

BACHELOR THESIS

Bluetooth in Digital Mobile Forensics

Can pairing requests be found on Bluetooth devices?

Bachelor Applied Computer Science

Specialization Computer & Cyber Crime Professional

Academic Year 2019 - 2020

Student Nick Casier

Internal Mentor Daan Pareit (Howest)

External Promotor Kris Carlier (BeDefence)



howest.be

BACHELOR THESIS

Bluetooth in Digital Mobile Forensics

Can pairing requests be found on Bluetooth devices?

Bachelor Applied Computer Science

Specialization Computer & Cyber Crime Professional

Academic Year 2019 - 2020

Student Nick Casier

Internal Mentor Daan Pareit (Howest)

External Promotor Kris Carlier (BeDefence)



DEFENSIE
LA DÉFENSE

howest.be

Toelating tot bruikleen / Permission of use

De auteur (student) geeft de toelating deze bachelorproef voor consultatie beschikbaar te stellen en delen van de bachelorproef te kopiëren voor persoonlijk gebruik. Elk ander gebruik valt onder de bepalingen van het auteursrecht, in het bijzonder met betrekking tot de verplichting de bron uitdrukkelijk te vermelden bij het aanhalen van resultaten uit deze bachelorproef.

The author (student) gives permission to make this bachelor dissertation available for consultation and to copy parts of this bachelor dissertation for personal use. In all cases of other use, the copyright terms have to be respected, in particular with regard to the obligation to state explicitly the source when quoting results, findings and/or conclusions from this bachelor dissertation.

9/06/2020

Preface

When I chose to study Computer and Cyber Crime Professional at Howest in Bruges, my main motivation was that I wanted to know more about cybercrime. In my opinion, this is one of the most important domains in the I.T.-world. When security is correctly implemented, a lot of problems and issues can be avoided.

Over the years I learned that I.T.-security is a broad domain. During my education I came in touch with the different subdomains within I.T.-security. One of these domains intrigued me the most. This was Digital Forensics. Being able to find data that seemed lost or that seemingly does not exist fascinated me.

My fascination for digital forensics was the driving factor to choose a forensic internship at BeDefence.

This brings me seamlessly to the people I would like to thank.

In the first place I want to thank BeDefence and, in particular, Kris Carlier, Senior Captain (OF3) from the General Intelligence and Security Service of the CYBER division at BeDefence for the wonderful internship. Also the digital forensic team of the CYBER division at BeDefence, in particular Pieter Coeck, digital forensics investigator at BeDefence for guidance, tips and tricks, advice and information. It were difficult times due to COVID-19. He did everything in his power to provide me with extra test devices. Unfortunately, this was not possible and I was forced to perform all test on my own two personal devices. Also, I would like to thank my internal mentor, Ir. Daan Pareit of HoWest for the guidance during the writing of this thesis and proofreading it. Furthermore I would like to thank Kawtar Ben Dahman, Nick Foulon, Dylan Renner and Stijn Tassenoy for the support and friendship during our internship. As well as my best friend, Mathias Standaert for providing general help and advice and his creative contributions in helping to create figures as a visualisation for this dissertation. I want to thank Marc Staelens, for lending me the iPhone. And last but not least, I want to thank my parents and grandparents for always supporting me and believing in me. Unfortunately, during the internship and the writing of this thesis, I had to say goodbye to my grandmother. She will always be remembered.

Nick Casier, Oudenburg – Belgium, 9 June 2020

Abstract

In this document we describe our research to find a Bluetooth artefact. This artefact is the result of a pairing request between two Bluetooth enabled devices.

We focus on mobile devices (Android smartphones) and the differences between different (types of) devices. Furthermore, we dive deeper into the differences between a request that timed-out, got cancelled, denied or accepted.

What we set out to find, we found, and much more. We were able to derive from the artefact what actions took place. We also determined that the current standard procedure during a forensic confiscation causes data loss.

During this research, we discovered that *adb dumpsys* is a potential goldmine for mobile digital forensics on Android devices.

We discuss what the value of this discovery is and what the impact on the current state of mobile forensics is.

Key words: Digital Forensics – Mobile Forensics – Bluetooth – Artefact – Android – Smartphones – Pairing Request – adb

Samenvatting

In dit document gaan we opzoek naar het al dan niet bestaan van een bluetooth artefact. Dit artefact is het resultaat van een koppelingsverzoek tussen twee bluetooth apparaten.

We spitsen ons in dit onderzoek toe op mobile toestellen (Android smartphones) en wat de eventuele verschillen zijn tussen verschillende (types) toestellen. We vragen ons verder ook af of er een verschil is tussen een verzoek waarvan de bevestigingsperiode is verlopen, een verzoek dat werd geannuleerd, geweigerd of bevestigd.

We hebben een artefact gevonden, maar ook andere informatie die bij een forensisch onderzoek een meerwaarde bied. Deze informatie werd nog niet eerder beschreven. De artefact geeft een duidelijke weergave over het verloop van het koppelingsverzoek. We stelden ook vast dat de huidige standaard procedures bij een forensische inbeslagname dataverlies veroorzaken.

Tijdens ons onderzoek hebben we vastgesteld dat *adb dumpsys* een goudmijn aan forensisch interessante data bevat.

We beschrijven de waarde van deze vaststelling is en wat de impact op de huidige situatie in mobile digital forensics is.

Sleutelwoorden: Digital Forensics – Mobile Forensics – Bluetooth – Artefact – Android – Smartphones – adb – Koppelingsverzoek

Glossary

ADB	Android Debug Bridge
DF	Digital Forensics
DFIR	Digital Forensics and Incident Response
kbits/s	kilobit per second
MAC address	Media Access Control address. Unique identifier used in network communications.
Mbit/s	Megabit per second
OS	Operating System
PID	Process ID, unique identifier of a process running on a device
RAM	Random Access Memory
VM	Virtual Machine

Table of Content

Preface

Abstract

Samenvatting

Glossary

1	INTRODUCTION	18
1.1	RESEARCH QUESTION	18
1.2	WHAT IS BLUETOOTH?.....	18
1.3	WHAT IS DIGITAL FORENSICS?.....	22
1.4	WHAT IS MOBILE DIGITAL FORENSICS?.....	23
2	OVERVIEW OF THE TOOLS.....	25
2.1	ANDROID DEBUG BRIDGE (ADB)	25
2.1.1	Logcat.....	26
2.1.2	Dumpsys	27
2.2	TSURUGI	27
3	TEST DEVICES.....	28
4	TEST SETUP	30
5	DETERMINATION OF TEST PROCEDURE.....	33
6	THE VALUE OF <i>DUMPSYS</i>	36
7	COMMUNICATION BETWEEN IOS DEVICE AND ANDROID (AND VICE VERSA).....	37
7.1	TEST 1: IPHONE SENDS REQUEST – TIME-OUT	37
7.2	TEST 2: NEXUS 6 SENDS REQUEST – TIME-OUT	38
7.3	TEST 3: IPHONE SENDS REQUEST – CANCELLED.....	39
7.4	TEST 4: NEXUS 6 SENDS REQUEST – CANCELLED.....	40
7.5	TEST 5: IPHONE SENDS REQUEST – DENIED	41
7.6	TEST 6: NEXUS 6 SENDS REQUEST – DENIED	42
7.7	TEST 7: IPHONE SENDS REQUEST – INCORRECT PIN	44
7.8	TEST 8: NEXUS 6 SENDS REQUEST – INCORRECT PIN	45
7.9	TEST 9: IPHONE SENDS REQUEST – ACCEPTED.....	46
7.10	TEST 10: FORGET DEVICE	47
7.11	TEST 11: NEXUS 6 SENDS REQUEST – ACCEPTED.....	48
7.12	TEST 12: RECONNECT DEVICES.....	49
7.13	TEST 13: FORGET DEVICE AND ATTEMPT TO RECONNECT	50
8	COMMUNICATION BETWEEN TWO ANDROID DEVICES	52
8.1	TEST 1: ONEPLUS SENDS REQUEST – TIME-OUT.....	52
8.2	TEST 2: NEXUS 6 SENDS REQUEST – TIME-OUT	54
8.3	TEST 3: ONEPLUS SENDS REQUEST – CANCELLED	55
8.4	TEST 4: NEXUS 6 SENDS REQUEST – CANCELLED.....	57
8.5	TEST 5: ONEPLUS SENDS REQUEST – DENIED.....	59
8.6	TEST 6: NEXUS 6 SENDS REQUEST – DENIED	61
8.7	TEST 7: ONEPLUS SENDS REQUEST – INCORRECT PIN.....	63
8.8	TEST 8: NEXUS 6 SENDS REQUEST – INCORRECT PIN	65
8.9	TEST 9: ONEPLUS SENDS REQUEST – ACCEPTED	67

8.10	TEST 10: FORGET DEVICE	69
8.11	TEST 11: NEXUS 6 SENDS REQUEST – ACCEPTED.....	71
8.12	TEST 12: RECONNECT DEVICES.....	72
8.13	TEST 13: FORGETTING DEVICE AND ATTEMPTED TO RECONNECT.....	74
8.14	TEST 14: ACCEPTING THE RECONNECTION REQUEST	76
8.15	TEST 15: SENDING A FILE FROM THE ONEPLUS.....	77
8.16	TEST 16: SENDING A FILE FROM THE NEXUS 6.....	79
9	COMMUNICATION BETWEEN SMARTPHONE AND AUDIO PLAYER.....	81
9.1	BONDING THE AUDIO PLAYER WITH THE SMARTPHONE	81
9.2	RECONNECTING THE AUDIO PLAYER WITH THE SMARTPHONE	82
9.3	FORGETTING THE AUDIO PLAYER.....	82
10	COMMUNICATION BETWEEN AN ANDROID SMARTPHONE AND SMARTWATCH .	84
11	ARTEFACT LIFESPAN	85
11.1	<i>ACCOUNTS</i>	85
11.2	<i>CONTENT</i>	85
11.3	<i>BLUETOOTH_MANAGER</i>	85
12	CRITICAL REFLECTION ON THE TESTS.....	87
13	DISCUSSION OF TEST RESULTS.....	88
13.1	TIME-OUT OF A REQUEST.....	88
13.1.1	Master device.....	88
13.1.2	Slave device	88
13.2	CANCELLATION OF A REQUEST.....	88
13.2.1	Master device.....	88
13.2.2	Slave device	89
13.3	DENIAL OF A REQUEST.....	89
13.3.1	Master device.....	89
13.3.2	Slave device	89
13.4	ACCEPTANCE OF A REQUEST.....	90
13.4.1	Master device.....	90
13.4.2	Slave device	90
13.4.3	Remarks	90
13.5	FORGETTING A CONNECTION.....	90
13.6	FILE TRANSFER	91
13.7	GENERAL REMARKS	91
13.8	VALUE OF THE ARTEFACT	91
14	DIFFERENCES BETWEEN ANDROID VERSIONS.....	93
15	CONCLUSION.....	94
15.1	WHAT DID WE DISCOVER WITHIN THIS DISSERTATION?	94
15.2	WHAT IS THE IMPACT ON THE CURRENT STATE OF MOBILE DIGITAL FORENSICS?	94
16	BIBLIOGRAPHY	95
	OVERVIEW OF ATTACHMENTS.....	98
	Attachment 1: List of all services that can be issued with “ <i>dumpsys</i> ” on Nexus 6.....	99
	Attachment 2: output of the command “ <i>adb logcat grep – i bluetooth</i> ”	102
	Attachment 3: Generalization of the applied test procedure.....	103
	Attachment 4: Pop-up when sending a request on Android	106
	Attachment 5: Pop-up when receiving a request on Android	107
	Attachment 6: Pop-up on Nexus 6 when pin is incorrect	108

Attachment 7: Forgetting a paired device on Android	109
Attachment 8: Photo used in testing to send between devices	111
Attachment 9: Example script to extract all information from the <i>dumpsys</i> modules.....	112

1 Introduction

1.1 Research Question

The research question that will be tackled within this dissertation is the following: *“Can a pairing request be found on Bluetooth devices?”*

In this thesis we look to find an artefact that is created during the pairing process of two Bluetooth-enabled devices. The question whether such artefact exists arose during my internship at the Belgian Armed Forces (BeDefence).

We focus on Android devices since Android has the greatest market share globally. Depending on the source, around 75% to 85% of the smartphones worldwide use Android as an operating system. [1], [2], [3]

To answer this question, we first need to know what Bluetooth is and what the context of our research is. Since we focus on Android mobile devices, our context is mobile digital forensics. Mobile digital forensics differs from “classic” digital forensics as we clarify in the following sections of this chapter.

1.2 What is Bluetooth?

Bluetooth is a well-known open standard for short-range wireless communication technology. It's used in a wide variety of electronic devices. Those devices include business and consumer devices. Examples of those are mobile phones, laptops, tablets, infotainment system in cars, keyboards & mice, headsets and many more. Recently, medical devices and personal devices are implementing Bluetooth as well. Examples of these devices are smartwatches, heart rate monitors, music speakers, home appliances, fitness trackers and many more. [4]

Bluetooth offers a high level of flexibility and scalability between devices. The main advantages of Bluetooth technology are:

- **Eliminating cables.** By going wireless you can reduce the amount of cables that are needed. This can improve flexibility and the ease of use while also providing (in some cases) a cleaner aesthetic. A great example of this are Bluetooth keyboards and mice.
- **Ease of sharing.** Bluetooth is an easy and user-friendly way of sharing files, such as pictures, with another nearby device. For example, sharing a picture shot on a smartphone with a Bluetooth-enabled computer.
- **Synchronization with devices.** Bluetooth can be used to easily synchronise between different kinds of Bluetooth-enabled devices. For instance, Bluetooth is capable of providing automatic synchronisation of data. Think of the synchronisation between your mobile phone and the car's infotainment system or with a smartwatch.
- **Bluetooth tethering.** One of the lesser known features of Bluetooth is its capability to share a device's Internet connection via Bluetooth to another Bluetooth-enabled device that is unable to access the Internet.

As one can deduce from these examples, you may use Bluetooth more often in your daily life than you would expect. Since Bluetooth is widely used, the technology is actively developed and maintained. Since the founding of Bluetooth in 1989, it never stopped evolving. In 1998, the Bluetooth Special Interest Group was founded and Bluetooth was formally announced. Currently, the most recent version of Bluetooth is Bluetooth 5.2.

Here you can find an overview of the different versions and the most important differences compared to the previous version.

- **Bluetooth 1.0 and 1.0B**
 - First iteration of the Bluetooth specification in 1998

- **Bluetooth 1.1**
 - Adaptation of the IEEE Standard 802.15.1-2002
 - Added possibility of non-encrypted channels
 - Received Signal Strength Indicator support

- **Bluetooth 1.2**
 - Faster Connection and discovery
 - Adaptive frequency-hopping spread spectrum to avoid use of overcrowded frequencies
 - Higher transfer speeds up to 721 kbits/s
 - Adaptation of the IEEE Standard 802.15.1-2005

- **Bluetooth 2.0 + EDR**
 - Introduction of Enhanced Data Rate (EDR) to improve transfer speeds up to 2.1 Mbit/s

- **Bluetooth 2.1 + EDR**
 - Adopted by the Bluetooth Special Interest Group (SIG) on the 26th of July 2007
 - Support for secure simple pairing (SSP)

- **Bluetooth 3.0 + HS**
 - Adopted by the Bluetooth SIG on the 21st of April 2009
 - Theoretical maximum transfer speed up to 24 Mbit/s over a collocated 802.11 link (for Bluetooth 3.0 + HS (high speed), not mandatory for the Bluetooth 3.0 standard)
 - Introduction of L2CAP Enhanced modes
 - Enhanced power control

- **Bluetooth 4.0**
 - Introduced on the 30th of June 2010 as Bluetooth Smart
 - Supports Classic Bluetooth (v1.x & v2.x), Bluetooth High Speed (v3) and Bluetooth Low Energy (BLE) products

- **Bluetooth 4.1**
 - Introduced on the 4th of December 2013
 - Incremental software update to v4.0 instead of hardware update
 - Mobile Wireless Service Coexistence Signalling
 - Train Nudging and Generalized Interlaced Scanning
 - Low Duty Cycle Directed Advertising
 - L2CAP Connection Oriented and Dedicated Channels with Credit-based Flow Control
 - Dual Mode and Topology

Bluetooth in Digital Mobile Forensics

Introduction

- LE Link Layer Topology
- 802.11n PAL
- Audio Architecture Updates for Wide Band Speech
- Fast Data Advertising Interval
- Limited Discovery Time

- **Bluetooth 4.2**
 - Introduced on the 2nd of December 2014
 - Mainly focused for the Internet of Things (IoT)
 - Low Energy Secure Connection with Data packet Length Extension
 - Link layer privacy with Extended Scanner Filter Policies
 - Internet Protocol Support Profile version 6 (IPSPv6)

- **Bluetooth 5.0**
 - Introduced on the 6th of December 2016 as Bluetooth 5
 - BLE can now burst up to 2 Mbit/s at the expense of range
 - Support for location navigation of BLE connections
 - Support for Slot Availability Mask (SAM)
 - BLE Long Range
 - High Duty Cycle Non-Connectable Advertising
 - BLE Advertising Extensions
 - Removal of Park State

- **Bluetooth 5.1**
 - Introduced on the 21st of January 2019
 - Support for Angle of Arrival (AoA) and Angle of Departure (AoD) for use in location and tracking
 - Addition of Advertising Channel Index
 - Addition of GATT Caching
 - Some minor enhancements compared to Bluetooth 5
 - Support for Models and Mesh-based model hierarchy
 - Removal of Unit Keys

- **Bluetooth 5.2**
 - Introduced on the 6th of January 2020
 - Support for BLE Audio
 - Support for Enhanced Attribute Protocol (EATT), improved version of the Attribute Protocol (ATT)
 - BLE Power Control
 - BLE Isochronous Channels

The Bluetooth technology resides in the second layer of the OSI-model, the data link layer. It's a protocol that operates in the 2.4GHz range. When a connection between two Bluetooth enabled devices is conducted, a three step progressive process is initiated. This process is called the Bluetooth handshake. This process is visualised in Figure 1 - Bluetooth Handshake.

