential required by an attacker to identify and exploit the potential vulnerabilities" [32, p. 184]. Level 5: "AVA_VAN.5 Advanced methodical vulnerability analysis", is the highest level. This level specifies that "A methodical vulnerability analysis is performed by the evaluator to ascertain the presence of potential vulnerabilities." [32, p. 188]. AVA_VAN.5 is part of the S3K250A CC EAL 5+ certification [30, p. 3]. Thus AVA_VAN.5 is a best effort to reveal any vulnerabilities of the S3K250A. It is unclear to us what exact analysis steps were performed by the evaluator in this particular case, but AVA_VAN.5 is referenced in the certification document [30], assuring that sufficient analysis was performed to achieve a CC EAL 5+ certification with AVA_VAN.5.

V.2.3 eSE Threat Model

Adapting the threat model of Mayrhofer et al. [18], we consider the eSE against threats [*T1*] and [*T2*], as these are the threats this TRH / eSE is designed to mitigate. These scenarios assume an attacker with physical control of the eSE. Attacking an isolated HW component, like the eSE, two main attack vectors present themselves: The logical interface between eSE and the host system, known as the Rich Execution Environment (REE), and (possibly HW assisted) side-channel attacks on the eSE. The logical interface between the eSE and REE uses "Application Protocol Data Units" (APDU), originally a communication protocol for smart-cards. APDU based communication, accepting attacker commands and data, could be vulnerable to design and implementation bugs. A simplified view of the logical

[1] Our translation from French.

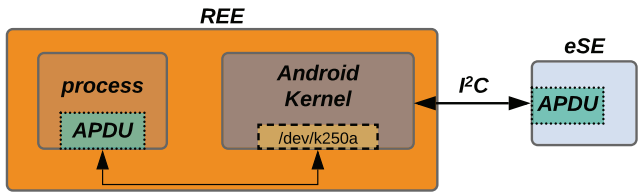


Figure V.1: eSE logical interface using APDU

interface between eSE and the REE is shown in Figure V.1.

V.3 Related Work

Attacks on black-box physical separation implementations are not new. Anderson et al. [33, 34] discusses the security of tamper-resistant cryptographic processors. They discuss their use and attacks with focus on two different attack scenarios: attacks involving physical access and logical attacks. Attacks involving physical access are referred to as *local* attacks and most side-channel attacks fall into this category, needing some physical interaction to mount an attack. Logical attacks are referred to as *remote*: These are attacks on the logical interface and they do not require physical access, and are thus independent of the distance between the attacker and the attacked device. Anderson et al. refer to these attacks as *API* attacks, using the provided Application Programming Interface (API). Exploitation of design and code flaws fall into this category. Anderson et al. discuss several attacks, including a cryptographic API attack on the IBM 4758 cryptoprocessor. The attack demonstrated design flaws leading to information leaking via the API, which could be used to mount a brute force attack on embedded DES keys. The security of the IBM 4758 HW was rendered moot because of flaws in the software running on the device. Anderson et al. predict in their conclusions that logical attacks "..are likely to remain the weak spot of most high-end systems".

More advanced attacks via the logical interface, relevant to physical separation implementations, can be found in more recent research. Bittau et al. [35] demonstrate how to write a remote buffer overflow attack without knowledge of the target binary. Where traditional attacks use known *gadgets* within the target binary to craft ROP attacks, Bittau et al. improve on this technique by using a so called blind ROP (BROP). The BROP technique can be used to attack closed source and unknown implementations using leaked information. Thus useful ROP gadgets can

be found simply by trial and error, building a complete attack using only simple information leak oracles, like a program crash. Lee et al. [36] use a similar approach to attack Intel SGX Secure Enclaves. Their attack, named *Dark-ROP*, uses information leak oracles from the Intel SGX to locate ROP gadgets and from there to build a functional ROP attack against selected Secure Enclaves. Dark-ROP demonstrates that critical implementation errors in secure enclave code can still be exploited by attackers, without knowledge of the target code. Van Bulck et al. [37, 38] demonstrate another powerful attack, *Foreshadow*, attacking the CPU cache to retrieve secure enclave secrets. There are several published papers on the security of Intel SGX [39–42].

Moghimi et al. [23] recently demonstrated an attack on TPMs, some CC EAL 4+ certified. Their attack uses black-box timing analysis to reveal secret key information during signature generation based on elliptic curves. Using this attack they demonstrate retrieval of 256-bit private keys. A key element in their attack is the magnitude of increased operating frequency of the main SoC compared to the TPM, facilitating high frequency timing of the "slow" TPM execution.

Numerous attacks exist on TrustZone implementations [43–46], demonstrating that code vulnerabilities, like design and coding quality, are crucial for security, often with devastating effect on security when such vulnerabilities are found. Cerdeira et al. [47] have summarised current security challenges of TrustZone-based TEE systems.

V.4 The Attack

The completely stand-alone eSE HW affects how an attack can be designed and performed. Compared to attacks published on other secure execution environments, discussed in the previous section, this requires a different approach. The major difference is the changed attack surface, requiring a different attack chain, with new attack oracles.

Our attack adapts elements from both BROP by Bittau et al. [35] and Dark-ROP by Lee et al. [36] to the physical separated black-box eSE HW, and we are, to the best of our knowledge, the first to do so. Although partly available for this particular eSE, our attack does not require knowledge of the binary (FW). We incorporate information leak oracles to aid in the attack on the eSE HW.

The attack was developed following these generic steps:

- **Information Gathering** Gain knowledge of the target eSE and how it is used.
- **Identify Attack Vectors** Gain knowledge of the eSE attack surface

with potential attack vectors.

- **0-day Information Leakage** Locate at least one information leakage oracle to aid in 0-day vulnerability discovery.
- **0-day Vulnerability Discovery and Exploitation** Locate at least one new exploitable vulnerability and use the discovered vulnerability to break confidentiality (secure data exposure) and/or break integrity (writing to code or data memory).

The resulting attack on the Samsung S3K250AF eSE HW [19] will follow, with the last section discussing the technical capabilities of our attack.

V.4.1 Attack Assumptions

Our attack is based on the assumption that we have access to the logical interface of the chip. This logical interface is exposed by the `/dev/k250a` virtual device (see Figure V.1). Access to this device enables the attacker to communicate, using APDUs, with all exposed functionality of the eSE HW. Thus, in this case, we can operate as a privileged REE process similar to the process depicted in Figure V.1. In a test environment this is achieved simply by executing a binary we provide with system privileges. We implemented all attack functionality to communicate with the eSE. We call this tool `chip_breaker`. Our setup executed this tool through a "root" adb shell [48], connected to test devices either with a cable or over a network connection. In a more realistic attack scenario, depending on how the attacker gains access, this can be achieved by infecting a process with system privileges and then communicating with the eSE. The next section identifies one such target process.

Our assumption seems realistic, as the design of the eSE is to withstand attacks against a fully compromised REE. Note that we do *not* require physical access to the chip, which might be a prerequisite for many side-channel attacks. Hence, our attack can even be performed remotely, over the air, assuming we have gained privileges to communicate with the eSE logical interface. So our attack can be performed using any remote, local or physical attack that gains elevated execution, like "root", on the device. Elevated execution can be achieved without triggering user data wipe. One path is to break the secure boot of the device to introduce attacker code [6, 49]. As history has shown that gaining such access is not necessarily difficult or uncommon [50], we do not address that problem further in this paper. Even de-soldering the eSE chip and communicating directly on the I2C lines is an option to perform our attack.