Bluetooth in Digital Mobile Forensics
Introduction

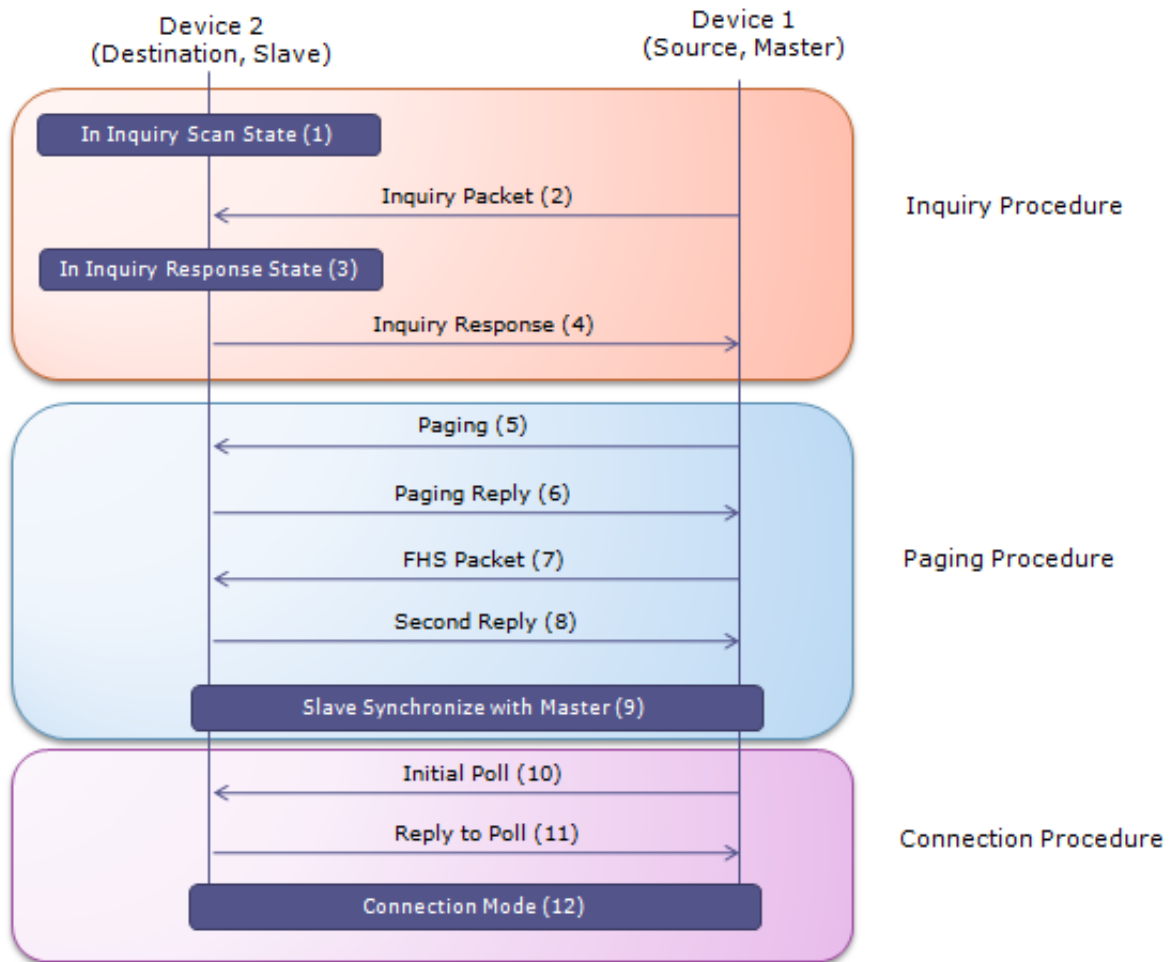


Figure 1 - Bluetooth Handshake [5]

In the first step, the **inquiry**, the two Bluetooth devices are unknown to each other. One device must run an inquiry in an attempted to discover the other. One device sends an inquiry or pairing request. The other device will respond to this request with some basic information, like for example its MAC address and name.

The second step, the **paging**, is the process of connecting the two devices.

The third and last step is when the two devices are **connected**. While connected, a device can be either actively participating (e.g. when transferring files) or in a low power mode (e.g. using a Bluetooth device to dynamically lock a pc when the device is out of range).

Data, more specific, control data that is created by using the Bluetooth technologies can provide important information when a digital forensic investigation is conducted.

Next, we need to understand what a digital forensics investigation is.

1.3 What is Digital Forensics?

When a judicial investigation is conducted, building a timeline of what the person of interest did during a certain timeframe is crucial. The computer and its data of that person can provide crucial information when this person is accused of a crime. The research that is committed on the electronic devices in such situations is called digital forensics.

These situations are the most common use cases for digital forensics, but it's more than that. Digital Forensics (DF) is defined as the process of preservation, identification, extraction and documentation of digital evidence. It's a subdomain within computer science that uses scientific investigatory techniques with the goal to preserve data.

Companies rely on forensic investigators when they have become victim to a cyber-attack. A forensic investigation is conducted to discover the origin of the attack, preserve non-contaminated data and recover contaminated or lost data.

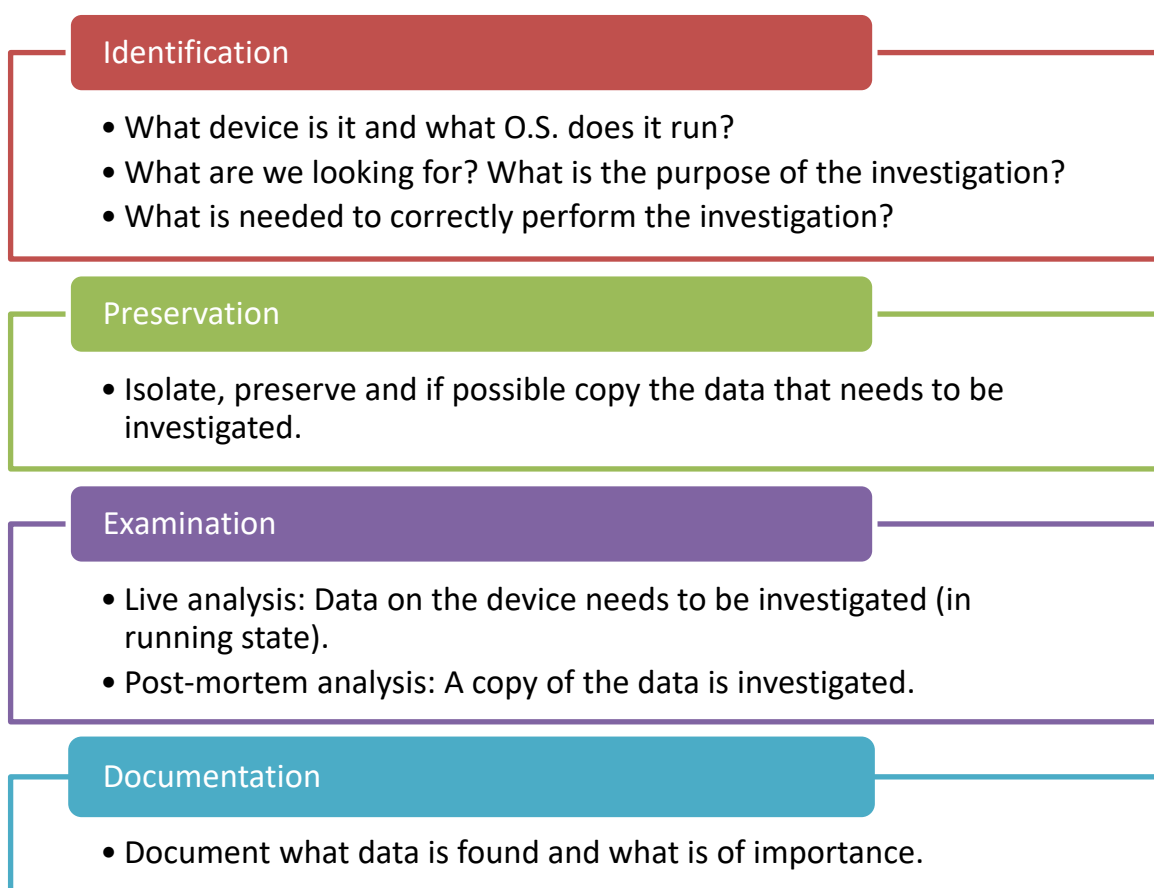


Figure 2 - Four core fundamentals of Digital Forensics

As one can derive from Figure 2 - Four core fundamentals of Digital Forensics, there are four core fundamentals in Digital Forensics. These are equally important in a forensics investigation.

The first stage in a digital forensic investigation is to **identify** the device that needs to be researched and its Operating System (O.S.). We need to know the goal of the investigation. Are we trying to recover data, looking for traces left on the device caused by malware or human interaction or do we need to find evidence to be used in court? All these factors need to be taken into account before starting the investigation. It's important that we know the context. For example, when conducting an investigation after a cyber-attack, we want to know how the malware entered the system and how it spread. When compared to an investigation in a legal context (for example to the computer of a criminal), we are more interested in what interactions the person in question had, what sites he/she has visited or what files he/she has created/deleted/alterd etc...

In both cases we need to make sure that the forensic investigator can perform the investigation and leaves as little traces on the device as possible. We need to decide what precautionary measurements have to be considered for the next stage.

The second stage is the **preservation** of data. The data has to be isolated, so it cannot be altered (or spread in case of malware). We want to save unsaved data (for example data that is present in RAM). This can vary from unsaved documents to running programs. In most cases, a dump of the memory and a full system copy can be made. This copy is used to be copied as many times as necessary without needing to access the original system and as a reference. These precautionary measurements are enforced to prevent potential changes to the data. During the investigations, one of the copies is being used, combined with a write blocker, to prevent potentially altering the data. Often, after each step in the investigation, the copy will be compared to a copy which is only intended for reference. This copy is also referred to as the *golden copy*. This is done to prevent and document potential changes to the used copy.

As hinted in the previous stage, the next step is the **examination** of the data. This can be done on a copy (post-mortem analysis) or on a live system (for example to analyse the behaviour of malware or an artefact). During this stage we start gathering proof of what happened. Commonly screenshots and screen recordings are being made to aid in the next stage.

The last stage of a digital forensic investigation is the **documentation** of the findings. The data that has been found, needs to be described and considered what is of importance depending on the context. With this data, a timeline can be created that can help a legal investigation or learn the behaviours of the malware and how it managed to infect the system. The documentation contains the proof that has been discovered in the previous stage and a detailed write-up of what has been found.

It is intended that further steps can be based on this documentation. In a judicial investigation, the police officers can use the documentation to identify the criminal. In the case of malware, improvements can be made based on the documentation to counteract an attack in the future.

1.4 What is Mobile Digital Forensics?

Mobile Digital Forensics is Digital Forensics that is dedicated to mobile devices such as smartphones and tablets. It is based on the same four core fundamentals but adds a few extra challenges. We focus within this dissertation on Android Digital Forensics.

Every manufacturer can give Android, and the filesystem that it uses, its own flavour. This means that even two devices running the same Android version but that are from another

Bluetooth in Digital Mobile Forensics

Introduction

brand could have different file structures. For example, pictures shot on an Android device are by default stored in a folder called DCIM. Manufacturers can change the folder where the camera app saves the shot images to, for example, Pictures. The folder DCIM could still be present but won't contain (all) the images. Since most apps also have their own home directory, the file structure on one Android device can be completely different to that from another Android device that seemed identically at first sight.

Since Android is also based on Unix, only root¹ has access to every file and partition on the device. This means that the forensic investigator needs to become root to access everything. To become root on an Android device, it has to be 'rooted'. Since most people do not root their phones, this means that the investigator needs to root the phone, thus extensively altering the device. This is contradictory to a correct digital forensics' investigation.

A user of an Android device has its own user account. With this account, the user can do everything he/she would want to do with the device. This is comparable to other computer systems where some sort of account management is in place. In the Unix world it is a good practice to use the root account only for a limited number of initial configurations. Afterwards, *su* and *sudo* can be used to perform tasks that need a higher privilege than a normal user has. One of the motivations comes from a security perspective. When an error is made while being root, the possibility exists that the entire system is bricked². This is why an Android devices' user is normally not root, since Android is based on a variant of Unix. Rooting the device can be defined as the process that lets you access all the settings, sub-settings and files of your phone. [6] This could potentially have severe consequences when by accident malware has been downloaded onto the phone.

For a DF investigation, these obstacles need to be overcome. In modern versions of Android, a lot can be overcome by enabling developer options (see chapter 4 – Test Setup). This can be done without rooting the device. Every Android user can enable these developer options.

In recent years, there have been tools developed for forensic investigators. These tools help to overcome the difficulty associated with the filesystems on Android devices. Each tool has his own strengths and weaknesses. Due to the large variety of structures, the results of the devices can also vary depending on the brand of the Android device that is being investigated.

Since these challenges require a specific knowledge compared to, for example, Windows digital forensics, mobile digital forensics is considered to be a separate branch within digital forensics. The fast evolution of mobile operating systems and the changes to the stock Android system make it difficult for forensic researchers to extensively test their tools on a variety of, let alone all, mobile devices. Windows, for example, does not have these differences. The Windows O.S. does not evolve as fast as Android and manufacturers do not alter the Windows O.S. as they do with Android. This means that the tools used to perform a forensic investigation on Windows are easier to be extensively tested and validated.

¹ **Root:** also called root-user, sometimes referred to as superuser. This user account has the highest privileges/access rights on a system and has access to all files and commands available on a system.

² **Bricked:** To render your device useless, as useless as a brick. Usually the result of tampering with the insides and doing irreversible damage. Bricking your hardware leaves you with a new paperweight. Can be the end effect of a faulty flash or firmware update, a modification gone bad for example. Bricked refers to *any* hardware that is unable to start up due to bad software. This could be caused by loss of necessary files due to a trojan that deletes necessary files for example. [40]

2 Overview of the Tools

This chapter describes the tools that are used during testing.

In this overview you can find the tools that were used to perform the test. All tools were up-to-date at the moment of testing.

Tool	Function	Version
Android Debug Bridge	Command-line tool for communication with Android Devices	29.0.5
Tsurugi LAB	Digital Forensics O.S. (ran as a VM)	LSB Version: core-9.20160110ubuntu0.2-amd64:core-9.20160110ubuntu0.2-noarch:security-9.20160110ubuntu0.2-amd64:security-9.20160110ubuntu0.2-noarch Release: 2019.2 Codename: Lastbamboo
VMware Workstation 15.5 Pro	Virtualisation software	15.5.1 build-15018445

2.1 Android Debug Bridge (adb)

Android Debug Bridge (adb) is a versatile command-line tool that lets you communicate with a device. The adb command facilitates a variety of device actions, such as installing and debugging apps, and it provides access to an Unix shell that you can use to run a variety of commands on a device. [7]

Adb is a client-server program. It has three components that build up the entire process. First there is the client. This is the computer that is used to connect with the Android device. In most cases, this connection is via USB. The second component is the daemon. This is the service that is running on the computer (client) as well as on the Android device itself. It allows the Android device to accept and execute the commands that are given via the client. The third part of adb is the server. This manages the communication between the client and the daemon. It sends the commands, that are given on the client, to the daemon. By default, the server runs on port 5037 as a background process on the client.

```
nick@XPS-Tsurugi:~$ adb devices -l
* daemon not running; starting now at tcp:5037
* daemon started successfully
List of devices attached
ZX1G5245DH          device usb:1-1 product:shamu model:Nexus_6 device:shamu transport_id:1
```

Figure 3 - Start-up of adb client-server program. Any adb command can be used to start the tool.

The two commands of adb we will be using are *logcat* and *dumpsys*.

2.1.1 Logcat

The logcat command-line tool dumps a log of the system. It returns more than just system messages. It can return full stack traces when a device throws an error or show the messages that have been written from an app with the *log* class.

Since logcat dumps all the logs of a device, this can be a bit overwhelming. Luckily logcat has some built-in features to help control the output.

There are filter expressions. These restrict the log output based on tags-priority combinations that you are interested in. They follow the format *tag:priority ...* where *tag* indicates the tag of interest and *priority* indicates the minimum level of priority to report for that tag. An example of a filter expression is: `adb logcat ActivityManager:I MyApp:D *:S`. This filter expression suppresses all log messages except those with the tag “ActivityManager”, at priority “Info” or above, all log messages with tag “MyApp”, with priority “Debug” or above. The last element of the expression, `*:S`, sets the priority for all other tags to silent. By doing this, the output is restricted to what has been specified.

Another way of controlling the output of the logcat command, is by formatting the log output. Log messages also contain a number of metadata fields. By modifying the output format for these messages, you can display specific metadata fields. This can be done as followed: `[adb] logcat [-v < format >]`. The supported output formats are:

- *brief*: Display priority, tag, and PID (process ID) of the process issuing the message.
- *long*: Display all metadata fields and separate messages with blank lines.
- *process*: Display PID only.
- *raw*: Display the raw log message with no other metadata fields.
- *tag*: Display the priority and tag only.
- *thread*: A legacy format that shows priority, PID, and TID (thread ID) of the thread issuing the message.
- *threadtime* (default): Display the date, invocation time, priority, tag, PID, and TID of the thread issuing the message.
- *time*: Display the date, invocation time, priority, tag, and PID of the process issuing the message.

There are also other log buffers that can be accessed, the so-called alternative log buffers. The Android logging system keeps multiple circular buffers for log messages. Not all log messages are sent to the default circular buffer. By issuing the *logcat* command with the *-b* option, you can request viewing an alternate circular buffer. This can be done as followed: `[adb]logcat [-b < buffer >]`. The alternative buffers are:

- *radio*: View the buffer that contains radio/telephony related messages.
- *events*: View the interpreted binary system event buffer messages.
- *main*: View the main log buffer (default). This does not contain system and crash log messages.
- *system*: View the system log buffer (default).
- *crash*: View the crash log buffer (default).
- *all*: View all buffers.
- *default*: Reports *main*, *system*, and *crash* buffers.

2.1.2 Dumpsys

The *dumpsys* command-line tool runs on the Android device itself, and provides information about system services. By calling this tool through adb, you can get diagnostic output for all system services running on a connected device. You can run the command *adb shell dumpsys* to get a diagnostic output for all system services. Since this output is unmanageable large, you can specify what service you want to examine by including it in the command. Since we are interested in Bluetooth, we can issue the command *adb shell dumpsys bluetooth_manager*. A full list of all possible services that can be called can be found in Attachment 1: List of all services that can be issued with “*dumpsys*” on Nexus 6.

2.2 Tsurugi

Tsurugi is an Digital Forensics and Incident Response (DFIR) Linux Distribution based on Ubuntu 64-bit LTS 16.04. As other forensics O.S., it has a plethora of built-in tools for forensic investigations. Tsurugi is a relative new O.S. and is still actively developed and improved. The team behind Tsurugi is, as they say themselves, a bunch of Backtrack and Deft Linux veterans united by the idea of developing a new DFIR O.S. [8].

There are multiple editions of Tsurugi. The version that is used here of Tsurugi Linux is Tsurugi LAB. This edition is intended to be used as a standalone OS. For live disk acquisitions there is Tsurugi Acquire. Bento is a portable toolkit designed for live forensics and incident response.

As good practice we used Tsurugi as a virtual machine.



Figure 4 - TSURUGI Linux – the sharpest weapon in your DFIR arsenal [9]

3 Test Devices


This chapter gives an overview of the devices used during the tests that are performed.

All devices are running the most recent software versions that are available at the moment of performing the test.

The Nexus 6 is connected to the VM via the Dell XPS over USB.

To simulate a real-life scenario, the two Android devices did not receive a factory reset before testing. The OnePlus is my daily driver and the Nexus 6 my previous phone. Since they did not receive a factory reset, we discovered some other interesting artefacts that we will touch later on.

The iPhone 6 Plus was leant to me and received a full factory reset for privacy reasons.

Device	Device name	Function	O.S. Version
	Dell XPS 15 9570	Host for Tsurugi VM and used to connect with the OnePlus 7T Pro via ADB	Windows 10 Home 1909 Build 18363.657
	OnePlus 7T Pro 48:01:C5:86:27:CE	Test subject with Bluetooth version 5.0 No factory reset	Android 10 Build-number Oxygen OS 10.0.7.HD01BA Android security patch January 2020
	Motorola Nexus 6 44:80:EB:F0:AD:8F	Test subject with Bluetooth version 4.1 No factory reset	Android 7.1.1 (Nougat) Build-number: N6F27M Android security patch October 2017


Bluetooth in Digital mobile Forensics

Test Devices

	<p>Apple iPhone 6 Plus BC:4C:C4:EE:11:9F</p>	<p>Test subject with Bluetooth version 4.0 Factory reset</p>	<p>iOS 12.4.5 (16G161)</p>
	<p>Fossil Q Marshal FC:45:96:6A:35:65</p>	<p>Test subject with Bluetooth version 4.1 No factory reset</p>	<p>Wear OS by Google 2.14 Android security patch February 2019</p>
	<p>UUV Airdot wireless in-ear headphones CA:71:06:FE:ED:A3</p>	<p>Test subject with Bluetooth version 5.0</p>	<p>N/A</p>

Bluetooth in Digital mobile Forensics

Test Setup

	<p>Orico SHC-U3 USB HUB</p>	<p>Hub to connect devices to laptop</p>	<p>N/A</p>
---	--	---	------------

4 Test Setup

In this chapter, we describe our setup and what preparations have to be made in order to do our testing.

We assume that all required tools and programs (that are not specific to our tests) are already installed.

Installing Tsurugi is easy. We recommend to assign at least 4GB RAM and, if possible, to assign 4 cores to the VM. We noticed a huge performance benefit when the VM has access to these resources. You'll need at least 25GB storage to be able to install Tsurugi. Once you start the iso installation file, select the first option, *TSURUGI Linux Live (GUI mode)*, and wait until you land on the desktop. On the desktop you can double click on "Install TSURUGI" and follow the steps.

Android Debug Bridge is already pre-installed on Tsurugi, so we do not have to install it.

To be able use adb on the mobile devices, we need to enable developer options so we can enable USB-debugging. Developer options can be enabled by tapping the build number (see figure 4) of the O.S. on the Android device seven (7) times. The build number can be found under settings, about the device.

If successful, you'll see *Developer options* in the settings menu (see figure 5).

In the Developer option we need to enable USB-debugging, as visible in figure 6.0.

The last step is to enable Bluetooth and set the device as visible. Some devices require you to tap a button to make the device visible.

Bluetooth in Digital mobile Forensics
Test Setup

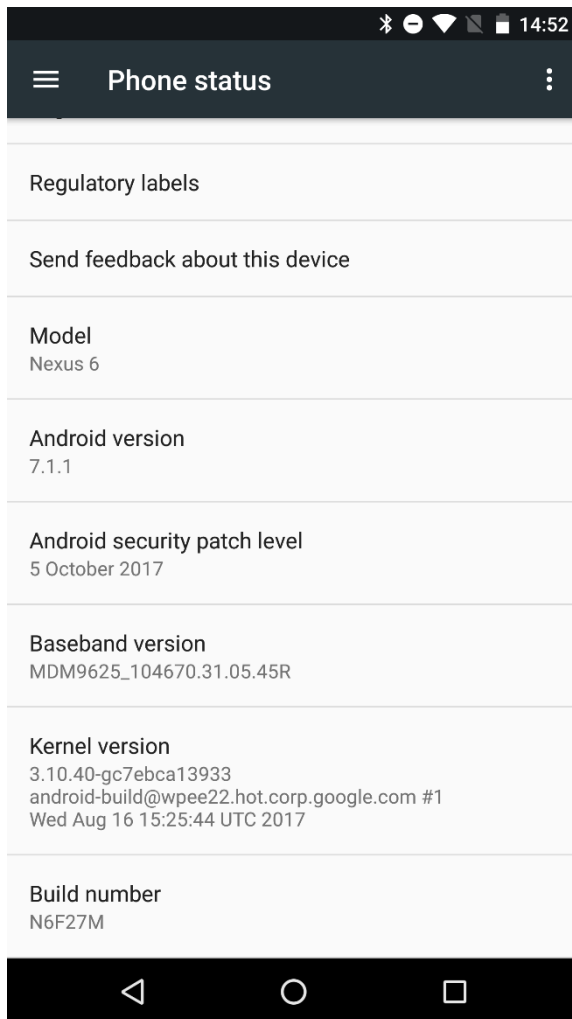


Figure 5 - About Phone

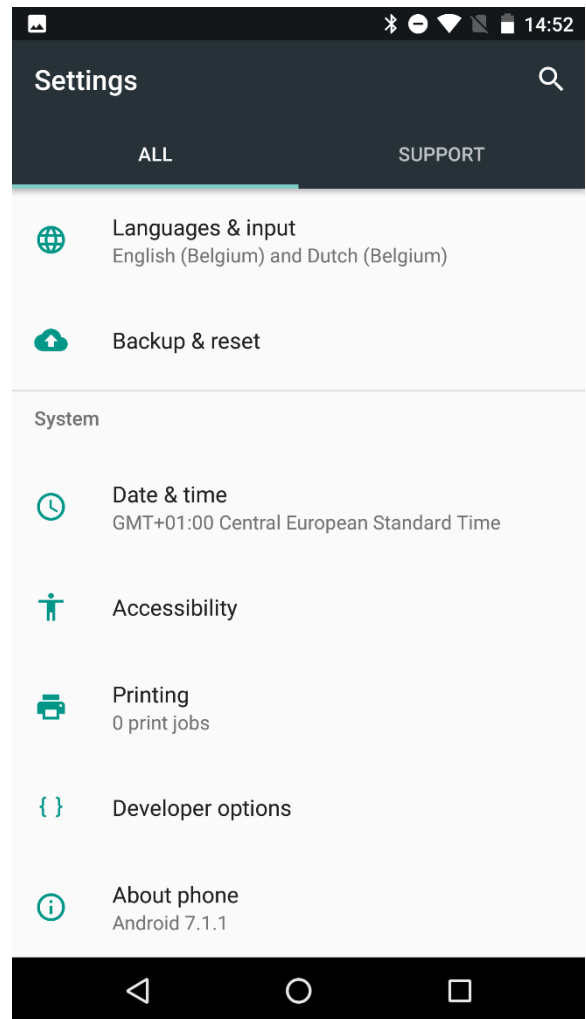


Figure 6 - Successful enabled Developer options

Bluetooth in Digital mobile Forensics
Test Setup

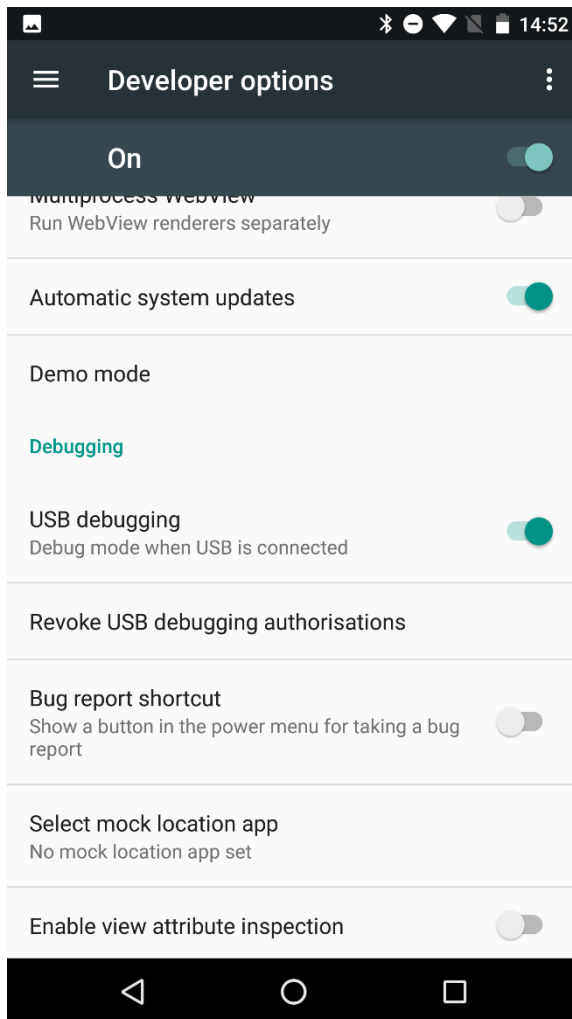


Figure 7 - Enable USB debugging under the debugging section under Developer options

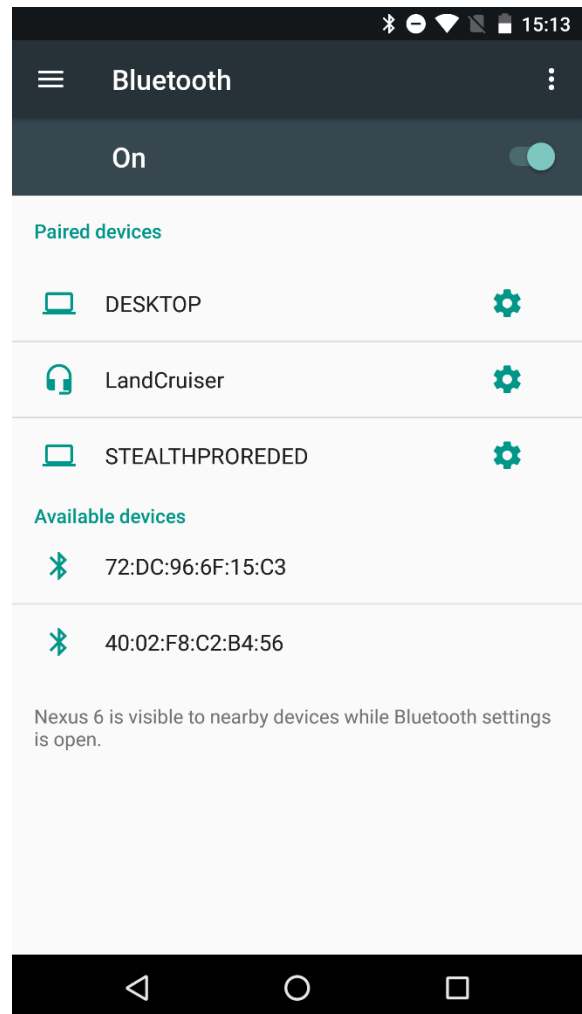


Figure 8 - Bluetooth settings

5 Determination of Test Procedure

This chapter describes the procedure we went through to determine our test method.

We started from the simple question that came up: Is there an artefact to be found on a mobile device when a Bluetooth pairing request has been send to it? Yes or No?

Since we did not know if any artefact existed, we started with a live forensic research. We hoped to find a change in a memory state or see at least a trace of where an internal process has been called.

The first thing that came to mind was looking into the RAM what is happening when a request is received. We used the OnePlus to send a request to the Nexus 6 and monitored the RAM using the adb command `adb shell dumpsys meminfo`. Unfortunately, we could only see that the Bluetooth process was active (com.android.bluetooth).

```
8,927K: Persistent Service
8,927K: com.android.bluetooth (pid 1409)
```

Figure 7 - Bluetooth service active visible in memory

This information did not indicate the existence of a potential artefact. If we can't see changes in the memory, where could we find potential changes in processes? Logcat can show us messages created by processes in the Android O.S.. We ran the command `adb logcat | grep -i bluetooth`. We started off by looking at the default information that logcat provides, but since we are only interested in the information from the Bluetooth process, we used grep to extract only the lines where "Bluetooth" was mentioned. The output of this command can be seen in Attachment 2: output of the command "`adb logcat | grep -i bluetooth`". There are a few interesting things to be seen in the output. We can, for example, see that there was a Bluetooth connect broadcast send by the OnePlus. We can see that this triggered a change in bond state.

The message of the *BluetoothEventManager*, notifying us that the bond state has been changed, made us wonder if we could access this manager and where we could find the bond state. Since this is a system service and `dumpsys` can return the various system services, we checked what the service was we needed to call in our command. As hinted in section 2.1.2, we can issue the command `adb shell dumpsys Bluetooth_manager`. We discovered that the name of the service we needed to call was that by calling the list of services that can be called using `adb shell dumpsys -l` as shown in Attachment 1: List of all services that can be issued with "`dumpsys`".

We issued the command `adb shell dumpsys Bluetooth_manager` and studied the output. We saw that there was a section "Bond Events". This seemed interesting since we saw via logcat that a change in bond state was made when we had sent our pairing request. We ran the command again and pushed the output to a text file. We issued a new request from the OnePlus to the Nexus 6, ran the command again and pushed the output to a new text file (`adb shell dumpsys bluetooth_manager > afterRequest.txt`). We compared the two files with the help of diff, a built-in Linux command-line tool, to see if there were any changes made to the section "Bond Events" (`diff -y beforeRequest.txt afterRequest.txt`). As you can see in Figure 9 – Output of the command `diff -y beforeRequest.txt afterRequest.txt`, we can clearly see changes made to the bond state, we can see at what time the bond state has been changed and the MAC-address of the devices that was communication with our device.