V.4.2 Information gathering

Several important initial information sources were identified:

- CC EAL certification documents [27, 30, 31].
- An android service process, `hermesd`. This privileged process communicates with the eSE using the APDU-based logical interface and it is the only REE process with this ability (Figure V.1). Processes communicating with the eSE were revealed by observing access to the eSE virtual device, `/dev/k250a`.
- Vendor specific libraries supporting `hermesd`. The most important being `libese-grdg.so`. This library implements the low level communication with the eSE. This communication uses APDUs. APDUs are communicated over the eSE logical device `/dev/k250a`.
- FW files found to be accessed by `hermesd`: `/vendor/etc/secnvm/k250a_00000009.img` and `/vendor/etc/secnvm/k250a_00000009_dev.img`.
  These files contain partly encrypted FW updates for the eSE. These files were revealed by observing files accessed by `hermesd`.

Unencrypted parts of `k250a_00000009.img` and `k250a_00000009_dev.img` revealed code in ARM THUMB mode [51]. The file `k250a_00000009.img` was assumed to be a "production" FW container. We refer to this as `FW_prod`. Correspondingly the `k250a_00000009_dev.img` is assumed to be a "development" FW container. We refer to this as `FW_dev`. Our research only recovered one version each of both these files, on all tested models, and analysed model FW (Appendix, Table V.1).

We inspected the partially unencrypted `FW_prod` and `FW_dev`. These turned out to be container files for different "images" for the eSE. The different image names are: `BOOT`, `CRPT`, `CORA`, `CORB`, `SNVM`, and `IWEA`. We developed a simple script to parse and extract images from this proprietary container format (Appendix, Table V.1). This revealed that most of the images are encrypted, while the images `SNVM` and `IWEA` are not. Images `SNVM` and `IWEA` are also signed, thus an attack on these images using simple FW modifications seems less probable. In later attack steps we recovered the encryption key to the encrypted images, and the decrypted images all included image signatures (Section 'Attack Capabilities and AES Key Exposure').

The Logical eSE Interface Attack Vector

The logical eSE interface utilises APDUs for communication (Appendix, 'APDU primer'). Thus all eSE APDU communication is considered a potential attack vector and we need to expose as many eSE APDU handlers

as possible. These APDU handlers are implemented by code running on the eSE ARM processor. The handlers will potentially accept attacker controlled input, which could lead to an input validation vulnerability. In addition to all APDU handlers, the APDU transport layer is an additional attack vector. Both the APDU handlers and the APDU protocol handling are part of the logical eSE interface.

All valid APDU CLA and INS values correspond to APDU handler functions within the eSE code. Observing the `hermesd` process communicating with the eSE using APDU and reverse engineering the the REE library, `libese-grdg.so`, enabled us to reveal the communication logic between the REE (`hermesd`) and the eSE. The exposed eSE specific functions in `libese-grdg.so` are listed in Table V.2 (Appendix) with their corresponding `grdg_*` name. These functions revealed valid APDU CLA and INS values, each communicating with different APDU handlers inside the eSE. As these functions only expose eSE features utilised by `libese-grdg.so`, additional eSE APDU handlers might exist. Some were indeed exposed by brute force of the APDU logical interface. By design, all the different APDU CLA and INS handlers inside the eSE are expected to return valid SW values, indicating success or various error states. Gkaniatsou et al. [52] demonstrated REPROVE, a system to aid in the reverse engineering of APDUs used in smart-cards. Inspired by their work, we produced a simple brute force process shown in Listing V.1, simply trying various combinations of (CLA,INS) pairs and observing returned SWs. The unknown SW response "unknown_command" classification is vendor implementation dependent and might vary from vendor to vendor, and even from CLA to CLA. However, it should be easily spotted as being the most common SW reply from a specific (valid) CLA and random (thus most probably not implemented) INS.

```
for ( all possible CLA ) {
  for ( all possible INS ) {
    SW = APDU_communicate_with_eSE(CLA, INS)
    if ( SW != unknown_command ) {
      // potential valid (CLA,INS) found
      // optional next step:
      P1_P2_Lc_Data_Le_brute_force(CLA, INS)
    }
  }
}
```

Code listing V.1: Simple APDU brute force pseudo code

Be warned that brute forcing valid APDU handlers might trigger an unwanted effect in the eSE if a valid (CLA,INS) pair is hit with valid P1, P2, Lc and Le values. One example could be a "factory reset" APDU, not in need of any valid P1, P2, Lc or Le values. Thus unknown (CLA,INS) pairs

with SW values indicating success, `0x9000`, should be treated with some caution.

Table V.2 in the Appendix lists eSE APDU handlers discovered through reverse engineering the `libese-grdg.so` library, APDU brute forcing, and confirmed by reverse engineering of the dumped eSE flash recovered later in the attack (Section 'Arbitrary flash and RAM read'). Knowing the available APDU handlers for the eSE allowed us to establish communication with eSE using its own protocol. All APDU handlers could potentially be exploited to have eSE perform unintended actions and is the most important attack vector for this eSE.

V.4.3 0-day Information Leak Oracles

Attacking a black-box entity like this eSE requires "blind" attack techniques, as introduced in Section 'Related Work'. Such attacks depend on information leaks from the device (oracles). That is, an attacker needs a way to know if an attack vector behaved unexpectedly, such as a crash. Any observable execution specific information from the eSE could potentially be a useful oracle. Potential oracles are e.g crash dumps, exception handling, page fault addresses, execution timing, etc. Such oracles could leak valuable information in the trial-and-error progress of a "blind" attack. For an eSE this could mean leaking information on code addresses, stack addresses, code content, data content, etc.

The physical separation of eSE makes such observation of (erroneous) behaviour challenging, reducing the existence of oracles. With a stand-alone eSE there is no returned crash response, no exception handler observation, no observable page faults, etc. Timing attacks can also be difficult, measuring execution time from outside the eSE. In our case we looked for logical information leak oracles, where information could be obtained through observable (mis)behavior by the normal logical interface.

We identified two information leak oracles that both play an important role in the attack.

Oracle 1 The first oracle is a common observable behaviour in black-box implementations: the lack of response. This is often the result of a crash. This is also the case with this eSE, which is expected to always reply with a status word (SW). Thus any crashing APDU handler will result in no SW being returned (a timeout error).

Oracle 2 An attacker can also try to look for logical information leak oracles in the ADPU handlers. Candidate APDU handlers are especially those

```
0000  53 65 63 72 65 74 00 00  00 00 00 00 00 00 00 00
0010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0020  00 00 00 01 00 00 00 D0  00 00 00 00 00 00 00 00
0030  00 00 00 00 00 00 00 00  00 00 00 01 00 00 00 05
0040  01 22 49 31 20 00 14 28  20 00 27 C0 00 00 00 00
0050  20 00 14 80 FF FF FF FF  00 02 85 F9 20 00 14 80
0060  20 00 27 C0 00 02 85 8B  00 00 00 00 20 00 0B 50
0070  00 00 00 00 FF FF FF FF  00 01 04 7F 00 00 00 00
...
```

Figure V.2: eSE stack leak using the `APDU_writeWeaver` / `APDU_readWeaver` oracle

that read and write data. If any of these functions can be manipulated to return more data than expected, leaked information can be used to mount a ROP attack.

Candidates are all `get`, `put`, `read` and `write` functions in Table V.2 (Appendix).