Bluetooth in Digital mobile Forensics

Determination of Test Procedure

Have we found an artefact?

```

Bond Events:
Total Number of events: 0

Bond Events:
| Total Number of events: 2
> Time      BD_ADDR      Function      State
> 13:11:12.579 bc:4c:c4:ee:11:9f bond_state_changed BOND_
> 13:12:16.806 bc:4c:c4:ee:11:9f bond_state_changed BOND_
    
```

Figure 8 - Output of the command diff -y beforeReques.txt afterRequest.txt

As we can derive from Figure 9, this does seem to be an artefact we were looking for. We take a closer look to the before and after of the section Bond Events.

```

Bond Events:
Total Number of events: 0
    
```

Figure 9 - before the request

```

Bond Events:
Total Number of events: 2
Time      BD_ADDR      Function      State
13:11:12.579 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING
13:12:16.806 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_NONE
    
```

Figure 10 - after the request

We can clearly see the number of *Bond Events*, at what time these events took place in the format HH:MM:SS.SSS, MAC-address of the device who is communicating with the test subject, the called function and the state. The seconds are decimal noted, which means that the timestamp is accurate up to 0.001 second, thus one millisecond. The displayed time is the local time of the device.

In the upcoming chapters, we investigate the discovered artefact and its behaviour in different scenarios. Each test case is organized according to the same clear structure and was performed at least three times. Each one has the scenario, the devices that were used, the procedure of testing, monitored devices, how the devices were monitored, what was monitored, the state of the artefact at the start of the test case, what we expected (with pseudocode to clarify our hypothesis), what the behaviour of the artefact was during the test (where applicable), after the test, what we can see in the artefact and how this translate to what we expected. A generalisation of the test procedure can be found in Attachment 3: Generalization of the applied test procedure.

In the next chapters, we discuss what the value of *dumpsys* is, we clarify this with some examples. We use the aforementioned artefact as a testcase. The testcases are divided over five chapters:

- Chapter 6 - The Value of *dumpsys*
We discuss what the value of *dumpsys* is for the current state of mobile DF.
- Chapter 7 - Communication between iOS device and Android (and vice versa)
Various scenarios of Bluetooth communication between an iPhone and the Nexus 6 where we discuss the behaviour of the found artefact.

Bluetooth in Digital mobile Forensics

Determination of Test Procedure

- Chapter 8 - Communication between two Android devices
Various scenarios of Bluetooth communication between the Nexus 6 and the OnePlus where we discuss the behaviour of the found artefact.
- Chapter 9 - Communication between smartphone and audio player
Investigation of the behaviour of the found artefact when an Android device interacts with an audio player.
- Chapter 10 - Communication between an Android smartphone and smartwatch
We investigate if there is a different behaviour when we connect with a known smartwatch.
- Chapter 11 - Artefact lifespan
Discussion about the lifespan of the found artefact.

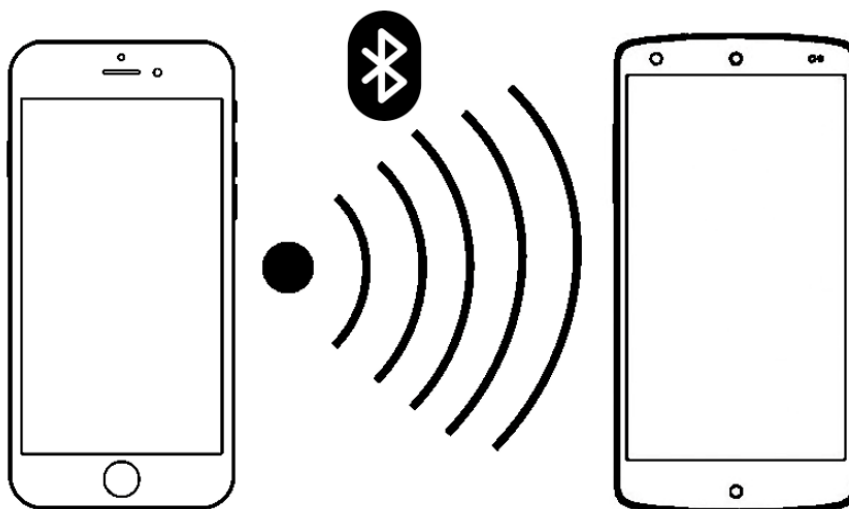


Figure 11 - Visual representation of an iPhone communicating over Bluetooth with an Android phone.

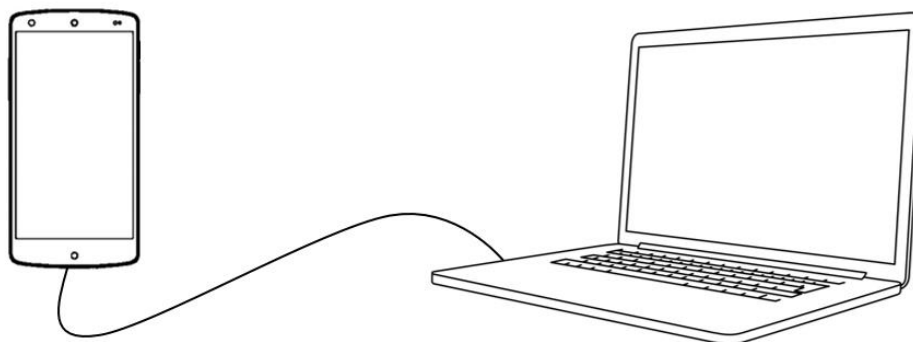


Figure 12 - Visual representation of an Android phone connected to a laptop via USB. Laptop is used for the adb connection.

6 The Value of *dumsys*

As mentioned before, one of the greatest challenges in mobile digital forensics is the fast evolving state of the device, their hardware and their software. For example, Windows 10 was released on the 29th of July 2015. Since the release of Windows 10, there have four new major versions of the Android O.S been released. And the 5th version, Android 11, has been announced.

Building tools for forensic investigations that can overcome all these issues that we covered in 1.4 – “ What is Mobile Digital Forensics?” is challenging. We said that adb could provide a solution to overcome most of challenges. But how can adb provide a solution?

By using the *dumsys* tool, we can retrieve (control) data of many different services of a device. The different modules can be found in Attachment 1: List of all services that can be issued with “*dumsys*”. To reflect the value of this tool, we will highlight a few of these modules.

During a DF investigation in a legal context, one of the goals could be to verify if an account of interest is known on a device. Via the command *adb shell dumsys account*, you can see all known accounts.

```
User UserInfo{0:Nick:13}:
Accounts: 16
Account {name=casiernick@gmail.com, type=com.google}
Account {name=, type=com.}
Account {name=, type=com.}
Account {name=, type=com.}
Account {name=, type=com.}
Account {name=, type=com.}
Account {name=, type=com.microsoft.}
Account {name= student.howest.be (Hogeschool West-Vlaanderen), type=com.microsoft.}
Account {name= @student.howest.be, type=com.microsoft.workaccount}
Account {name=, type=com.}
Account {name=Casiernick, type=com.}
Account {name=WhatsApp, type=com.whatsapp}
Account {name=, type=com.}
Account {name=, type=com.}
Account {name=, type=com.google.}
Account {name=, type=com.}
```

Figure 13 - Example of registered accounts on a device

As you can see in Figure 13 - Example of registered accounts on a device, the accounts are displayed in the pattern *name=<account name>, type=<application>*. The image has been partially blurred for security and privacy reasons. Furthermore, under the visible output, via the *account* module, the accounts history can be seen and the cache. As a security/privacy measure, these are not screenshotted.

Another module within *dumsys* that could be meaningful when researching accounts on an Android device is *content*. This module contains all the synchronisation of services and applications where the accounts visible in the module *account* are used. As a security measure, no screenshot of this output was added.

These are just two modules of *dumsys*. These modules contains valuable data about the user of a device. There are over hundred modules within *dumsys* on the Nexus 6. On the OnePlus there are almost two hundred modules. Each containing valuable information that could help forensic investigations.

In the coming chapters we will dive deeper into the module *Bluetooth_manager*. We specifically focus on the section “Bond Events” of this module.

Bluetooth in Digital mobile Forensics
Communication between iOS device and Android (and vice versa)

7 Communication between iOS device and Android (and vice versa)

In the first series of tests, we used the iPhone and the Nexus 6 as test subjects.

The reason we chose these two devices for the first series is twofold. The iPhone got a factory reset and should be clean. We also wanted to know if there is a noticeable difference in the artefact depending on the device's OS sending the request.

Before each test we rebooted each device.

7.1 Test 1: iPhone sends request – time-out

Test Case	An iOS device sends a request to an Android device.
Devices	Apple iPhone 6 Plus & Motorola Nexus 6 (bc:4c:c4:ee:11:9f & 44:80:eb:f0:ad:8f)
Procedure	The iOS device sent a request to the Nexus 6. We didn't interact with the devices after the request has been sent. We waited until the request has timed-out.
Monitored devices	Nexus 6
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	<pre>Bond Events: Total Number of events: 0</pre>
What is expected?	Visible incoming bonding/pairing request. <i>timestamp – BDaddr of iPhone – change state – bonding</i>
During the process	<pre>Bond Events: Total Number of events: 1 Time BD_ADDR Function State 09:37:20.904 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING</pre>
After the process	<pre>Bond Events: Total Number of events: 2 Time BD_ADDR Function State 09:37:20.904 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING 09:38:25.212 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_NONE</pre>
What is visible after the process?	We can see the number of events that took place, at what time the request is received, and the MAC-address of the sender. We can also see that the function " <i>bond_state_changed</i> " is being called and that the bond state is changed to bonding. When the request is timed-out, we can see that the same function is called again and that the bonding state has been set to none. It took about 1 minute before the request has timed-out.

Bluetooth in Digital mobile Forensics
Communication between iOS device and Android (and vice versa)

Is it as expected?	Yes, we expected to see an incoming bond request and we can clearly see an incoming request.
---------------------------	--

7.2 Test 2: Nexus 6 sends request – time-out

Test Case	An Android device sends a request to an iOS device.																				
Devices	Apple iPhone 6 Plus & Motorola Nexus 6 (bc:4c:c4:ee:11:9f & 44:80:eb:f0:ad:8f)																				
Procedure	The Android device sent a request to the Nexus 6. We didn't interact with the devices after the request has been sent. We waited until the request has timed-out.																				
Monitored devices	Nexus 6																				
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>																				
What is monitored?	Bond events																				
Start scenario	Bond Events: Total Number of events: 0																				
What is expected?	The visibility that a request has been send to the iPhone. We expect at least to see the initialization of the request and when it timed-out. We expected that the request would time-out after about 1 minute as seen in the previous test. <i>timestamp – BDaddr of iPhone – create bond – create bond</i> <i>timestamp – BDaddr of iPhone – change state – send request</i> <i>timestamp – BDaddr of iPhone – change state – timeout</i>																				
During the process	Bond Events: Total Number of events: 3 <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Time</th> <th>BD_ADDR</th> <th>Function</th> <th>State</th> </tr> </thead> <tbody> <tr> <td>10:35:00.537</td> <td>bc:4c:c4:ee:11:9f</td> <td>btif_dm_create_bond</td> <td>BOND_STATE_NONE</td> </tr> <tr> <td>10:35:00.544</td> <td>bc:4c:c4:ee:11:9f</td> <td>bond_state_changed</td> <td>BOND_STATE_BONDING</td> </tr> <tr> <td>10:35:04.625</td> <td>bc:4c:c4:ee:11:9f</td> <td>bond_state_changed</td> <td>BOND_STATE_BONDING</td> </tr> </tbody> </table>	Time	BD_ADDR	Function	State	10:35:00.537	bc:4c:c4:ee:11:9f	btif_dm_create_bond	BOND_STATE_NONE	10:35:00.544	bc:4c:c4:ee:11:9f	bond_state_changed	BOND_STATE_BONDING	10:35:04.625	bc:4c:c4:ee:11:9f	bond_state_changed	BOND_STATE_BONDING				
Time	BD_ADDR	Function	State																		
10:35:00.537	bc:4c:c4:ee:11:9f	btif_dm_create_bond	BOND_STATE_NONE																		
10:35:00.544	bc:4c:c4:ee:11:9f	bond_state_changed	BOND_STATE_BONDING																		
10:35:04.625	bc:4c:c4:ee:11:9f	bond_state_changed	BOND_STATE_BONDING																		
After the process	Bond Events: Total Number of events: 4 <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Time</th> <th>BD_ADDR</th> <th>Function</th> <th>State</th> </tr> </thead> <tbody> <tr> <td>10:35:00.537</td> <td>bc:4c:c4:ee:11:9f</td> <td>btif_dm_create_bond</td> <td>BOND_STATE_NONE</td> </tr> <tr> <td>10:35:00.544</td> <td>bc:4c:c4:ee:11:9f</td> <td>bond_state_changed</td> <td>BOND_STATE_BONDING</td> </tr> <tr> <td>10:35:04.625</td> <td>bc:4c:c4:ee:11:9f</td> <td>bond_state_changed</td> <td>BOND_STATE_BONDING</td> </tr> <tr> <td>10:35:39.649</td> <td>bc:4c:c4:ee:11:9f</td> <td>bond_state_changed</td> <td>BOND_STATE_NONE</td> </tr> </tbody> </table>	Time	BD_ADDR	Function	State	10:35:00.537	bc:4c:c4:ee:11:9f	btif_dm_create_bond	BOND_STATE_NONE	10:35:00.544	bc:4c:c4:ee:11:9f	bond_state_changed	BOND_STATE_BONDING	10:35:04.625	bc:4c:c4:ee:11:9f	bond_state_changed	BOND_STATE_BONDING	10:35:39.649	bc:4c:c4:ee:11:9f	bond_state_changed	BOND_STATE_NONE
Time	BD_ADDR	Function	State																		
10:35:00.537	bc:4c:c4:ee:11:9f	btif_dm_create_bond	BOND_STATE_NONE																		
10:35:00.544	bc:4c:c4:ee:11:9f	bond_state_changed	BOND_STATE_BONDING																		
10:35:04.625	bc:4c:c4:ee:11:9f	bond_state_changed	BOND_STATE_BONDING																		
10:35:39.649	bc:4c:c4:ee:11:9f	bond_state_changed	BOND_STATE_NONE																		
What is visible after the process?	We can see the number of events that took place, at what time the request has been created and what the MAC-address of the receiver is. We can see that when the request is created, the function <i>"btif_dm_create_bond"</i> is called and that this doesn't affect the bond state. After the creation of the request, the function <i>"bond_state_changed"</i> is called and the state is set to bonding. After four seconds, the same function is called again with the same outcome. 35 seconds after the last function call, the same function is																				

Bluetooth in Digital mobile Forensics
Communication between iOS device and Android (and vice versa)

	called again, this time the outcome is the change of the bond state to none. In this test, it took about 39 seconds before the request is timed-out.
Is it as expected?	No, as we expected, we can see the creation of the request and we can also see, based on the MAC-address, that the iPhone is the receiver of the request. We can also see when the request has timed-out. We did not expect to see a difference in time-out time. In this test, the time-out happened almost half a minute earlier than in test 1. We also did not expect to see the function <i>"bond_state_changed"</i> being called within 4 seconds. We have no idea why this happens.

7.3 Test 3: iPhone sends request – cancelled

Test Case	An iOS device sends a request to an Android device. The iOS device cancels the request.
Devices	Apple iPhone 6 Plus & Motorola Nexus 6 (bc:4c:c4:ee:11:9f & 44:80:eb:f0:ad:8f)
Procedure	The iPhone sent a request to the Nexus 6. When the Nexus 6 has received the request, we cancelled the request on the iPhone (similar to Android, see Attachment 4: Pop-up when sending a request on Android).
Monitored devices	Nexus 6
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	<code>Bond Events: Total Number of events: 0</code>
What is expected?	We expected to see that a request has been send to the Nexus 6. We expected to see a different behaviour in the artefact when the request was cancelled compared to test 1 when the request timed-out. <i>timestamp – BDaddr of iPhone – change state – bonding</i> <i>timestamp – BDaddr of iPhone – cancel bond – none</i>
During the process	<code>Bond Events: Total Number of events: 1 Time BD_ADDR Function State 10:49:24.502 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING</code>

Bluetooth in Digital mobile Forensics

Communication between iOS device and Android (and vice versa)

After the process	<pre>Bond Events: Total Number of events: 2 Time BD_ADDR Function State 10:49:24.502 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING 10:49:32.546 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_NONE</pre>
What is visible after the process?	<p>We can see when the request is received and that the sender was the iPhone. We see that the bond state is set to bonding when the request is received. We see that when we cancelled the request (± 10 sec. after receiving) that the bond state is changed to none. Since this timeframe is shorter compared to a time-out, this means that the Nexus 6 actually knows that the request has been cancelled.</p>
Is it as expected	<p>No, we expected to see a different behaviour in the bonding state of the artefact when we cancelled the request. This is not the case. It could be possible that this is the case between two Android devices or devices running the same O.S. version. Further testing will answer this.</p>

7.4 Test 4: Nexus 6 sends request – cancelled

Test Case	<p>An Android device sends a request to an iOS device. The Android device cancels the request.</p>
Devices	<p>Apple iPhone 6 Plus & Motorola Nexus 6 (bc:4c:c4:ee:11:9f & 44:80:eb:f0:ad:8f)</p>
Procedure	<p>The Nexus 6 sent a request to the iPhone. When the iPhone had received the request, we cancelled the request on the Nexus 6. (see Attachment 4: Pop-up when sending a request on Android)</p>
Monitored devices	<p>Nexus 6</p>
How is it monitored?	<p><code>\$ adb shell dumpsys bluetooth_manager</code></p>
What is it monitored?	<p>Bond events</p>
Start scenario	<pre>Bond Events: Total Number of events: 0</pre>
What is expected?	<p>We expected to see the creation of the request, change of the bond state, destruction of the request and finally a change of the bond state to none.</p> <p><i>timestamp – BDaddr of iPhone – create bond – none</i> <i>timestamp – BDaddr of iPhone – change state – bonding</i> <i>timestamp – BDaddr of iPhone – delete bond – bonding</i> <i>timestamp – BDaddr of iPhone – change state – none</i></p>

Bluetooth in Digital mobile Forensics

Communication between iOS device and Android (and vice versa)

<p>During the process</p>	<pre>Bond Events: Total Number of events: 3 Time BD_ADDR Function State 15:21:47.097 bc:4c:c4:ee:11:9f btif_dm_create_bond BOND_STATE_NONE 15:21:47.115 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING 15:21:49.883 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING</pre>
<p>After the process</p>	<pre>Bond Events: Total Number of events: 5 Time BD_ADDR Function State 15:21:47.097 bc:4c:c4:ee:11:9f btif_dm_create_bond BOND_STATE_NONE 15:21:47.115 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING 15:21:49.883 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING 15:22:00.912 bc:4c:c4:ee:11:9f Invalid value BOND_STATE_BONDING 15:22:00.938 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_NONE</pre>
<p>What is visible after the process</p>	<p>We can see the creation of the request, together with the corresponding timestamp and the MAC-address of the iPhone. We see twice the change in bond state to bonding. Between the two calls there is about 2 seconds. When we cancelled the request, we see the function "Invalid value" being called. This triggered the function "bond_state_changed" to set the state to "BOND_STATE_NONE".</p>
<p>Is it as expected</p>	<p>No, we did not expect to see twice the function "bond_state_changed" to be called. We did expect to see the other behaviours of the artefact. We expected to be able to extract this info, such as the MAC-address and timestamp. We expected to see the destruction of the request. The function "Invalid value" seems to be taking care of this. Afterwards, we see as expected, the change to "BOND_STATE_NONE".</p>

7.5 Test 5: iPhone sends request – denied

<p>Test Case</p>	<p>An iOS device sends a request to an Android device. The Android device denies the request.</p>
<p>Devices</p>	<p>Apple iPhone 6 Plus & Motorola Nexus 6. (bc:4c:c4:ee:11:9f & 44:80:eb:f0:ad:8f)</p>
<p>Procedure</p>	<p>The iPhone sent a request to the Nexus 6. Once the request has been received on the Nexus 6, we pressed cancel on the pop-up screen (similar to Attachment 4: Pop-up when sending a request on Android) and thus denying the request.</p>
<p>Monitored devices</p>	<p>Nexus 6</p>
<p>How is it monitored?</p>	<p><i>\$ adb shell dumpsys bluetooth_manager</i></p>
<p>What is monitored?</p>	<p>Bond events</p>
<p>Start scenario</p>	<pre>Bond Events: Total Number of events: 0</pre>

Bluetooth in Digital mobile Forensics

Communication between iOS device and Android (and vice versa)

What is expected?	<p>We expected to see the incoming request as before, followed by a change in the bond state and some other function or state that indicates the denial of the request.</p> <p><i>timestamp – BDaddr of iPhone – change state – bonding</i> <i>timestamp – BDaddr of iPhone – invalid – bonding</i> <i>timestamp – BDaddr of iPhone – change state – none</i></p>
During the process	<pre>Bond Events: Total Number of events: 1 Time BD_ADDR Function State 10:54:19.434 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING</pre>
After the process	<pre>Bond Events: Total Number of events: 3 Time BD_ADDR Function State 10:54:19.434 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING 10:54:28.168 bc:4c:c4:ee:11:9f Invalid value BOND_STATE_BONDING 10:54:28.169 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_NONE</pre>
What is visible after the process?	<p>We can see that when the request is received, the bond state changes as expected. When we denied the request, the function “Invalid value” was called. This function doesn’t seem to effect the bond state. After the “Invalid value” function, the function “bond_state_changed” is called once again to set the bond state to none.</p>
Is it as expected?	<p>Yes, we expected to see this behaviour.</p>

7.6 Test 6: Nexus 6 sends request – denied

Test Case	An Android device sends a request to an iOS device. The iOS device denies the request.
Devices	Apple iPhone 6 Plus & Motorola Nexus 6. (bc:4c:c4:ee:11:9f & 44:80:eb:f0:ad:8f)
Procedure	The Nexus 6 has sent a request to the iPhone. Once the request has been received by the iPhone, we pressed cancel on the pop-up screen (similar to the pop-up screen in Attachment 5: Pop-up when receiving a request on Android) and thus denying the request.
Monitored devices	Nexus 6
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	<pre>Bond Events: Total Number of events: 0</pre>

Bluetooth in Digital mobile Forensics

Communication between iOS device and Android (and vice versa)

<p>What is expected?</p>	<p>We expected to see the creation of the request, change of bond state, and some state of the artefact that indicated the denial of the request.</p> <p><i>timestamp – BDaddr of iPhone – create bond – none</i> <i>timestamp – BDaddr of iPhone – change state – bonding</i> <i>timestamp – BDaddr of iPhone – bond denied – none</i></p>
<p>During the process</p>	<pre>Bond Events: Total Number of events: 3 Time BD_ADDR Function State 09:52:46.515 bc:4c:c4:ee:11:9f btif_dm_create_bond BOND_STATE_NONE 09:52:46.522 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING 09:52:54.358 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING</pre>
<p>After the process</p>	<pre>Bond Events: Total Number of events: 3 Time BD_ADDR Function State 09:52:46.515 bc:4c:c4:ee:11:9f btif_dm_create_bond BOND_STATE_NONE 09:52:46.522 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING 09:52:54.358 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING</pre> <p>After half a minute we got a notification that the pairing failed and there is an extra entry.</p> <pre>Bond Events: Total Number of events: 4 Time BD_ADDR Function State 09:52:46.515 bc:4c:c4:ee:11:9f btif_dm_create_bond BOND_STATE_NONE 09:52:46.522 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING 09:52:54.358 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING 09:53:29.391 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_NONE</pre>
<p>What is visible after the process?</p>	<p>We can see when the request has been created and what the receivers MAC-address is. We see again twice that the bonding is in progress. This time there was an eight second interval.</p> <p>After declining the request on the iPhone, we do not see any changes on the Nexus 6. It took approximately half a minute before the Nexus 6 knew that the request had failed. This behaviour is remarkable.</p>
<p>Is it as expected?</p>	<p>No, we did not expected that the Nexus would have such behaviour. We expected that it would look like a time-out or that the Nexus knew immediately that the iPhone cancelled the request. Due to the notification, it seems a special time-out. The notification hints that there is some function that is called when there is an issue with the request. Further research to this function is needed.</p>

Bluetooth in Digital mobile Forensics
Communication between iOS device and Android (and vice versa)

7.7 Test 7: iPhone sends request – incorrect pin

Test Case	iPhone sends request. iPhone approves the pin, Nexus 6 does not approve the pin.
Devices	Apple iPhone 6 Plus & Motorola Nexus 6 (bc:4c:c4:ee:11:9f & 44:80:eb:f0:ad:8f)
Procedure	We used the iPhone to send a request to the Nexus 6. We confirmed the request on the pop-screen on the iPhone (similar as on Android, see Attachment 4: Pop-up when sending a request on Android). On the Nexus 6 we decline the request by pressing “cancel” (see Attachment 5: Pop-up when receiving a request on Android).
Monitored devices	Nexus 6
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	<pre>Bond Events: Total Number of events: 0</pre>
What is expected?	We expected to see the arrival of the request on the Nexus 6 with the time it has arrived accompanied with the MAC-address of the sender (in this case the iPhone). We expected to see the function “ <i>Invalid value</i> ” to be called when the request was declined, followed by a change in bond state to none. <i>timestamp – BDaddr of iPhone – change state – bonding</i> <i>timestamp – BDaddr of iPhone – invalid value – bonding</i> <i>timestamp – BDaddr of iPhone – change state – none</i>
During the process	<pre>Bond Events: Total Number of events: 1 Time BD_ADDR Function State 16:12:37.736 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING</pre>
After the process	<pre>Bond Events: Total Number of events: 3 Time BD_ADDR Function State 16:12:37.736 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING 16:12:45.068 bc:4c:c4:ee:11:9f Invalid value BOND_STATE_BONDING 16:12:45.069 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_NONE</pre>
What is visible after the process?	We can see the arrival of the request, accompanied by the MAC-address of the sender and the timestamp of arrival. We see that the arrival caused the bond state to be bonding. We see that when we pressed cancel, the function “ <i>Invalid value</i> ” being called followed by a change in bond state to none.
Is it as expected?	Yes, we expected to see this behaviour.

Bluetooth in Digital mobile Forensics
Communication between iOS device and Android (and vice versa)

7.8 Test 8: Nexus 6 sends request – incorrect pin

Test Case	The Nexus 6 sends a request and approves the pin. The iPhone receiving the request declines it.
Devices	Apple iPhone 6 Plus & Motorola Nexus 6 (bc:4c:c4:ee:11:9f & 44:80:eb:f0:ad:8f)
Procedure	We used the Nexus 6 to send a request to the iPhone. We confirmed the request on the pop-screen (similar as seen in Attachment 4: Pop-up when sending a request on Android). On the iPhone we decline the request by pressing “cancel” (similar as seen in Attachment 5: Pop-up when receiving a request on Android).
Monitored devices	Nexus 6
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	<pre>Bond Events: Total Number of events: 0</pre>
What is expected?	We expected to see the creation of the request with its timestamp and the MAC-address of the receiver. We expected to see a change in bond state to bonding followed by the function “Invalid value” being called causing a change in bond state to none. <i>timestamp – BDaddr of iPhone – create bond – none</i> <i>timestamp – BDaddr of iPhone – change state – bonding</i> <i>timestamp – BDaddr of iPhone – invalid value – bonding</i> <i>timestamp – BDaddr of iPhone – change state – none</i>
During the process	<pre>Bond Events: Total Number of events: 3 Time BD_ADDR Function State 16:16:10.367 bc:4c:c4:ee:11:9f btif_dm_create_bond BOND_STATE_NONE 16:16:10.373 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING 16:16:14.248 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING</pre>
After the process	<pre>Bond Events: Total Number of events: 4 Time BD_ADDR Function State 16:16:10.367 bc:4c:c4:ee:11:9f btif_dm_create_bond BOND_STATE_NONE 16:16:10.373 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING 16:16:14.248 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING 16:16:22.375 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_NONE</pre>
What is visible after the process?	We can see when we created the request and who the receiver was based on the MAC-address. We see again twice the change in bond state. When the iPhone declined the request, we just see that the bond state has been set to none. However, we received a

Bluetooth in Digital mobile Forensics
Communication between iOS device and Android (and vice versa)

	notification on the screen of the Nexus telling us that the pin was incorrect (see Attachment 6: Pop-up on Nexus 6 when pin is incorrect).
Is it as expected?	No, we did not see any behaviour in the artefact indicating specifically that the request was declined or that the pin was incorrect.

7.9 Test 9: iPhone sends request – accepted

Test Case	An iOS device sends a request to an Android device. The request is accepted and the two devices are bond/paired.
Devices	Apple iPhone 6 Plus & Motorola Nexus 6 (bc:4c:c4:ee:11:9f & 44:80:eb:f0:ad:8f)
Procedure	The iPhone sent a request to the Nexus 6. We accepted the request on both devices. By doing so, both devices are now bonded/paired via Bluetooth.
Monitored devices	Nexus 6
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	<pre>Bond Events: Total Number of events: 0</pre>
What is expected?	We expected to see an incoming request, a change in bond state and some form of confirmation that the request has been accepted. <i>timestamp – BDaddr of iPhone – change state – bonding</i> <i>timestamp – BDaddr of iPhone – chnage state – bonded</i>
During the process	<pre>Bond Events: Total Number of events: 1 Time BD_ADDR Function State 10:21:18.064 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING</pre>
After the process	<pre>Bond Events: Total Number of events: 2 Time BD_ADDR Function State 10:21:18.064 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING 10:21:30.774 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDED</pre>
What is visible after the process?	We can see when the incoming request has been received, we can see the MAC-address of the sender of the request and that the incoming request caused a change in the bond state. We can see that when the bond request has been accepted that the bond state is changed to “ <i>BOND_STATE_BONDED</i> ”.
Is it as expected?	Yes, we expected to see the incoming bond request and some form of confirmation that the two devices are bonded. We hypothesized if

Bluetooth in Digital mobile Forensics
Communication between iOS device and Android (and vice versa)

	the confirmation and the bond state would be the same entry or not, but as you can see, it is the same entry.
--	---

7.10 Test 10: Forget Device

Test Case	Both devices are connect. We forget the connection between both.
Devices	Apple iPhone 6 Plus & Motorola Nexus 6 (bc:4c:c4:ee:11:9f & 44:80:eb:f0:ad:8f)
Procedure	Both devices are at the beginning of the test connect with each other. In the Bluetooth settings of the Nexus 6, we look for the iPhone and pressed “forget” (see Attachment 7: Forgetting a paired device on Android).
Monitored devices	Nexus 6
How is it monitored?	<i>\$ adb shell dumpsys bluetooth_manager</i>
What is monitored?	Bond events
Start scenario	<pre>Bond Events: Total Number of events: 2 Time BD_ADDR Function State 10:21:18.064 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING 10:21:30.774 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDED</pre>
What is expected?	<p>We expected to see a change in the state or a function that is called that indicates that a bonded/paired device is forgotten/deleted.</p> <p><i>timestamp – BDaddr of iPhone – remove bond – none</i> <i>timestamp – BDaddr of iPhone – change state – none</i></p>
During the process	N/A
After the process	<pre>Bond Events: Total Number of events: 4 Time BD_ADDR Function State 10:21:18.064 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING 10:21:30.774 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDED 10:21:54.582 bc:4c:c4:ee:11:9f btif_dm_remove_bond BOND_STATE_NONE 10:21:54.611 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_NONE</pre>
What is visible after the process?	We can see that the function “ <i>btif_dm_remove_bond</i> ” has been called. The name of the function indicates that this functions removes the known connection between the two devices and sets the state to none.
Is it as expected?	Yes, we expected to see a change in the artefact indicating that we forgot the Bluetooth connection between the two devices;

Bluetooth in Digital mobile Forensics
Communication between iOS device and Android (and vice versa)