The eSE handlers corresponding to `libese-grdg.so` functions `grdg_readWeaver` and `grdg_writeWeaver` were identified as an information leak oracle, when combined. We could not use the `libese-grdg.so` functions `grdg_readWeaver` and `grdg_writeWeaver` directly because of checks performed by the library before submitting the APDU to the eSE, so we re-implemented these REE functions. Our `chip_breaker` tool contains new versions of `grdg_write-/readWeaver` named `chip_breaker_write-/readWeaver`. We call the corresponding eSE APDU handlers `APDU_write-/readWeaver`. One implementation of these two eSE ADPU handler functions can be found in the `IWEA` image in `FW_dev` (Appendix, Table V.1). These eSE handler functions could together become an oracle in the following way: The eSE `APDU_writeWeaver` receives two buffers of data from `chip_breaker_writeWeaver`: `CHALLENGE` and `SECRET`.
`APDU_readWeaver` would send back a `SECRET` buffer from the eSE *iff* the caller submitted a matching `CHALLENGE`, written to on-board storage with `APDU_writeWeaver`. The oracle revealed itself by manipulating a `SECRET` length of `>32`, as this seemed to be a fixed length used inside the eSE. The `SECRET` size variable sent is only one byte, so `SECRET` length can be in the interval `>1` and `<256`. Thus `APDU_readWeaver` would return a `SECRET` buffer with up to `256` bytes of data. This leaked valuable stack data.

A stack leak from this oracle can be seen in Figure V.2.

Thus we had two information leak oracles: lack of APDU response if the eSE crashes (Oracle 1) and a stack leak from `APDU_writeWeaver/APDU_readWeaver` (Oracle 2).

The Oracle 2 stack leak in Figure V.2 gave us valuable information, indicating memory pointers at offsets `0x44` (`0x20001428`), `0x48` (`0x200027c0`), `0x50` (`0x20001480`), `0x5c` (`0x20001480`), and so on.

Keeping in mind this is ARM BE8, memory pointers are 32 bit big-endian. This makes these point to memory locations all in the `0x2000xxxx` range. Further, code pointers can be found at offsets `0x58` (`0x000285f9`), `0x64` (`0x0002858b`), `0x78` (`0x00010423`), and so on. The reason is that they can all be interpreted as 32-bit ARM BE8 THUMB mode addresses, where the least significant bit (LSB) is always 1 to indicate THUMB mode to the processor. These code pointers are `POP`'ed from the stack during a typical ARM THUMB function epilogue: `POP {PC}`.

The leaked addresses gave valuable information both for further reverse engineering efforts and for exploitation.

V.4.4 0-day Vulnerability Discovery and Exploitation

Oracle 2 is crucial for both vulnerability discovery and revealing information to further understand the attack vectors, but more importantly to reveal information needed for successful exploitation, revealing memory addresses for use in for example a ROP attack.

Oracle 2 was further developed by submitting larger `SECRET` buffers to `APDU_writeWeaver`, and not only to manipulate the returned size in `APDU_readWeaver`. Submitting a `SECRET` buffer larger than 84 bytes led to Oracle 1 activating with no reply from the eSE. This indicated a crash and we assumed from the leaked stack contents in Figure V.2 that we were overwriting important stack pointers. However, since Figure V.2 shows the leaked stack from `APDU_readWeaver`, this did *not* necessarily match the stack of `APDU_writeWeaver`. Without knowledge of `APDU_writeWeaver` code, we could now implement a simple brute force attack for `secret[84:88]` based on the assumption that this was a code pointer and not a data pointer. If this was the case, there should be at least one address that responds with a SW, indicating an attacker controlled ROP. The leaked stack from Figure V.2 already gave valuable ranges for brute forcing. Having access to the `IWEA` code image extracted from `FW_dev`, this step can also be solved by reverse engineering the eSE APDU handlers `APDU_writeWeaver` and `APDU_readWeaver`. We manually estimated the stack use by both eSE handlers and adapted to any changes between the two. This enabled us to correctly guess the stack layout of the `APDU_writeWeaver` function based on observation of the `APDU_readWeaver` stack leak.

Analysing the trigger of Oracle 1 showed that `APDU_writeWeaver` suffered from a standard stack buffer overflow [53], enabling a full overwrite of the `IWEA` slot storage (`SECRET` + `FOOTER`) and then `APDU_writeWeaver` stack data. Figure V.3 shows a simplified view of the effect of the buffer overflow: The first 32 bytes are written to the normal

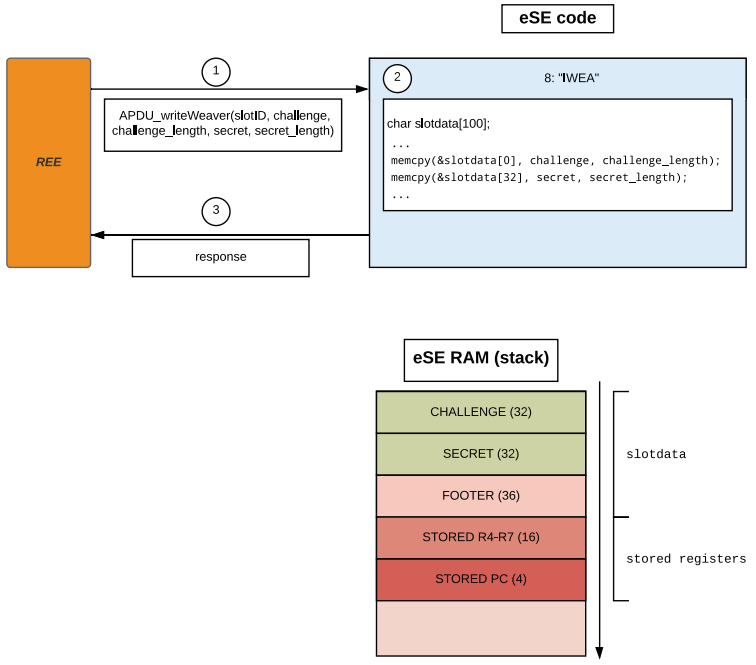


Figure V.3: Buffer overflow in eSE APDU_writeWeaver handler

`SECRET` buffer. The next 36 bytes overwrite the `FOOTER` and the next 16 bytes overwrite values of registers `R4-R7` stored on the stack. Finally, the next 4 bytes overwrite the stored `LR` register, which will get `POP`'ed into `PC` when `APDU_writeWeaver` returns. This leads to the now well known subversion of control flow and could be used for a ROP attack.

Arbitrary flash and RAM read

The `APDU_writeWeaver` buffer overflow can be used to read flash and RAM memory by locating a special ROP gadget that takes an attacker controlled address as input and will return 16 bytes. The ROP gadget in Listing V.2 can be set up by crafting the stack overflow with correct values of `R4-R7` which are *identical* to those stored on the stack for `APDU_readWeaver` (Figure V.2). This is due to the semi-static nature of the eSE running with a 100% predictable execution and memory layout. So we control `R4-R7` and `PC`, which is set to the address of the ROP gadget.

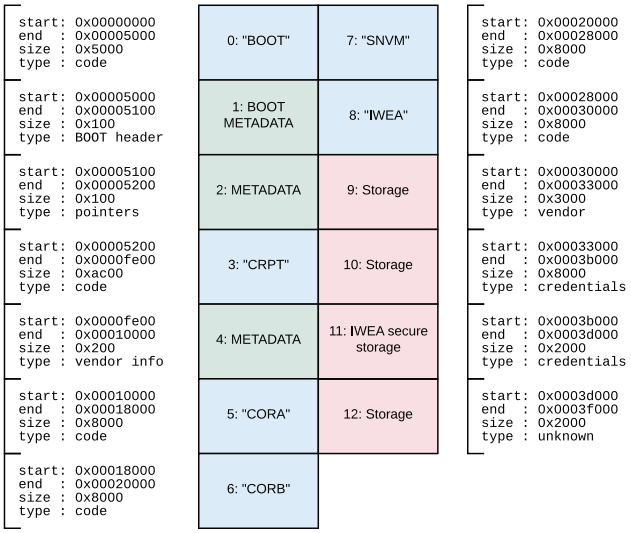


Figure V.4: Full eSE flash layout

```
MOVS    R0, #0x10   ; size to read
STR     R7, [R4]    ; Store address
STR     R0, [R4,#4] ; Store size
MOVS    R0, #0x90   ; SW1
STRB    R0, [R4,#8] ; Store SW1
MOV     R0, R5      ; SW2
STRB    R5, [R4,#9] ; Store SW2
POP     {R1-R7,PC}  ; pop and return
```

Code listing V.2: ROP gadget for arbitrary flash and RAM read

This simple ROP gadget can be used to read the full flash and RAM of the eSE by setting the `R7` to the address to read 16 bytes from, and iterate. Indeed, we used this ROP gadget to read both the complete eSE flash and RAM. The resulting layout of the dumped 252K eSE flash can be seen in Figure V.4. The code image names, 0-8, are matched with the corresponding image names from the FW file `FW_dev`. The names of the secure storage data images, 9-12, are based on reverse engineering code images and their use of various secure storage addresses.