7.11 Test 11: Nexus 6 sends request – accepted

Test Case	An Android device sends a request to an iOS device. The request is accepted and both devices are bonded/paired.																				
Devices	Apple iPhone 6 Plus & Motorola Nexus 6 (bc:4c:c4:ee:11:9f & 44:80:eb:f0:ad:8f)																				
Procedure	We used the Nexus 6 to send a request to the iPhone. We accepted the request on both devices. By doing so, both devices are now bonded/paired.																				
Monitored devices	Nexus 6																				
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>																				
What is monitored?	Bond events																				
Start scenario	Bond Events: Total Number of events: 0																				
What is expected?	We expected to see the creation of the request, change in bonding state (probably twice to bonding), and a bonding state change to bonded. <i>timestamp – BDaddr of iPhone – create bond – none</i> <i>timestamp – BDaddr of iPhone – change state – bonding</i> <i>timestamp – BDaddr of iPhone – change state – bonding</i> <i>timestamp – BDaddr of iPhone – change state – bonded</i>																				
During the process	Bond Events: Total Number of events: 3 <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Time</th> <th>BD_ADDR</th> <th>Function</th> <th>State</th> </tr> </thead> <tbody> <tr> <td>10:31:35.062</td> <td>bc:4c:c4:ee:11:9f</td> <td>btif_dm_create_bond</td> <td>BOND_STATE_NONE</td> </tr> <tr> <td>10:31:35.067</td> <td>bc:4c:c4:ee:11:9f</td> <td>bond_state_changed</td> <td>BOND_STATE_BONDING</td> </tr> <tr> <td>10:31:37.556</td> <td>bc:4c:c4:ee:11:9f</td> <td>bond_state_changed</td> <td>BOND_STATE_BONDING</td> </tr> </tbody> </table>	Time	BD_ADDR	Function	State	10:31:35.062	bc:4c:c4:ee:11:9f	btif_dm_create_bond	BOND_STATE_NONE	10:31:35.067	bc:4c:c4:ee:11:9f	bond_state_changed	BOND_STATE_BONDING	10:31:37.556	bc:4c:c4:ee:11:9f	bond_state_changed	BOND_STATE_BONDING				
Time	BD_ADDR	Function	State																		
10:31:35.062	bc:4c:c4:ee:11:9f	btif_dm_create_bond	BOND_STATE_NONE																		
10:31:35.067	bc:4c:c4:ee:11:9f	bond_state_changed	BOND_STATE_BONDING																		
10:31:37.556	bc:4c:c4:ee:11:9f	bond_state_changed	BOND_STATE_BONDING																		
After the process	Bond Events: Total Number of events: 4 <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Time</th> <th>BD_ADDR</th> <th>Function</th> <th>State</th> </tr> </thead> <tbody> <tr> <td>10:31:35.062</td> <td>bc:4c:c4:ee:11:9f</td> <td>btif_dm_create_bond</td> <td>BOND_STATE_NONE</td> </tr> <tr> <td>10:31:35.067</td> <td>bc:4c:c4:ee:11:9f</td> <td>bond_state_changed</td> <td>BOND_STATE_BONDING</td> </tr> <tr> <td>10:31:37.556</td> <td>bc:4c:c4:ee:11:9f</td> <td>bond_state_changed</td> <td>BOND_STATE_BONDING</td> </tr> <tr> <td>10:31:46.017</td> <td>bc:4c:c4:ee:11:9f</td> <td>bond_state_changed</td> <td>BOND_STATE_BONDED</td> </tr> </tbody> </table>	Time	BD_ADDR	Function	State	10:31:35.062	bc:4c:c4:ee:11:9f	btif_dm_create_bond	BOND_STATE_NONE	10:31:35.067	bc:4c:c4:ee:11:9f	bond_state_changed	BOND_STATE_BONDING	10:31:37.556	bc:4c:c4:ee:11:9f	bond_state_changed	BOND_STATE_BONDING	10:31:46.017	bc:4c:c4:ee:11:9f	bond_state_changed	BOND_STATE_BONDED
Time	BD_ADDR	Function	State																		
10:31:35.062	bc:4c:c4:ee:11:9f	btif_dm_create_bond	BOND_STATE_NONE																		
10:31:35.067	bc:4c:c4:ee:11:9f	bond_state_changed	BOND_STATE_BONDING																		
10:31:37.556	bc:4c:c4:ee:11:9f	bond_state_changed	BOND_STATE_BONDING																		
10:31:46.017	bc:4c:c4:ee:11:9f	bond_state_changed	BOND_STATE_BONDED																		
What is visible after the process?	We can see when the request was created, and who the receiver is based on the MAC-address. We see a change in the bonding state to bonding and when the request has been accepted to bonded.																				
Is it as expected?	Yes, we expected to see this behaviour. We still do not know why there is twice a change in bonding state to bonding. This time the time difference is about 2 seconds.																				

Bluetooth in Digital mobile Forensics
 Communication between iOS device and Android (and vice versa)

7.12 Test 12: Reconnect devices

Test Case	Both devices are bonded/paired with each other. The devices are initially not in range of each other. When they are in range, they are reconnected.
Devices	Apple iPhone 6 Plus & Motorola Nexus 6 (bc:4c:c4:ee:11:9f & 44:80:eb:f0:ad:8f)
Procedure	Both devices were already bonded/paired but not in range of each other to be connected. We brought them back in each other range and connected them.
Monitored devices	Nexus 6
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	<code>Bond Events: Total Number of events: 0</code>
What is expected?	We expected to see a change in bond state to be bonded. <i>timestamp – BDaddr of iPhone – change state – bonded</i>
During the process	N/A
After the process	<code>Bond Events: Total Number of events: 0</code>
What is visible after the process?	Nothing, there is nothing visible in the artefact when two devices that already have been paired are reconnecting. Reconnecting two phones is not a convenient way to test this scenario, so we'll come back to reconnecting devices in other scenarios in the coming chapters.
Is it as expected?	No, we expected a change in the bond state, but we see no change in the artefact.

Bluetooth in Digital mobile Forensics
Communication between iOS device and Android (and vice versa)

7.13 Test 13: Forget Device and attempt to reconnect

Test Case	The iPhone forgot the connection and the Nexus 6 wants to reconnect
Devices	Apple iPhone 6 Plus & Motorola Nexus 6 (bc:4c:c4:ee:11:9f & 44:80:eb:f0:ad:8f)
Procedure	Via the Bluetooth settings we forget the connection on the iPhone. (similar to Android, see Attachment 7: Forgetting a paired device on Android). With the Nexus 6 we tried to reconnect with the iPhone via Bluetooth. We denied the request on the iPhone when we received a new request.
Monitored devices	Nexus 6
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	<pre>Bond Events: Total Number of events: 0</pre>
What is expected?	We expected to see a change in bond state to bonding followed by a similar behaviour as observed when the request sent by the Nexus 6 was denied (see Test 6: Nexus 6 sends request – denied). <i>timestamp – BDaddr of iPhone – change state – bonding</i> <i>timestamp – BDaddr of iPhone – create bond – none</i> <i>timestamp – BDaddr of iPhone – change state – bonding</i> <i>timestamp – BDaddr of iPhone – change state – bonding</i> <i>timestamp – BDaddr of iPhone – change state – none</i>
During the process of reconnecting	<pre>Bond Events: Total Number of events: 0</pre>
After the process of reconnecting and sending the new request	<pre>Bond Events: Total Number of events: 1 Time BD_ADDR Function State 16:26:21.560 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING</pre>
After deny	<pre>Bond Events: Total Number of events: 2 Time BD_ADDR Function State 16:26:21.560 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_BONDING 16:26:56.604 bc:4c:c4:ee:11:9f bond_state_changed BOND_STATE_NONE</pre>
What is visible after the process?	We see when we send the new request to the iPhone that the bond state is changed to bonding. When the request was denied, we see that the state has been changed to none. However, the iPhone is no longer listed between the paired devices.

Bluetooth in Digital mobile Forensics

Communication between iOS device and Android (and vice versa)

Is it as expected?	No, we expected to see a traces of the attempt to reconnect, as well as the creation of a new request when the iPhone received a new pairing request.
---------------------------	---

8 Communication between two Android devices

In this series of test, all the test of chapter 7 - Communication between iOS device and Android (and vice versa) have been redone. The test are now between two Android devices. Here, we used the OnePlus 7T Pro and the Nexus 6 as test subject.

We wanted to verify if there was any difference in the artefact when the request got cancelled or denied. We specifically hoped to see a different behaviour with the sender of the request when the receiver of the request denies the request. We suspected that there will be no difference when the request is timed-out or accepted.

As previously mentioned, these two devices are my personal devices. The OnePlus is the successor to my previous daily driver, the Nexus 6.

In the screenshots of the output, the output we got with the Nexus 6 is in the black font, the output with the OnePlus is in the blue font. The first screenshot will always be from the Nexus 6 and the second one from the OnePlus.

8.1 Test 1: OnePlus sends request – time-out

Test Case	An Android device (device A) sends a request to another Android device (device B). The request is timed-out.	
Devices	Motorola Nexus 6 & OnePlus 7T Pro (44:80:eb:f0:ad:8f & 48:01:c5:86:27:ce)	
Procedure	We used the OnePlus to send a request to the Nexus 6. We didn't interact with the devices once the request was sent and we let the request time-out.	
Monitored devices	Nexus 6 & OnePlus	
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>	
What is monitored?	Bond events	
Start scenario	Start scenario on Nexus 6:	<code>Bond Events: Total Number of events: 0</code>
	Start scenario on OnePlus:	<code>Bond Events: Total Number of events: 0</code>
What is expected?	On the Nexus 6 we expected to see the a change of bond state to bonding when the request has been received. When the request has timed-out, we expected to see a change in bond state to none. <i>timestamp – BDaddr of OnePlus – change state – bonding</i> <i>timestamp – BDaddr of OnePlus – change state – none</i>	

Bluetooth in Digital mobile Forensics

Communication between two Android devices

	<p>We expected to see the creation of the request on the OnePlus with the time it was created and the MAC-address of the Nexus 6. We expected to see a change in bond state to bonding and a change to bond state none.</p> <p><i>timestamp – BDaddr of Nexus 6 – create bond – none</i> <i>timestamp – BDaddr of Nexus 6 – change state – bonding</i> <i>timestamp – BDaddr of Nexus 6 – change state – bonding</i> <i>timestamp – BDaddr of Nexus 6 – change state – none</i></p>
<p>During the process</p>	<pre> Bond Events: Total Number of events: 1 Time BD_ADDR Function State 10:15:38.016 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING </pre> <pre> Bond Events: Total Number of events: 3 Time address Function State 10:15:27.466 44:80:eb:f0:ad:8f btif_dm_create_bond BOND_STATE_NONE 10:15:27.466 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 10:15:37.131 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING </pre>
<p>After the process</p>	<pre> Bond Events: Total Number of events: 2 Time BD_ADDR Function State 10:15:38.016 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 10:16:08.319 48:01:c5:86:27:ce bond_state_changed BOND_STATE_NONE </pre> <pre> Bond Events: Total Number of events: 4 Time address Function State 10:15:27.466 44:80:eb:f0:ad:8f btif_dm_create_bond BOND_STATE_NONE 10:15:27.466 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 10:15:37.131 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 10:16:07.376 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_NONE </pre>
<p>What is visible after the process?</p>	<p>We can see on the output from the Nexus 6 when the request has been received and what the MAC-address of the sender was. We can also see that the request was timed-out after about half a minute or 30 seconds.</p> <p>On the OnePlus we can see when the request was created and the MAC-address of the receiver. We see twice a change in bond state to bonding. This time it takes about ten seconds. If we compare these timestamps to the timestamps of the Nexus 6, we can derive that the second time we see the bonding state being set to bonding is around the same time as the Nexus 6 is in bonding state.</p> <p>On the OnePlus we can also see that the request is timed-out after the same time period as we saw on the Nexus 6, about half a minute or 30 seconds.</p>
<p>Is it as expected?</p>	<p>Yes, we expected to see these results.</p> <p>We can see when the artefact is created, we see the change in bond state on the OnePlus, we can see the arrival of the request based on the bond state on the Nexus 6. We can see on both devices when the request timed-out after 30 seconds.</p>

Bluetooth in Digital mobile Forensics
Communication between two Android devices

8.2 Test 2: Nexus 6 sends request – time-out

Test Case	An Android device (device B) sends a request to another Android device (device A). The request is timed-out
Devices	Motorola Nexus 6 & OnePlus 7T Pro (44:80:eb:f0:ad:8f & 48:01:c5:86:27:ce)
Procedure	We used the Nexus 6 to send a request to the OnePlus. We didn't interact with the devices once the request was sent and we let the request time-out.
Monitored devices	Nexus 6 & OnePlus
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	<pre>Bond Events: Total Number of events: 0</pre> <pre>Bond Events: Total Number of events: 0</pre>
What is expected?	<p>We expected to see the creation of the request on the Nexus 6 with the time of creation and the MAC-address of the OnePlus. We expected to see a change in bond state to bonding followed with a second change in bond state to bonding when the OnePlus has received the request. We expect to see a last change in bond state to none when the request is timed-out.</p> <p><i>timestamp – BDaddr of OnePlus – create bond – none</i> <i>timestamp – BDaddr of OnePlus – change state – bonding</i> <i>timestamp – BDaddr of OnePlus – change state – bonding</i> <i>timestamp – BDaddr of OnePlus – change state – none</i></p> <p>On the OnePlus we expected to see a change in bond state to bonding indicating that the OnePlus has received the request. When the request is timed-out, a change of bond state to none.</p> <p><i>timestamp – BDaddr of Nexus 6 – change state – bonding</i> <i>timestamp – BDaddr of Nexus 6 – change state – none</i></p>
During the process	<pre>Bond Events: Total Number of events: 3 Time BD_ADDR Function State 10:57:51.632 48:01:c5:86:27:ce btif_dm_create_bond BOND_STATE_NONE 10:57:51.637 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 10:57:55.217 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING</pre> <pre>Bond Events: Total Number of events: 1 Time address Function State 10:57:55.652 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING</pre>

Bluetooth in Digital mobile Forensics
Communication between two Android devices

<p>After the process</p>	<pre> Bond Events: Total Number of events: 4 Time BD_ADDR Function State 10:57:51.632 48:01:c5:86:27:ce btif_dm_create_bond BOND_STATE_NONE 10:57:51.637 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 10:57:55.217 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 10:58:25.561 48:01:c5:86:27:ce bond_state_changed BOND_STATE_NONE </pre> <pre> Bond Events: Total Number of events: 2 Time address Function State 10:57:55.652 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 10:58:25.921 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_NONE </pre>
<p>What is visible after the process?</p>	<p>On the Nexus 6 we can see the creation of the request and at what time it was created. We can also derive the receiver of the request based on the MAC-address. After the creation of the request, we can see twice the change in bond state to bonding followed by the change in bond state to none. The time between the two bonding states is now four seconds. After about half a minute (30 seconds) the request is timed-out. As in the previous test of this series, we can now derive why we see twice the change in bond state on the sending device.</p> <p>The output of the OnePlus shows when the request is received and from who based on the MAC-address. We can see that this causes a change in bond state, setting the state to bonding. After about half a minute we can see here too that the request is timed-out and the state has been set to none.</p>
<p>Is it as expected?</p>	<p>Yes, we expected to see this behaviour. This verifies also the suspicion of what the meaning is of the returning change in bond state “<i>BOND_STATE_BONDING</i>” on the sending device.</p>

8.3 Test 3: OnePlus sends request – cancelled

<p>Test Case</p>	<p>An Android device (device A) sends a request to another Android device (device B). The request is being cancelled on device A.</p>
<p>Devices</p>	<p>Motorola Nexus 6 & OnePlus 7T Pro (44:80:eb:f0:ad:8f & 48:01:c5:86:27:ce)</p>
<p>Procedure</p>	<p>We used the OnePlus to send a request to the Nexus 6. Once the Nexus 6 has received the request, we cancelled the request on the OnePlus (see Attachment 4: Pop-up when sending a request on Android).</p>
<p>Monitored devices</p>	<p>Nexus 6 & OnePlus</p>
<p>How is it monitored?</p>	<p><i>\$ adb shell dumpsys bluetooth_manager</i></p>
<p>What is monitored?</p>	<p>Bond events</p>

Bluetooth in Digital mobile Forensics
Communication between two Android devices

<p>Start scenario</p>	<pre>Bond Events: Total Number of events: 0 Bond Events: Total Number of events: 0</pre>
<p>What is expected?</p>	<p>On the Nexus 6 we expected to see when the request is received, who the sender was and the change in bond state. We hoped to see that when the OnePlus has cancelled the request, we see a new behaviour indicating that the request has been cancelled.</p> <p><i>timestamp – BDaddr of OnePlus – change state – bonding</i> <i>timestamp – BDaddr of OnePlus – change state – cancelled</i> <i>timestamp – BDaddr of OnePlus – change state – none</i></p> <p>On the OnePlus we expected to see the creation of the request, what time it is created and the receiver’s MAC-address. We expected to see the bond change twice. We expect something similar to the “Invalid value” we saw in the previous series of test (Chapter 7 – Communication between iPhone and Android).</p> <p><i>timestamp – BDaddr of Nexus 6 – create bond – none</i> <i>timestamp – BDaddr of Nexus 6 – change state – bonding</i> <i>timestamp – BDaddr of Nexus 6 – change state – bonding</i> <i>timestamp – BDaddr of Nexus 6 – invalid value – bonding</i> <i>timestamp – BDaddr of Nexus 6 – change state – none</i></p>
<p>During the process</p>	<pre>Bond Events: Total Number of events: 1 Time BD_ADDR Function State 13:08:19.877 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING Bond Events: Total Number of events: 3 Time address Function State 13:08:17.801 44:80:eb:f0:ad:8f btif_dm_create_bond BOND_STATE_NONE 13:08:17.802 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 13:08:20.381 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING</pre>
<p>After the process</p>	<pre>Bond Events: Total Number of events: 2 Time BD_ADDR Function State 13:08:19.877 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 13:08:26.859 48:01:c5:86:27:ce bond_state_changed BOND_STATE_NONE Bond Events: Total Number of events: 5 Time address Function State 13:08:17.801 44:80:eb:f0:ad:8f btif_dm_create_bond BOND_STATE_NONE 13:08:17.802 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 13:08:20.381 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 13:08:27.110 44:80:eb:f0:ad:8f Invalid value BOND_STATE_BONDING 13:08:27.147 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_NONE</pre>
<p>What is visible after the process?</p>	<p>On the Nexus 6 we see when and from what device, based on the MAC-address, we received the request. We can see that this causes the state to change to bonding. We can also see that the state has been set to none after five seconds. When we compare these</p>

Bluetooth in Digital mobile Forensics
Communication between two Android devices

	<p>timestamps to the once of the OnePlus, we see that this is when we pressed cancel.</p> <p>On the OnePlus we see the creation of the artefact, when it has been created and the MAC-address of the receiver. We can see the two bond state changes and when we pressed cancel, we can see that the function “<i>Invalid value</i>” has been called. We see that afterwards the state is set to none again.</p>
Is it as expected?	<p>No, we expected to see a difference in the artefact on the Nexus 6 when the request got cancelled. We see the same behaviour as we saw with the iPhone, indicating that even if the Nexus 6 knows that the request is cancelled, the state is just set to none and no further attention is given to the event.</p>

8.4 Test 4: Nexus 6 sends request – cancelled

Test Case	An Android device (device B) sends a request to another Android device (device A). The request is being cancelled on device B.
Devices	Motorola Nexus 6 & OnePlus 7T Pro (44:80:eb:f0:ad:8f & 48:01:c5:86:27:ce)
Procedure	We used the Nexus 6 to send a request to the OnePlus. Once the OnePlus has received the request, we cancel the request on the Nexus 6 (see Attachment 4: Pop-up when sending a request on Android).
Monitored devices	Nexus 6 & OnePlus
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	<pre>Bond Events: Total Number of events: 0 Bond Events: Total Number of events: 0</pre>
What is expected?	<p>We expected to see when the request has been created and who the receiver is based on the MAC-address. We expected to see twice the change in bond state to bonding. We expect to see the destruction/deletion of the request on the Nexus 6.</p> <p><i>timestamp – BDaddr of OnePlus – create bond – none</i> <i>timestamp – BDaddr of OnePlus – change state – bonding</i> <i>timestamp – BDaddr of OnePlus – change state – bonding</i></p>

Bluetooth in Digital mobile Forensics

Communication between two Android devices

	<p><i>timestamp – BDaddr of OnePlus – Invalid value – bonding</i> <i>timestamp – BDaddr of OnePlus – change state – none</i></p> <p>On the OnePlus we expected to see the arrival of the request, with the according timestamp and the MAC-address of the sender. We hoped to see a new behaviour indicating the annulation of the request instead of a change of bonding state.</p> <p><i>timestamp – BDaddr of Nexus 6 – change state – bonding</i> <i>timestamp – BDaddr of Nexus 6 – change state – cancelled</i> <i>timestamp – BDaddr of Nexus 6 – change state – none</i></p>
<p>During the process</p>	<pre>Bond Events: Total Number of events: 3 Time BD_ADDR Function State 13:24:33.582 48:01:c5:86:27:ce btif_dm_create_bond BOND_STATE_NONE 13:24:33.585 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 13:24:37.217 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING</pre> <pre>Bond Events: Total Number of events: 1 Time address Function State 13:24:37.977 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING</pre>
<p>After the process</p>	<pre>Bond Events: Total Number of events: 5 Time BD_ADDR Function State 13:24:33.582 48:01:c5:86:27:ce btif_dm_create_bond BOND_STATE_NONE 13:24:33.585 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 13:24:37.217 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 13:24:54.126 48:01:c5:86:27:ce Invalid value BOND_STATE_BONDING 13:24:54.152 48:01:c5:86:27:ce bond_state_changed BOND_STATE_NONE</pre> <pre>Bond Events: Total Number of events: 2 Time address Function State 13:24:37.977 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 13:24:55.014 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_NONE</pre>
<p>What is visible after the process?</p>	<p>On the Nexus 6 we can see the creation of the artefact, with the MAC-address of the receiver of the request and when the request was issued. We can see that this is followed by a change in bond state, indicating that the receiver is ready to pair. After about four seconds we see once again a change in bond state, indicating that the receiver is ready to bond. When we pressed cancel we can see that the function “<i>Invalid value</i>” is called, this is followed by a change in bond state to none.</p> <p>On the OnePlus we can see when we received the request, accompanied with the MAC-address of the sender. When we cancelled the request, we see on the OnePlus a change in bond state to “<i>BOND_STATE_NONE</i>”.</p>
<p>Is it as expected?</p>	<p>No, we expected to see a new behaviour on the OnePlus when we cancelled the request. The other observed behaviours were as expected. We expected to see at the sender’s end the MAC-address of the receiver accompanied with the creation of the request, the bonding states, and the cancellation. All with their timestamps.</p>

Bluetooth in Digital mobile Forensics
Communication between two Android devices

	We saw at the receiver's end the receipt of the request with the sender's MAC-address and the timestamp, causing the bond state to change to bonding.
--	---

8.5 Test 5: OnePlus sends request – denied

Test Case	An Android device (device A) sends a request to another Android device (device B). The request is being declined on device B.	
Devices	Motorola Nexus 6 & OnePlus 7T Pro (44:80:eb:f0:ad:8f & 48:01:c5:86:27:ce)	
Procedure	We used the OnePlus as a sender for the request. We send the request to the Nexus 6. Once received on the Nexus 6, we declined the request by pressing "cancel" (see Attachment 5: Pop-up when receiving a request on Android).	
Monitored devices	Nexus 6 & OnePlus	
How is it monitored?	\$ adb shell dumpsys bluetooth_manager	
What is monitored?	Bond events	
Start scenario	<pre style="background-color: #ffffcc; padding: 5px;">Bond Events: Total Number of events: 0</pre> <pre style="background-color: #ffffcc; padding: 5px;">Bond Events: Total Number of events: 0</pre>	
What is expected?	<p>We expected to see the receipt of the request on the Nexus 6, accompanied with the sender's MAC-address (the OnePlus' MAC-address) and the timestamp of when the request has been received. We expected to see the "Invalid value" function being called when we denied the request on the Nexus 6. Lastly we expected to see the bond state being set to none.</p> <p><i>timestamp – BDaddr of OnePlus – change state – bonding</i> <i>timestamp – BDaddr of OnePlus – Invalid value – bonding</i> <i>timestamp – BDaddr of OnePlus – change state – none</i></p> <p>On the OnePlus we expected to see the creation of the request, accompanied with the receiver's MAC-address and the timestamp of when the request was created. After the creation we expected to see the change in bond state to bonding twice, followed by a new behaviour indicating that the request has been declined. Finally we expected to see the destruction of the request on the OnePlus.</p> <p><i>timestamp – BDaddr of Nexus 6 – create bond – none</i> <i>timestamp – BDaddr of Nexus 6 – change state – bonding</i></p>	

Bluetooth in Digital mobile Forensics

Communication between two Android devices

	<p><i>timestamp – BDaddr of Nexus 6 – change state – bonding</i></p> <p><i>timestamp – BDaddr of Nexus 6 – change state – denied</i></p> <p><i>timestamp – BDaddr of Nexus 6 – change state – none</i></p>
During the process	<pre> Bond Events: Total Number of events: 1 Time BD_ADDR Function State 14:37:26.194 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING </pre> <pre> Bond Events: Total Number of events: 3 Time address Function State 14:37:22.388 44:80:eb:f0:ad:8f btif_dm_create_bond BOND_STATE_NONE 14:37:22.388 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 14:37:26.720 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING </pre>
After the process	<pre> Bond Events: Total Number of events: 3 Time BD_ADDR Function State 14:37:26.194 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 14:37:46.948 48:01:c5:86:27:ce Invalid value BOND_STATE_BONDING 14:37:46.948 48:01:c5:86:27:ce bond_state_changed BOND_STATE_NONE </pre> <pre> Bond Events: Total Number of events: 4 Time address Function State 14:37:22.388 44:80:eb:f0:ad:8f btif_dm_create_bond BOND_STATE_NONE 14:37:22.388 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 14:37:26.720 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 14:37:56.982 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_NONE </pre>
What is visible after the process?	<p>On the Nexus 6 we can see when the request has been received and who sent it based on the MAC-address. We can see when we declined the request and that in order to do so, the function “<i>Invalid value</i>” is called. This causes a change in bond state to none.</p> <p>On the OnePlus we can see when we created the request and that, based on the MAC-address, the Nexus 6 is the receiver of the request. We can see when the OnePlus was ready to bond in the first entry where the bond state is changed to bonding and when the Nexus 6 was ready based on the second entry of “<i>BOND_STATE_BONDING</i>”. This was after approximately four seconds. When the Nexus 6 has declined the request, around the 14:37:46 mark. It took 30 seconds before the OnePlus’ bond state changed. This indicates that the OnePlus did not know that the request has been denied.</p>
Is it as expected?	<p>No, we expected that the OnePlus would know when the Nexus 6 declined, thus causing an entry in the bond events section. We did not see this behaviour, similar to a cancelation. We expected to see the other entries to the section.</p>

Bluetooth in Digital mobile Forensics
Communication between two Android devices

8.6 Test 6: Nexus 6 sends request – denied

Test Case	An Android device (device B) sends a request to another Android device (device A). The request is being declined on device B.
Devices	Motorola Nexus 6 & OnePlus 7T Pro (44:80:eb:f0:ad:8f & 48:01:c5:86:27:ce)
Procedure	In this test we used the Nexus 6 to send a request to the OnePlus. Once the OnePlus received the request, we declined the request by pressing “cancel” (see Attachment 5: Pop-up when receiving a request on Android).
Monitored devices	Nexus 6 & OnePlus
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	<pre>Bond Events: Total Number of events: 0</pre> <pre>Bond Events: Total Number of events: 0</pre>
What is expected?	<p>We expected to see the creation of the request on the Nexus 6 with the timestamp of creation and the receiver’s MAC-address Followed by twice the change in bond state to bonding and a new behaviour indication that the request has been denied causing the bond state to be set to none.</p> <p><i>timestamp – BDaddr of OnePlus – create bond – none</i> <i>timestamp – BDaddr of OnePlus – change state – bonding</i> <i>timestamp – BDaddr of OnePlus – change state – bonding</i> <i>timestamp – BDaddr of OnePlus – bond denied – bonding</i> <i>timestamp – BDaddr of OnePlus – change state – none</i></p> <p>On the OnePlus we expect to see receiving the request, accompanied with the MAC-address of the sender and the timestamp of arrival. We expected to see the function “Invalid value” to be called when we cancel the request, followed by a change in bond state to none.</p> <p><i>timestamp – BDaddr of Nexus 6 – change state – bonding</i> <i>timestamp – BDaddr of Nexus 6 – Invalid value – bonding</i> <i>timestamp – BDaddr of Nexus 6 – change state – none</i></p>

Bluetooth in Digital mobile Forensics

Communication between two Android devices

<p>During the process</p>	<pre> Bond Events: Total Number of events: 3 Time BD_ADDR Function State 14:44:52.002 48:01:c5:86:27:ce btif_dm_create_bond BOND_STATE_NONE 14:44:52.006 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 14:45:16.239 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING Bond Events: Total Number of events: 1 Time address Function State 14:45:16.808 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING </pre>
<p>After the process</p>	<pre> Bond Events: Total Number of events: 4 Time BD_ADDR Function State 14:44:52.002 48:01:c5:86:27:ce btif_dm_create_bond BOND_STATE_NONE 14:44:52.006 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 14:45:16.239 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 14:45:30.348 48:01:c5:86:27:ce bond_state_changed BOND_STATE_NONE Bond Events: Total Number of events: 6 Time address Function State 14:45:16.808 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 14:45:30.744 44:80:eb:f0:ad:8f Invalid value BOND_STATE_BONDING 14:45:30.744 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_NONE 14:45:30.757 44:80:eb:f0:ad:8f btif_dm_remove_bond BOND_STATE_NONE 14:45:30.803 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_NONE 14:45:30.864 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_NONE </pre>
<p>What is visible after the process?</p>	<p>On the Nexus 6 we can see the creation of the request, along with the timestamp and MAC-address of the receiver. We can see when the Nexus 6 is ready to bond and when the OnePlus has received the request based on the second bond state change to bonding. This is after approximately 24 seconds. Another 14 seconds later we see that the bond state has been set to none. This corresponds with the denial on the OnePlus of the request.</p> <p>On the request we see when we received the request and who the sender is based on the MAC-address. We see that when we pressed cancel, the function "Invalid value" is being called. After which the bond state is set to none. Afterwards, we see that the bond request is being removed. This is done by the function "btif_dm_remove_bond". Thereupon, we see twice that the function "bond_state_changed" is being called. The state stays on "BOND_STATE_NONE".</p>
<p>Is it as expected?</p>	<p>No, what we thought we would see with the Nexus 6 as well as with the OnePlus are incorrect. The behaviour is even different compared to the previous test.</p> <p>On the Nexus 6 we expected to see some new behaviour when the request has been cancelled. In a way we see this, since the bond state is changed when we declined the request on the OnePlus. In the previous test we had similar behaviour to an time-out on the sending device. We did not see a new function being called or a new state when the request was denied.</p>

Bluetooth in Digital mobile Forensics
Communication between two Android devices

	<p>On the OnePlus we see a completely new behaviour of the artefact. We can see, as expected, the receipt of the request and all its info, as well as the denial of the request based on the function “<i>Invalid value</i>” being called followed by the change in bond state. What we see afterwards is unexpected. The OnePlus removes the bond request. This can be derived from the function “<i>btif_dm_remove_bond</i>” that is being called. Subsequently, twice the function “<i>bond_state_changed</i>” is being called with no effect on the bond state. We let to believe this is a new behaviour on Android 10 that is similar to when this function is being called immediately after the creation of a bond request. We expect that this causes the sender to know that the request has been denied and that the final call of this function is to finalise the process.</p> <p>Further research, preferably with two devices running Android 10³, to this behaviour is needed.</p>
--	---

8.7 Test 7: OnePlus sends request – incorrect pin

Test Case	An Android device (device A) sends a request to another device (device B). Device A accepts the request pin, but device B rejects the request.
Devices	Motorola Nexus 6 & OnePlus 7T Pro (44:80:eb:f0:ad:8f & 48:01:c5:86:27:ce)
Procedure	<p>We used the OnePlus to send a request to the Nexus 6. We accepted the pin on the screen (see Attachment 4: Pop-up when sending a request on Android) of the OnePlus (the sender).</p> <p>When the pin was confirmed on the OnePlus, we declined the request on the Nexus 6 by pressing cancel (see Attachment 5: Pop-up when receiving a request on Android).</p>
Monitored devices	Nexus 6 & OnePlus
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	<pre>Bond Events: Total Number of events: 0 Bond Events: Total Number of events: 0</pre>

³ Android 10 is at the moment of writing the most recent edition of the Android operating system.

Bluetooth in Digital mobile Forensics

Communication between two Android devices

<p>What is expected?</p>	<p>On the Nexus 6 we expected to see the receipt of the request with the according timestamp and the sender's MAC-address. When cancelling the request, we expected to see the function "Invalid value" being called, followed by a change in bond state to none.</p> <p><i>timestamp – BDaddr of OnePlus – change state – bonding</i> <i>timestamp – BDaddr of OnePlus – Invalid value – bonding</i> <i>timestamp – BDaddr of OnePlus – change state – none</i></p> <p>On the OnePlus we expected to see the creation of the request, with the MAC-address of the receiver of the request and the timestamp, followed by the change in bond state to bonding twice. We expected to see no change when we confirmed the request on the OnePlus. We expected to see the function "Invalid value" or a similar function to be called when we declined the request on the Nexus 6.</p> <p><i>timestamp – BDaddr of Nexus 6 – create bond – none</i> <i>timestamp – BDaddr of Nexus 6 – change state – bonding</i> <i>timestamp – BDaddr of Nexus 6 – change state – bonding</i> <i>timestamp – BDaddr of Nexus 6 – Invalid value – bonding</i> <i>timestamp – BDaddr of Nexus 6 – change state – none</i></p>
<p>During the process</p>	<pre>Bond Events: Total Number of events: 1 Time BD_ADDR Function State 16:56:58.325 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING</pre> <pre>Bond Events: Total Number of events: 3 Time address Function State 16:56:56.487 44:80:eb:f0:ad:8f btif_dm_create_bond BOND_STATE_NONE 16:56:56.487 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 16:56:59.335 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING</pre>
<p>After the process</p>	<pre>Bond Events: Total Number of events: 3 Time BD_ADDR Function State 16:56:58.325 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 16:57:11.807 48:01:c5:86:27:ce Invalid value BOND_STATE_BONDING 16:57:11.807 48:01:c5:86:27:ce bond_state_changed BOND_STATE_NONE</pre> <pre>Bond Events: Total Number of events: 4 Time address Function State 16:56:56.487 44:80:eb:f0:ad:8f btif_dm_create_bond BOND_STATE_NONE 16:56:56.487 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 16:56:59.335 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 16:57:12.900 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_NONE</pre>
<p>What is visible after the process?</p>	<p>On the Nexus 6 we see when we received the request and who was the sender. When we pressed cancel on the Nexus 6, we see the function "Invalid value" being called, followed by a change in bond state to none.</p> <p>On the OnePlus we can see when we created the request and who the receiver was based on the MAC-address. We see twice the</p>

Bluetooth in Digital mobile Forensics
Communication between two Android devices

	change in bond state. When the pin was rejected, we do not see any change in behaviour.
Is it as expected?	No, we expected to see a change in the artefact when the pin was rejected. We see no trace of an incorrect pin, the only indication that something went wrong is that the bond state is change to none.