Arbitrary code execution

The `APDU_writeWeaver` buffer overflow can even be used to execute attacker provided code. As there is no NX or other “no execute” protection of

the eSE stack memory, we can simply execute supplied *shellcode*. Embedding ARM code in the `SECRET` buffer and setting the `PC` to this stack address will execute arbitrary attacker controlled code in the eSE. The address of this stack buffer was located by using the ROP gadget (Listing V.2) to dump the stack memory, in the range `0x20000000 - 0x200002800`.

This provided us with full read *and* write control of the eSE HW flash and RAM. All code and secure storage from Figure V.4 could thus be read and written to. The use of this exploit is demonstrated in Section 'Attack Capabilities and AES Key Exposure' and Section 'Attack Implications'.

Our developed `chip_breaker` tool fully implements the exploit of this vulnerability, executing any provided shellcode on the eSE processor.

Persistence

The code images `BOOT`, `CORA`, `CORB`, `CRPT`, `SNVM`, and `IWEA` are all stored unencrypted and unsigned on the eSE flash. The only integrity checks performed on any image *after* flash write (as part of a FW update) are simple CRC32 and SHA256 hash verifications. These hashes are also stored on the eSE flash. This means that we can freely modify any code image on the flash and simply update the corresponding integrity check hashes. This means that the eSE has no root-of-trust and there is no secure boot present. The consequence is that there is no way the eSE can verify any code stored on the eSE flash during boot, where the eSE starts executing on-board ROM before continuing execution of `BOOT`. This `BOOT` image is writable by us, *without* any signature verification, and this completely breaks the code trust of the eSE. We confirmed this by developing a *writeflash* shellcode that was capable of modifying any code image. These changes were persistent across reboot of the device and thus reboot of the eSE. This shellcode was tested with our `chip_breaker` tool.

V.4.5 Attack Capabilities and AES Key Exposure

With the full dumping of eSE flash (Section 'Arbitrary flash and RAM read'), all eSE secure storage is now readable to us (data images 9-12 in Figure V.4). Also, full reverse engineering of the eSE images `BOOT`, `CORA` and `CORB` is now possible. These images are not encrypted on the eSE flash, which suggests that they are decrypted as part of the FW update process. This turned out to be performed with an embedded eSE AES key and initialisation vector (IV) embedded in the dumped `BOOT` and `CORA` images. As this key is now exposed by our attack, any attacker with knowledge of this key can decrypt any previous, and *future*, FW updates for the eSE. We verified this by decrypting the `BOOT`, `CORA` and `CORB` images in the `FW_dev`

FW file (Appendix, Table V.1). Updating this pre-shared AES key and IV can thus not be done by supplying a new eSE FW update file, as part of a normal over-the-air (OTA) phone FW update, as this would leak the new key to an attacker already aware of the present one. This update can only be done by a secure update mechanism, such as physically attaching to the eSE HW at a secure vendor site.

Although our attack has fully compromised the security of the S3K250AF eSE HW, our research is by no means exhaustive. More eSE FW security vulnerabilities might exist, including the previously encrypted eSE images, now available for vulnerability research. Our research did not evaluate any side-channel attacks and such vulnerabilities might also exist, as our research identified non-constant time execution functions in the eSE FW. One example is the data dependent execution of the internal `memcmp()` functions, used for example in authentication functions (`grdg_readWeaver` in Table V.2). As the S3K250AF, containing the exposed vulnerability, can be flashed with arbitrary researcher provided code, it is an ideal research platform for future research of the eSE HW and its resistance against side-channel attacks.

V.5 Attack Implications

Our full compromise of the eSE has devastating effects on the system security of affected devices. All eSE security features are made moot by our attack, as an attacker can read and write arbitrary flash and RAM, in addition to making persistent changes to any code and data stored in the on-board flash. This is trivial due to the eSE lacking security features like NX, ASLR, Stack canaries and secure boot. All code in the eSE is running in a single thread of execution, with no privilege separation. This means that a single compromise, like that demonstrated by our attack, gives access to all code and data from both flash and RAM.

In this section we demonstrate a confirmed example of a security feature that fails as a consequence of our attack. We note that Android Keymaster and device attestation also seem to be affected by a vulnerable eSE as both features seem to rely on eSE security features. However, we have not confirmed this.

The following example has been implemented and verified as working.

V.5.1 Android User Screen Lock Brute Force

This section demonstrates how to recover the user screen lock credential. The user screen lock credential is used, together with the encryption key material contained in the eSE secure storage, to reproduce the Credential Encrypted (CE) storage encryption key needed for Android's file-based encryption (FBE) [54]. The CE storage contains most of the sensitive user data on the device and thus the eSE is crucial in protecting the needed key material. Recovering the screen lock credential is therefore mandatory to facilitate digital forensic acquisition (DFA) of powered-off devices and devices seized before the user has unlocked the device at least once since power on, known as the *before-first-unlock* (BFU) state.

The Android user screen lock protection supports the use of a "weaver" hardware abstraction layer (HAL). Google has documented this HAL [55]. The documentation states that the weaver provides secure storage for secret values and that these may only be read if a corresponding key, or *challenge*, has been provided. The S3K250AF eSE provides the weaver functionality in the `IWEA` image (Figure V.4), accessible through the `grdg_writeWeaver` and `grdg_readWeaver` functions in `libese-grdg.so` (Appendix, Table V.2). With our attack in Section 'The Attack' an attacker can read of all the eSE secure storage, including storage belonging to `IWEA` (image 11 in Figure V.4). This means that sensitive `IWEA` storage belonging to the Android user screen lock protection can be read by an attacker without knowledge of the corresponding challenge. A fragment of the Google screen lock verification code [56] running on affected test devices can be seen in Listing V.3.

```
// Weaver based user password
...
result.gkResponse = weaverVerify(weaverSlot, passwordTokenToWeaverKey(
    ↪ pwdToken));
if (result.gkResponse.getResponseCode() != VerifyCredentialResponse.
    ↪ RESPONSE_OK) {
    return result;
}
...
applicationId = transformUnderWeaverSecret(pwdToken,result.gkResponse.
    ↪ getPayload());
```

Code listing V.3: `unwrapPasswordBasedSyntheticPassword()` code

A user-entered credential, a pattern, pin or password, is transformed by a key derivation function (KDF) into `pwdToken`, which again is transformed into a `CHALLENGE` by `passwordTokenToWeaverKey()`. This `CHALLENGE` is verified by the eSE using `grdg_readWeaver`. If the eSE successfully verifies the `CHALLENGE`, `weaverVerify()` will re-

turn the corresponding eSE stored `SECRET`, accessible through the call `result.gkResponse.getPayload()`. Both the `pwdToken`, derived from `CHALLENGE`, and `SECRET` are needed in the screen lock verification. These are also used to unlock the encryption keys used for the on-device file-based encryption (FBE) of user data [54].

As we can bypass the `weaverVerify` verification step and instantly retrieve the correct `CHALLENGE` and `SECRET` from the eSE, *off-device* brute force of user credentials can be achieved by performing a brute force attack outlined in Listing V.4. The `passwordTokenToWeaverKey()` simply produces a SHA512 hash and `KDF()` is currently `scrypt()`. The `salt` can be retrieved from the on-device file `/data/system_de/0/spblob/<id>.pwd`, available under the same attack assumptions as before (Section 'Attack Assumptions').

```
if (passwordTokenToWeaverKey(KDF(passcode_candidate, salt))[0:32] ==
    ↪ eSE_CHALLENGE ) {
  // correct passcode found
}
```

Code listing V.4: Simplified screen lock brute force pseudo code

We successfully implemented a simple CPU based python version of this off-device screen lock brute force attack, and the results showed that an attacker could recover any four digit pin or 3x3 pattern in less than 1 h on a modest dual-core laptop. This attack could of course be highly optimised on dedicated HW to drastically improve performance.

With the user screen lock credential recovered by this brute force attack, we gain full access to the contents of the mobile device. The credential can be used to authenticate and retrieve FBE keys protecting user data. This fully breaks the confidentiality of the device and the encrypted user data.

V.6 Discussion

Our attack shows how recent (and not so recent) research in attack techniques ([33, 35, 36]) can be adapted to new areas, in this case the eSE HW platform. This improves the probability of success by minimising the necessary knowledge of the target eSE HW and FW. Though the information gathering phase of our attack revealed some unencrypted FW code that could be analysed for security vulnerabilities, this is not a mandatory step. Thus our attack methodology, using information leak oracles from the eSE logical interface, can be applied with no prior knowledge of FW contents.

Our attack demonstrates a complete compromise of the eSE integ-

rity, confidentiality and availability, thus all the main security goals for the eSE CC EAL (SG1-SG3) in Section 'CC EAL' are violated. A *single* software security vulnerability is enough, and a single attacker can with limited resources easily discover, and exploit, this vulnerability. Our research required nothing but access to commercially available (COTS) test devices and publicly available information. Vulnerability discovery and exploit development work were done by a single person in approximately one man-month's worth of time, with no special tools required. Our attack does not require physical access to the eSE and can therefore be performed remotely, over-the-air, needing only a privilege escalation vulnerability to be able to communicate via the logical interface of the eSE. This shows that the threat model from Section 'eSE Threat Model' does not match this eSE, as physical control is not required to perform our attack.

Restoring the eSE CC EAL (SG1-SG3) security goals (Section 'CC EAL') and trust through an eSE FW update seems infeasible, due to the lack of a root-of-trust and secure boot. The eSE can simply not validate its own code as there does not seem to be any on-board cryptographic integrity checks. The only integrity checks performed are simple CRC32 and SHA256 hash comparing. These hashes can be updated by an attacker and thus have no effect on security. In addition, integrity verification of an installed eSE FW cannot be performed by the host system (REE), as the black-box design of the eSE leaves no way to perform external validation of the installed eSE code. A stealthy backdoor implementation by an attacker could be very hard to reveal, making it challenging to detect if the eSE FW has been tampered with. Our discovered vulnerability thus completely breaks any forward trust in the eSE HW. Our results should make users question the validity of this CC EAL 5+ certification.

Furthermore, the exposure of the AES key used for encrypted FW updates of the eSE secure OS and boot images, makes updating this key using normal OTA FW updates difficult, if not impossible, as an attacker with knowledge of this AES key can decrypt any attempt of additional secret sharing with the eSE, such as replacement of the AES key. The effect is that Samsung can no longer exchange confidential information with the eSE HW through FW updates, exposing any encrypted parts of previous and future FW updates (Section 'Attack Capabilities and AES Key Exposure'). Samsung is of course free to change the key on newly manufactured devices, but this key cannot also be used in updated firmware for already shipped devices, as that would leak it.

To be able to regain trust in the eSE HW on already shipped devices, the authors believe the only secure option is a physical replacement of the eSE HW which is probably unreasonable.

V.7 Conclusions and Future Work

We have presented a remote attack on the S3K250AF eSE HW, using our discovered 0-day security vulnerability, exploitable through the logical interface. The attack contributes to the development of new DFA methods of affected devices. The attack was done by building on attacks from other security research areas and applying this to the physically separate eSE HW platform. The eSE HW is designed to withstand high level, and resourceful attackers, relying on a small code base, mostly unavailable to attackers, and resistance to side-channel attacks. Our vulnerability discovery and exploit development required no special tools or access, and the complete attack was developed with very limited resources, far from "state actor" capabilities. The attack enables an attacker to execute arbitrary shellcode to facilitate both reading and writing of both code and data, in both flash and RAM. This completely breaks all the eSE security goals stated in its CC EAL certification, and also enables an attacker to install hard-to-discover, persistent, backdoors and modifications to the eSE FW. Regaining trust in this eSE HW seems challenging, with physical replacement being the only realistically secure option. As this eSE HW is soldered to the PCB in the mobile device, such replacement is not trivial.

We conclude that one simple exploitable buffer overflow vulnerability enables full attacker takeover of the eSE HW, permanently.

Our attack facilitates digital forensic acquisition of devices in a *before-first-unlock* (BFU) state, with no prior knowledge of user credentials. With the aid of a more readily available privilege escalation vulnerability in Android, this becomes a complete solution for digital forensic acquisition of affected devices.

Our attack demonstrates the gap between the *intended* and *achieved* security level of this state-of-the-art eSE HW utilised by a major Android mobile phone vendor. The CC EAL 5+ certification gave no guarantee that the eSE was free from exploitable security vulnerabilities, only that some unidentified amount of effort had been made in an attempt to prevent them. We argue that the trust in the value certifications such as CC EAL provide, needs to be evaluated carefully on a case-by-case basis. Our attack also shows that such certifications should not discourage research into new DFA methods based on offensive techniques.

Our research is not exhaustive, and further attacks, including side-channel attacks, are left for future work. Further research is needed to reveal if other physical separation black-box solutions fall to similar attacks as the ones demonstrated in this research. A new testing methodology for logical interfaces of black-box HW can arise from our work, to potentially improve the CC Advanced methodical vulnerability analysis (AVA_VAN).

We believe our research further emphasises the challenges inherent in trusted computing, and that it demonstrates how fragile the *trust* put in such solutions is, whether this trust comes from certifications like CC EAL, or not.

Responsible Disclosure

Samsung is informed of the vulnerabilities discovered in this research and the authors have collaborated with Samsung to mitigate the risks they exposed. A patch for affected devices has been released, assigned with CVE-2020-28341 [57] and SVE-2020-18632 [58]. We thank Samsung for their professional cooperation.

Acknowledgements

The research leading to these results has received funding from the Research Council of Norway programme IKTPLUSS, under the R&D project Ars Forensica grant agreement 248094/O70.

V.8 References

[1] G. Alendal, S. Axelsson and G. O. Dyrkolbotn, 'Chip chop - smashing the mobile phone secure chip for fun and digital forensics,' *Forensic Science International: Digital Investigation*, vol. 37, p. 301 191, 2021, ISSN: 2666-2817. DOI: https://doi.org/10.1016/j.fsidi.2021.301191. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666281721000998.

[2] Venturebeat, *Apple vs. fbi: A timeline of the iphone encryption case*, http://venturebeat.com/2016/02/19/apple-fbi-timeline/ [Online; accessed 30-November-2020], 2016. [Online]. Available: http://venturebeat.com/2016/02/19/apple-fbi-timeline/.

[3] F. Focus, *Current challenges in digital forensics*, https://articles.forensicfocus.com/2016/05/11/current-challenges-in-digital-forensics/ [Online; accessed 30-November-2020], 2016. [Online]. Available: https://articles.forensicfocus.com/2016/05/11/current-challenges-in-digital-forensics/.

[4] D. Lillis, B. Becker, T. O'Sullivan and M. Scanlon, 'Current Challenges and Future Research Areas for Digital Forensic Investigation,' *The 11th ADFSL Conference on Digital Forensics, Security and Law (CDFSL 2016)*, May 2016.

[5] ARM, *Trustzone model*, [Online; accessed 12-Jan-2021], 2009. [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0301h/Chdebaee.html.

[6] G. Alendal, G. O. Dyrkolbotn and S. Axelsson, 'Forensics acquisition — Analysis and circumvention of samsung secure boot enforced common criteria mode,' *Digital Investigation*, vol. 24, S60–S67, 2018.

[7] A. P. Fournaris and G. Keramidas, 'From hardware security tokens to trusted computing and trusted systems,' in *System-Level Design Methodologies for Telecommunication*. Cham: Springer International Publishing, 2014, pp. 99–117, https://doi.org/10.1007/978-3-319-00663-5_6, ISBN: 978-3-319-00663-5. DOI: 10.1007/978-3-319-00663-5_6. [Online]. Available: https://doi.org/10.1007/978-3-319-00663-5_6.

[8] R. Anderson, 'Cryptography and competition policy: Issues with 'trusted computing',' in *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, https://dl.acm.org/doi/abs/10.1145/872035.872036, 2003, pp. 3–10.

[9] Intel, *Intel software guard extensions*, https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html [Online; accessed 06-October-2020], 2020. [Online]. Available: https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html.

[10] T. C. Group, *Tpm 1.2 main specification*, https://trustedcomputinggroup.org/resource/tpm-main-specification/ [Online; accessed 13-October-2020], 2020. [Online]. Available: https://trustedcomputinggroup.org/resource/tpm-main-specification/.

[11] M. Sabt, M. Achemlal and A. Bouabdallah, 'Trusted execution environment: What it is, and what it is not,' in *2015 IEEE Trustcom/BigDataSE/ISPA*, https://ieeexplore.ieee.org/abstract/document/7345265 [Online; accessed 13-October-2020], vol. 1, 2015, pp. 57–64.

[12] S. Mavrovouniotis and M. Ganley, 'Hardware security modules,' in *Secure Smart Embedded Devices, Platforms and Applications*. New York, NY: Springer New York, 2014, pp. 383–405, https://doi.org/10.1007/978-1-4614-7915-4_17, ISBN: 978-1-4614-7915-4. DOI: 10.1007/978-1-4614-7915-4_17. [Online]. Available: https://doi.org/10.1007/978-1-4614-7915-4_17.

[13] M. Vauclair, 'Secure element,' in *Encyclopedia of Cryptography and Security*. Boston, MA: Springer US, 2011, pp. 1115–1116, https://doi.org/10.1007/978-1-4419-5906-5_303 [Online; accessed 01-September-2020], ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_303. [Online]. Available: https://doi.org/10.1007/978-1-4419-5906-5_303.

[14] C. P. Pfleeger, *Security in computing*. Pearson Education India, 2009.

[15] L. Hatton, 'Reexamining the fault density-component size connection,' *IEEE Softw.*, vol. 14, no. 2, pp. 89–97, Mar. 1997, https://ieeexplore.ieee.org/abstract/document/582978/, ISSN: 0740-7459. DOI: 10.1109/52.582978. [Online]. Available: https://doi.org/10.1109/52.582978.

[16] A. Ozment and S. E. Schechter, 'Milk or wine: Does software security improve with age?' In *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, ser. USENIX-SS'06, https://www.usenix.org/legacy/events/sec06/tech/full_papers/ozment/ozment.pdf, Vancouver, B.C., Canada: USENIX Association, 2006.

[17] C. Criteria, *Publications*, https://www.commoncriteriaportal.org/cc/ [Online; accessed 01-September-2020], 2020. [Online]. Available: https://www.commoncriteriaportal.org/cc/.

[18] R. Mayrhofer, J. V. Stoep, C. Brubaker and N. Kralevich, 'The android platform security model,' *arXiv preprint arXiv:1904.05572*, 2019, https://arxiv.org/abs/1904.05572 [Online; accessed 04-September-2020].

[19] Samsung, *Samsung introduces best-in-class data security chip solution for mobile devices*, https://news.samsung.com/global/samsung-introduces-best-in-class-data-security\-chip-solution-for-mobile-devices [Online; accessed 26-August-2020], 2020. [Online]. Available: https://news.samsung.com/global/samsung-introduces-best-in-class-data-security-chip%5C%5C-solution-for-mobile-devices.

[20] Samsung, *Samsung, bsi, bundesdruckerei and telekom security partner to bring national id to your smartphone*, https://www.samsungmobilepress.com/pressreleases/samsung-bsi-bundesdruckerei-and-t-systems-partner-to\-bring-national-id-to-your-smartphone [Online; accessed 14-September-2020], 2020. [Online]. Available: https://www.samsungmobilepress.com/pressreleases/samsung-bsi-bundesdruckerei-and-t-systems-partner-to%5C%5C-bring-national-id-to-your-smartphone.

[21] B. für Wirtschaft und Energie, *Optimos 2.0*, https://www.digitale-technologien.de/DT/Redaktion/DE/Standardartikel/SmartServiceWeltProjekte/Wohnen_Leben/SSWII_Projekt_OPTIMOS_20.html [Online; accessed 14-September-2020], 2020. [Online]. Available: https://www.digitale-technologien.de/DT/Redaktion/DE/Standardartikel/SmartServiceWeltProjekte/Wohnen_Leben/SSWII_Projekt_OPTIMOS_20.html.

[22] M. Nemec, M. Sys, P. Svenda, D. Klinec and V. Matyas, 'The return of coppersmith's attack: Practical factorization of widely used rsa moduli,' in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, https://dl.acm.org/doi/abs/10.1145/3133956.3133969, 2017, pp. 1631–1648.

[23] D. Moghimi, B. Sunar, T. Eisenbarth and N. Heninger, 'TPM-FAIL:TPM meets timing and lattice attacks,' in *29th USENIX Security Symposium (USENIX Security 20)*, https://www.usenix.org/conference/usenixsecurity20/presentation/moghimi-tpm, 2020, pp. 2057–2073.

[24] A. Limited, *Securcore sc000*, https://developer.arm.com/ip-products/processors/securcore/sc000-processor [Online; accessed 26-August-2020], 2020. [Online]. Available: https://developer.arm.com/ip-products/processors/securcore/sc000-processor.

[25] NIST, *Cryptographic algorithm validation program*, https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/details?product=11431 [Online; accessed 26-August-2020], 2020. [Online]. Available: https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/details?product=11431.

[26] A. Limited, *Arm compiler toolchain linker reference*, https://developer.arm.com/documentation/dui0493/g/linker-command-line-options/--be8 [Online; accessed 26-August-2020], 2020. [Online]. Available: https://developer.arm.com/documentation/dui0493/g/linker-command-line-options/--be8.

[27] Samsung, *S3k250a/s3k232a/s3k212a 32-bit risc microcontroller for smart card with optional at1 secure libraries including specific ic dedicated software*, https://www.commoncriteriaportal.org/files/epfiles/anssi-cible-cc-2019_61en.pdf [Online; accessed 26-August-2020], 2020. [Online]. Available: https://www.commoncriteriaportal.org/files/epfiles/anssi-cible-cc-2019_61en.pdf.

[28] Samsung, *Samsung ese sdk*, https://developer.samsung.com/ese/overview.html [Online; accessed 26-August-2020], 2020. [Online]. Available: https://developer.samsung.com/ese/overview.html.

[29] commoncriteriaportal.org, *Certified products*, https://www.commoncriteriaportal.org/products/ [Online; accessed 26-August-2020], 2020. [Online]. Available: https://www.commoncriteriaportal.org/products/.

[30] A. N. de la Sécurité des Systèmes d'Information (ANSSI), *Rapport de certification anssi-cc-2019/61*, https://www.commoncriteriaportal.org/files/epfiles/anssi-cc-2019_61fr.pdf [Online; accessed 26-August-2020], 2019. [Online]. Available: https://www.commoncriteriaportal.org/files/epfiles/anssi-cc-2019_61fr.pdf.

[31] I. T. AG, I. Secure, N. S. G. GmbH and STMicroelectronics, *Security ic platform protection profile with augmentation packages*, https://www.commoncriteriaportal.org/files/ppfiles/pp0084b_pdf.pdf [Online; accessed 26-August-2020], 2014. [Online]. Available: https://www.commoncriteriaportal.org/files/ppfiles/pp0084b_pdf.pdf.

[32] C. Criteria, *Common criteria for information technology security evaluation - part 3: Security assurance components*, https://www.commoncriteriaportal.org/files/ccfiles/CCPART3V3.1R5.pdf [Online; accessed 26-August-2020], 2017. [Online]. Available: https://www.commoncriteriaportal.org/files/ccfiles/CCPART3V3.1R5.pdf.

[33] R. Anderson, M. Bond, J. Clulow and S. Skorobogatov, 'Cryptographic processors-a survey,' *Proceedings of the IEEE*, vol. 94, no. 2, pp. 357–369, 2006, https://ieeexplore.ieee.org/document/1580505.

[34] M. Bond and R. Anderson, 'Api-level attacks on embedded systems,' *Computer*, vol. 34, no. 10, pp. 67–75, 2001, https://ieeexplore.ieee.org/abstract/document/955101/.

[35] A. Bittau, A. Belay, A. Mashtizadeh, D. Mazières and D. Boneh, 'Hacking blind,' in *2014 IEEE Symposium on Security and Privacy*, https://ieeexplore.ieee.org/abstract/document/6956567/, IEEE, 2014, pp. 227–242.

[36] J. Lee, J. Jang, Y. Jang, N. Kwak, Y. Choi, C. Choi, T. Kim, M. Peinado and B. B. Kang, 'Hacking in darkness: Return-oriented programming against secure enclaves,' in *26th USENIX Security Symposium (USENIX Security 17)*, https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/lee-jaehyuk, 2017, pp. 523–539.

[37] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom and R. Strackx, 'Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution,' in *27th USENIX Security Symposium (USENIX Security 18)*, https://www.usenix.org/conference/usenixsecurity18/presentation/bulck, 2018, pp. 991–1008.

[38] O. Weisse, J. Van Bulck, M. Minkin, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, R. Strackx, T. F. Wenisch and Y. Yarom, 'Foreshadow-ng: Breaking the virtual memory abstraction with transient out-of-order execution,' 2018, https://lirias.kuleuven.be/2089352?limo=0.

[39] Y. Jang, J. Lee, S. Lee and T. Kim, 'Sgx-bomb: Locking down the processor via rowhammer attack,' in *Proceedings of the 2nd Workshop on System Software for Trusted Execution*, https://dl.acm.org/doi/abs/10.1145/3152701.3152709, 2017, pp. 1–6.

[40] A. Biondo, M. Conti, L. Davi, T. Frassetto and A.-R. Sadeghi, 'The guard's dilemma: Efficient code-reuse attacks against intel SGX,' in *27th USENIX Security Symposium (USENIX Security 18)*, https://www.usenix.org/conference/usenixsecurity18/presentation/biondo, 2018, pp. 1213–1227.

[41] M. Schwarz, S. Weiser and D. Gruss, 'Practical enclave malware with intel sgx,' in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, https://link.springer.com/chapter/10.1007/978-3-030-22038-9_9, Springer, 2019, pp. 177–196.

[42] A. Nilsson, P. N. Bideh and J. Brorsson, 'A survey of published attacks on intel sgx,' Tech. rep, Tech. Rep., 2020, https://arxiv.org/abs/2006.13598.

[43] G. Beniamini, *Qsee privilege escalation vulnerability and exploit (cve2015-6639)*, https://bits-please.blogspot.com/2016/05/qsee-privilege-escalation-vulnerability.html [Online; accessed 01-December-2020], 2016.

[44] G. Beniamini, *Trust issues: Exploiting trustzone tees*, https://googleprojectzero.blogspot.com/2017/07/trust-issues-exploitingtrustzone-tees.html [Online; accessed 01-December-2020], 2017.

[45] G. Beniamini, *War of the worlds - hijacking the linux kernel from qsee*, https://bits-please.blogspot.com/2016/05/war-of-worlds-hijacking-linux-kernel.html [Online; accessed 01-December-2020], 2016.

[46] Y. Chen, Y. Zhang, Z. Wang and T. Wei, *Downgrade attack on trustzone*, https://arxiv.org/abs/1707.05082, 2017. arXiv: 1707.05082 [cs.CR].

[47] D. Cerdeira, N. Santos, P. Fonseca and S. Pinto, 'Sok: Understanding the prevailing security vulnerabilities in trustzone-assisted tee systems,' in *2020 IEEE Symposium on Security and Privacy (SP)*, Los Alamitos, CA, USA: IEEE Computer Society, May 2020, pp. 1416–1432. DOI: 10.1109/SP40000.2020.00061. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP40000.2020.00061.

[48] S. Gunasekera, 'Rooting your android device,' in *Android Apps Security: Mitigate Hacking Attacks and Security Breaches*. Berkeley, CA: Apress, 2020, pp. 173–223, https://doi.org/10.1007/978-1-4842-1682-8_8, ISBN: 978-1-4842-1682-8. DOI: 10.1007/978-1-4842-1682-8_8. [Online]. Available: https://doi.org/10.1007/978-1-4842-1682-8_8.

[49] C.-Y. Chao, H. C. Su and C.-Y. Wu, 'Breaking samsung's root of trust: Exploiting samsung s10 secure boot,' *Black Hat USA*, 2020, https://www.blackhat.com/us-20/briefings/schedule/#breaking-samsungs-root-of-trust-exploiting-samsung-s-secure-boot-20290.

[50] Google, *Android security bulletins*, https://source.android.com/security/bulletin [Online; accessed 06-October-2020], 2020. [Online]. Available: https://source.android.com/security/bulletin.

[51] A. Limited, *The thumb instruction set*, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0210c/CACBCAAE.html [Online; accessed 26-August-2020], 2020. [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0210c/CACBCAAE.html.

[52] A. Gkaniatsou, F. McNeill, A. Bundy, G. Steel, R. Focardi and C. Bozzato, 'Getting to know your card: Reverse-engineering the smart-card application protocol data unit,' in *Proceedings of the 31st Annual Computer Security Applications Conference*, https://dl.acm.org/doi/pdf/10.1145/2818000.2818020, 2015, pp. 441–450.

[53] A. One, 'Smashing the stack for fun and profit,' *Phrack*, vol. 7, no. 49, Nov. 1996, http://www.phrack.com/issues.html?issue=49&id=14.

[54] Google, *File-based encryption*, https://source.android.com/security/encryption/file-based [Online; accessed 21-September-2020], 2020. [Online]. Available: https://source.android.com/security/encryption/file-based.

[55] Google, *Google git*, https://android.googlesource.com/platform/hardware/interfaces/+/refs/heads/master/weaver/1.0/IWeaver.hal [Online; accessed 21-September-2020], 2020. [Online]. Available: https://android.googlesource.com/platform/hardware/interfaces/+/refs/heads/master/weaver/1.0/IWeaver.hal.

[56] Google, *Android code search*, https://cs.android.com/android/platform/superproject/+/master:frameworks/base/services/core/java/com/android/server/locksettings/SyntheticPasswordManager.java [Online; accessed 21-September-2020], 2020. [Online]. Available: https://cs.android.com/android/platform/superproject/+/master:frameworks/base/services/core/java/com/android/server/locksettings/SyntheticPasswordManager.java.

[57] MITRE, *CVE-2020-28341*. Available from MITRE, CVE-ID CVE-2020-28341. Dec. 2020. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-28341 (visited on 20/01/2021).

[58] Samsung, *SVE-2020-18632*. November 2020, SVE-2020-18632. Nov. 2020. [Online]. Available: https://security.samsungmobile.com/securityUpdate.smsb (visited on 20/01/2021).

[59] ISO/IEC, *Iso/iec 7816-4:2020 identification cards — integrated circuit cards — part 4: Organization, security and commands for interchange*, https://www.iso.org/standard/77180.html [Online; accessed 26-August-2020], 2020. [Online]. Available: https://www.iso.org/standard/77180.html.

V.9 Appendix

APDU primer

ISO/IEC 7816 is an international standard for smart-cards. This standard is divided into 15 sub-parts specifying different aspects of smart-card char-

acteristics. ISO 7816-4 [59] describes security aspects, and commands to communicate with smart-cards. This includes a protocol specification, using what's called an "Application Protocol Data Unit" (APDU). An APDU defines the structure used to send/receive commands, and data. All communication is of the request–reply form, and is always initiated by the host. The smart-card never initiates communication. An APDU consists of a mandatory 4 byte header with the elements: *CLA*, *INS*, *P1* and *P2*, each one byte long. The CLA is referred to as the "class", often tied to the logical handler of the INS: the "instruction" or simply *command*. Inter-industry commands (INS) are defined in CLA `0`. ISO 7816-4 [59] defines a whole range of standard INS commands, all belonging to the CLA `0`. Vendors are free to implement vendor specific commands using for example CLA `0x80`. P1 and P2 are parameters for use in the specific (CLA,INS) pair and can thus be viewed as normal function parameters to the (CLA,INS) handler on the smart-card.

Additional APDU fields are optional, giving the possibility of appending any necessary data required by the specific (CLA,INS) pair: *Lc*, *DATA* and *Le*. Lc defines the size of appended DATA, and Le is the size of the expected data returned from the smart-card.

The smart-card is expected to reply with a valid return value, *SW* (Status Word). The SW is a 16-bit value consisting of bytes *SW1* and *SW2*, respectively. This reply is mandatory even if the requested (CLA,INS) pair is not implemented by the smart-card. SW is used to give informative error values back to the caller. Although the ISO 7816-4 defines some standardised error codes, these can be vendor defined for all proprietary CLA values. If the (CLA,INS) successfully completes the request, it is expected to return the SW value `0x9000` (SW1 = `0x90`, SW2 = `0x0`).

Tables

Table V.1: Analysed eSE FW images

Image filename	Image name	Size	Version int / string	SHA256sum
`k250a_00000009.img`	(whole file)	33280	0x47000101 / "191128145540"	638dad7cbf79ede847331516c118ff5b / d27002818ab35356987dcfa498e3262c
`k250a_00000009.img`	SNVM	33024	0x47000101 / "191128145539"	cc8349038e84313a3354459285deade2 bc1aa9ccb6eb3eef9c3a38689d46320b
`k250a_00000009_dev.img`	(whole file)	198808	0x100 / "191120090956"	9914566a795b081fee2040c1f530fba0 / 3ec1b57521d4dd4b1352a86600ffde5a
`k250a_00000009_dev.img`	BOOT*	20992	0x100 / "191120090947"	3e64599b94e36ed6746a7b15cb48859b a2ec0d419ce954e0b285e104ea711bc6
`k250a_00000009_dev.img`	CRPT	43928	0x100 / "191120090947"	37fcff13acda015611d6d0c3ec8576b4 52642e509b5bd799d83c716356ea8a57
`k250a_00000009_dev.img`	CORA*	33792	0x100 / "191120090947"	622c75c652d637775f92e9b37c03c2fc 0c00494c52d14fa61bd6229d05df4328
`k250a_00000009_dev.img`	CORB*	33792	0x100 / "191120090947"	71967cacde8d30d8f5597eba808e6d97 39025a4cc434c8c137ba42b4d9b4dedc
`k250a_00000009_dev.img`	SNVM	33024	0x100 / "191120090955"	aa82c67c9b545b35f6da05baebfb3549 1402cf27f99e27f81a12a1fdf0f028dc
`k250a_00000009_dev.img`	IWEA	33024	0x100 / "191120090955"	57c0a4e9033d0c9106bb0cde3af6ff82 5a79a4e8efc0de6b445ab7dd14e61c11

* Image encrypted with eSE embedded AES key + IV

Table V.2: eSE attack vectors: Exposed valid eSE APDU CLA and INS

CLA	INS	`libese-grdg.so` function	Comment
0x0	0x82	grdg_provisionAK	Provisioning
0x0	0xea	grdg_updateFW	FW Update (APP)
0x0	0xb1	grdg_getInfo / grdg_updateFW / grdg_updateCrypto	Retrieves various eSE info
0x0	0xf1	grdg_selfTest / grdg_IOTest	eSE on-board testing
0x0	0xf7	<not exposed>	Unknown
0x0	0xf8	<not exposed>	Unknown
0x0	0xf9	<not exposed>	Unknown
0x0	0xfa	grdg_updateFW	FW Update (CORA/CORB)
0x0	0xfb	grdg_updateFW / grdg_updateCrypto	FW Update (CRYPT)
0x0	0xfd	<not exposed>	FW Update (BOOT)
0x0	0xa4	grdg_snvmInit	APP (SNVM) Init
0x80	0xe4	grdg_factoryReset	APP (SNVM) Factory reset
0x80	0x84	grdg_getRandomNonce	APP (SNVM) Get 32 random bytes
0x80	0xb1	grdg_getAppInfo	APP (SNVM) Retrieve APP version info
0x80	0xdb	grdg_putCredential / grdg_putPersistentCredential	APP (SNVM) Put credential data
0x90	0xdb	grdg_putCredential / grdg_putPersistentCredential	APP (SNVM) Put credential data (large)
0x80	0xcb	grdg_getCredential / grdg_getPersistentCredential	APP (SNVM) Get credential data
0x80	0xee	grdg_deleteCredential / grdg_deletePersistentCredential	APP (SNVM) Erase credential data
0x1	0xa4	grdg_iweaverInit	APP (iWEAVER) Init
0x81	0xb1	<not exposed>	APP (iWEAVER) Retrieve APP version info
0x81	0x35	grdg_getWeaverConfig	APP (iWEAVER) Retrieve configuration
0x81	0xdb	grdg_writeWeaver	APP (iWEAVER) write credential slot (secret/challenge)
0x81	0xcb	grdg_readWeaver	APP (iWEAVER) read credential (secret) using challenge
0x81	0xdf	grdg_updateWeaver	APP (iWEAVER) Update all slots throttle data

Paper VI

Breaking Android Security by Abusing Implicit HW Trust

G. Alendal, 'Breaking Android Security by Abusing Implicit HW Trust,' In submission

This paper under submission and is therefore not included.

ISBN 978-82-326-6191-6 (printed ver.)
ISBN 978-82-326-5328-7 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

NTNU
Norwegian University of
Science and Technology