8.8 Test 8: Nexus 6 sends request – incorrect pin

Test Case	An Android device (device B) sends a request to another device (device A). Device B accepts the request pin, but device A rejects the request.
Devices	Motorola Nexus 6 & OnePlus 7T Pro (44:80:eb:f0:ad:8f & 48:01:c5:86:27:ce)
Procedure	In this testcase we used the Nexus 6 as the sender of the request. The OnePlus operated as receiver. We accepted the pin of the sent request (see Attachment 4: Pop-up when sending a request on Android) on the Nexus 6. Once we confirmed the pin on the Nexus 6, we declined the request on the OnePlus by pressing cancel (see Attachment 5: Pop-up when receiving a request on Android).
Monitored devices	Nexus 6 & OnePlus
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	<pre>Bond Events: Total Number of events: 0</pre> <pre>Bond Events: Total Number of events: 0</pre>
What is expected?	<p>We expected to see the creation of the request on the Nexus 6 accompanied with the MAC-address of the OnePlus and the timestamp of creation. We expected to see the change in bond state to bonding twice. We expected to see a similar behaviour to the function “Invalid value” being called when we denied the request on the OnePlus.</p> <pre>timestamp – BDaddr – create bond – none timestamp – BDaddr – change state – bonding timestamp – BDaddr – change state – bonding timestamp – BDaddr – Invalid value – bonding timestamp – BDaddr – change state – none</pre>

Bluetooth in Digital mobile Forensics

Communication between two Android devices

	<p>On the OnePlus we expected to see the arrival of the request, with the timestamp and the sender's MAC-address, followed by a change in bond state to bonding. When declining the request, we expected the function "Invalid value" to be called, followed by a change in bond state, the removal of the request and the finalisation of the procedure designated with twice calling the function "bond_state_changed".</p> <p><i>timestamp – BDaddr of Nexus 6 – change state – bonding</i> <i>timestamp – BDaddr of Nexus 6 – Invalid value – bonding</i> <i>timestamp – BDaddr of Nexus 6 – change state – none</i> <i>timestamp – BDaddr of Nexus 6 – remove bond – none</i> <i>timestamp – BDaddr of Nexus 6 – change state – none</i> <i>timestamp – BDaddr of Nexus 6 – change state – none</i></p>
<p>During the process</p>	<pre>Bond Events: Total Number of events: 3 Time BD_ADDR Function State 17:02:06.370 48:01:c5:86:27:ce btif_dm_create_bond BOND_STATE_NONE 17:02:06.370 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 17:02:08.880 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING</pre> <pre>Bond Events: Total Number of events: 1 Time address Function State 17:02:09.365 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING</pre>
<p>After the process</p>	<pre>Bond Events: Total Number of events: 4 Time BD_ADDR Function State 17:02:06.370 48:01:c5:86:27:ce btif_dm_create_bond BOND_STATE_NONE 17:02:06.370 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 17:02:08.880 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 17:02:23.766 48:01:c5:86:27:ce bond_state_changed BOND_STATE_NONE</pre> <pre>Bond Events: Total Number of events: 6 Time address Function State 17:02:09.365 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 17:02:24.180 44:80:eb:f0:ad:8f Invalid value BOND_STATE_BONDING 17:02:24.181 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_NONE 17:02:24.196 44:80:eb:f0:ad:8f btif_dm_remove_bond BOND_STATE_NONE 17:02:24.239 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_NONE 17:02:24.251 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_NONE</pre>
<p>What is visible after the process?</p>	<p>On the Nexus 6 we can see the creation of the request, with the timestamp and who the receiver is. We can see twice the change in bond state and when we declined the request on the OnePlus, based on the timestamps. The only effect visible in the artefact is that the bond state has been changed to none. Remarkably, we saw a notification on the Nexus 6 screen when we cancelled the request that mentioned that the pin was incorrect (see Attachment 6: Pop-up on Nexus 6 when pin is incorrect).</p> <p>On the OnePlus we can see when we received the request and who the sender of the request was based on the change in bond state. We can see when we denied the request based on the function</p>

Bluetooth in Digital mobile Forensics
Communication between two Android devices

	<p><i>“Invalid value”</i> being called. This causes a change in bond state to none. As we observed in Test 6: Nexus 6 sends request – denied, we can see that the OnePlus starts a procedure of removing the bond request, followed by twice a change in bond state. If we assume that, when we have send a request, the double change in bond state is once on the sender/creator of the artefact and once for the receiver, is the returning of the change in bond state similar to this and does this cause the notification we saw in Attachment 6: Pop-up on Nexus 6 when pin is incorrect? Further research with two devices, that are preferably running Android 10, is needed.</p>
Is it as expected?	<p>No, we expected a different behaviour on the Nexus 6 when we declined the request on the OnePlus. Our other assumptions concerning the behaviour of the artefact are correct.</p>

8.9 Test 9: OnePlus sends request – accepted

Test Case	An Android device (device A) sends a request to another Android device (device B). Both devices accept the request.
Devices	Motorola Nexus 6 & OnePlus 7T Pro (44:80:eb:f0:ad:8f & 48:01:c5:86:27:ce)
Procedure	We used the OnePlus to send a request to the Nexus 6. We accepted the request on both devices. Both devices are now paired/bonded.
Monitored devices	Nexus 6 & OnePlus
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	<pre>Bond Events: Total Number of events: 0 Bond Events: Total Number of events: 0</pre>
What is expected?	<p>On the Nexus 6 we expected to see the arrival of the request, accompanied with its timestamp and the MAC-address of the sender. We expected to see, once the request has been accepted, a change in bond state to bonded.</p> <p><i>timestamp – BDaddr of OnePlus – change state – bonding</i> <i>timestamp – BDaddr of OnePlus – change state – bonded</i></p> <p>On the OnePlus we expected to see the creation of the request with accordantly the MAC-address of the Nexus 6 and the timestamp on which the request was created. We expect to see twice the change in</p>

Bluetooth in Digital mobile Forensics

Communication between two Android devices

	<p>bond state to bonding. When the request has been accepted, we expect to see a change in bond state to bonded.</p> <p><i>timestamp – BDaddr of Nexus 6 – bond created – none</i> <i>timestamp – BDaddr of Nexus 6 – change state – bonding</i> <i>timestamp – BDaddr of Nexus 6 – change state – bonding</i> <i>timestamp – BDaddr of Nexus 6 – change state – bonded</i></p>
<p>During the process</p>	<pre> Bond Events: Total Number of events: 1 Time BD_ADDR Function State 17:54:59.339 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING </pre> <pre> Bond Events: Total Number of events: 3 Time address Function State 17:54:58.220 44:80:eb:f0:ad:8f btif_dm_create_bond BOND_STATE_NONE 17:54:58.220 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 17:54:59.729 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING </pre>
<p>After the process</p>	<pre> Bond Events: Total Number of events: 3 Time BD_ADDR Function State 17:54:59.339 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 17:55:13.736 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDED 17:55:14.840 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDED </pre> <pre> Bond Events: Total Number of events: 5 Time address Function State 17:54:58.220 44:80:eb:f0:ad:8f btif_dm_create_bond BOND_STATE_NONE 17:54:58.220 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 17:54:59.729 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 17:55:14.126 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDED 17:55:15.092 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDED </pre>
<p>What is visible after the process?</p>	<p>On the Nexus 6 we can see when we received the request, and who the sender was based on the MAC-address. We observe twice a change in bond state to bonded when the request was accepted.</p> <p>On the OnePlus we can see when we created the request, and who the receiver was based on the MAC-address. We see twice a change to bonding and twice a change to bonded when we accepted the request.</p> <p>Based on previous findings, we can assume that the phenomenon of the double changes in bond state to bonded is once to communicate to the other device that the request has been accepted and once to change to bond state itself on the device.</p>
<p>Is it as expected?</p>	<p>No, we did not expect to see the double change in bond state to bonded. Our other assumptions about the behaviour of the artefact are correct.</p>

Bluetooth in Digital mobile Forensics
Communication between two Android devices

8.10 Test 10: Forget Device

Test Case	Both Android devices are connected. We forget the connection between both devices.
Devices	Motorola Nexus 6 & OnePlus 7T Pro (44:80:eb:f0:ad:8f & 48:01:c5:86:27:ce)
Procedure	We forget the connection between the two devices. This is done first on the Nexus 6. We go into the Bluetooth settings and search for the OnePlus. We pressed the advanced options (gear button) and chose to forget the device (see Attachment 7: Forgetting a paired device on Android). Afterwards, we do the same on the OnePlus since the device do not know that the other device deleted the know device.
Monitored devices	Nexus 6 & OnePlus
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	<pre> Bond Events: Total Number of events: 3 Time BD_ADDR Function State 17:54:59.339 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 17:55:13.736 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDED 17:55:14.840 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDED </pre> <pre> Bond Events: Total Number of events: 5 Time address Function State 17:54:58.220 44:80:eb:f0:ad:8f btif_dm_create_bond BOND_STATE_NONE 17:54:58.220 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 17:54:59.729 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 17:55:14.126 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDED 17:55:15.092 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDED </pre>
What is expected?	<p>We expected to see the same on both devices. We expected to see the removal of the bond and a change in bond state to none on both devices.</p> <p><i>timestamp – BDaddr of OnePlus – remove bond – none</i> <i>timestamp – BDaddr of OnePlus – change state – none</i></p> <p><i>timestamp – BDaddr of Nexus 6 – remove bond – none</i> <i>timestamp – BDaddr of Nexus 6 – change state – none</i></p>

Bluetooth in Digital mobile Forensics

Communication between two Android devices

<p>After the process of forgetting the connection on Nexus 6</p>	<pre> Bond Events: Total Number of events: 6 Time BD_ADDR Function State 17:54:59.339 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 17:55:13.736 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDED 17:55:14.840 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDED 18:04:09.393 48:01:c5:86:27:ce btif_dm_remove_bond BOND_STATE_NONE 18:04:09.434 48:01:c5:86:27:ce bond_state_changed BOND_STATE_NONE 18:04:09.459 48:01:c5:86:27:ce bond_state_changed BOND_STATE_NONE Bond Events: Total Number of events: 5 Time address Function State 17:54:58.220 44:80:eb:f0:ad:8f btif_dm_create_bond BOND_STATE_NONE 17:54:58.220 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 17:54:59.729 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 17:55:14.126 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDED 17:55:15.092 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDED </pre>
<p>After the process of forgetting the connection on OnePlus</p>	<pre> Bond Events: Total Number of events: 6 Time BD_ADDR Function State 17:54:59.339 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 17:55:13.736 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDED 17:55:14.840 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDED 18:04:09.393 48:01:c5:86:27:ce btif_dm_remove_bond BOND_STATE_NONE 18:04:09.434 48:01:c5:86:27:ce bond_state_changed BOND_STATE_NONE 18:04:09.459 48:01:c5:86:27:ce bond_state_changed BOND_STATE_NONE Bond Events: Total Number of events: 7 Time address Function State 17:54:58.220 44:80:eb:f0:ad:8f btif_dm_create_bond BOND_STATE_NONE 17:54:58.220 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 17:54:59.729 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 17:55:14.126 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDED 17:55:15.092 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDED 18:04:53.951 44:80:eb:f0:ad:8f btif_dm_remove_bond BOND_STATE_NONE 18:04:53.969 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_NONE </pre>
<p>What is visible after the process?</p>	<p>On the Nexus 6 we see the removal of the bond, followed by twice the change in bond state.</p> <p>On the OnePlus we do not see any behaviour indicating that the bond has been deleted. However, after we removed the bond on the OnePlus, we see only one change in bond state.</p>
<p>Is it as expected?</p>	<p>No, we expected to see the same behaviour on both devices, but we do not see the same behaviour. On the Nexus 6 we see twice the change in bond to none. This is remarkable.</p>

Bluetooth in Digital mobile Forensics
Communication between two Android devices

8.11 Test 11: Nexus 6 sends request – accepted

Test Case	An Android device (device B) sends a request to another Android device (device A). Both devices accept the request.
Devices	Motorola Nexus 6 & OnePlus 7T Pro (44:80:eb:f0:ad:8f & 48:01:c5:86:27:ce)
Procedure	With the Nexus we have sent a request to the OnePlus. We accepted the request on both devices.
Monitored devices	Nexus 6 & OnePlus
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	<pre>Bond Events: Total Number of events: 0</pre> <pre>Bond Events: Total Number of events: 0</pre>
What is expected?	<p>We expected to see the creation of the request on the Nexus 6 accompanied with the timestamp of creation and the MAC-address of the OnePlus. We expected this to be followed by twice a change in bond state to bonding. When the request is accepted we expected a change in bond state to bonded.</p> <p><i>timestamp – BDaddr of OnePlus – create bond – none</i> <i>timestamp – BDaddr of OnePlus – change state – bonding</i> <i>timestamp – BDaddr of OnePlus – change state – bonding</i> <i>timestamp – BDaddr of OnePlus – change state – bonded</i></p> <p>On the OnePlus we expected to see the arrival of the request, with the MAC-address of the Nexus 6 and the timestamp of arrival. We expected to see a change in bond state to bonded when the request is accepted.</p> <p><i>timestamp – BDaddr of Nexus 6 – change state – bonding</i> <i>timestamp – BDaddr of Nexus 6 – change state – bonded</i></p>
During the process	<pre>Bond Events: Total Number of events: 3 Time BD_ADDR Function State 16:19:49.533 48:01:c5:86:27:ce btif_dm_create_bond BOND_STATE_NONE 16:19:49.534 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 16:19:54.920 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING</pre> <pre>Bond Events: Total Number of events: 1 Time address Function State 16:19:56.189 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING</pre>

Bluetooth in Digital mobile Forensics
Communication between two Android devices

<p>After the process</p>	<pre> Bond Events: Total Number of events: 4 Time BD_ADDR Function State 16:19:49.533 48:01:c5:86:27:ce btif_dm_create_bond BOND_STATE_NONE 16:19:49.534 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 16:19:54.920 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 16:20:03.107 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDED Bond Events: Total Number of events: 2 Time address Function State 16:19:56.189 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 16:20:04.378 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDED </pre>
<p>What is visible after the process?</p>	<p>On the Nexus 6 we see the creation of the request accompanied with the timestamp and the MAC-address of the receiver. We see twice the change in bond state to bonding. Once the request has been accepted, we see once the change of bond state to bonded.</p> <p>On the OnePlus we see the arrival of the request with its timestamp and the MAC-address of the sender. Once the request is accepted, the bond state is set to bonded.</p>
<p>Is it as expected?</p>	<p>Yes, we expected this behaviour.</p>

8.12 Test 12: Reconnect devices

<p>Test Case</p>	<p>Both devices are connected but out of range of each other. We bring them back together and try to connect them again.</p>
<p>Devices</p>	<p>Motorola Nexus 6 & OnePlus 7T Pro (44:80:eb:f0:ad:8f & 48:01:c5:86:27:ce)</p>
<p>Procedure</p>	<p>For this test, we did not reboot the devices.</p> <p>Both devices were coupled before the start of the test. We brought the device out of the range. We then brought them out of range and back in range and connected them with each other again.</p> <p>We checked if the Nexus 6 was out of range of the OnePlus based on the connection of a smartwatch (Fossil Q Marshal) that was connected to the OnePlus.</p>
<p>Monitored devices</p>	<p>Nexus 6 & OnePlus</p>
<p>How is it monitored?</p>	<p><i>\$ adb shell dumpsys bluetooth_manager</i></p>
<p>What is monitored?</p>	<p>Bond events</p>

Bluetooth in Digital mobile Forensics

Communication between two Android devices

<p>Start scenario</p>	<pre> Bond Events: Total Number of events: 4 Time BD_ADDR Function State 16:19:49.533 48:01:c5:86:27:ce btif_dm_create_bond BOND_STATE_NONE 16:19:49.534 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 16:19:54.920 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 16:20:03.107 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDED Bond Events: Total Number of events: 2 Time address Function State 16:19:56.189 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 16:20:04.378 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDED </pre>
<p>What is expected?</p>	<p>We expected to see on both device the same. We expected to see a change in bond state to none when they are out of range. Once they are in range again and connected, we expect to see a change in bond state to bonded.</p> <p><i>timestamp – BDaddr of OnePlus – change state – none</i> <i>timestamp – BDaddr of OnePlus – change state – bonded</i></p> <p><i>timestamp – BDaddr of Nexus 6 – change state – none</i> <i>timestamp – BDaddr of Nexus 6 – change state – bonded</i></p>
<p>After losing connection</p>	<pre> Bond Events: Total Number of events: 4 Time BD_ADDR Function State 16:19:49.533 48:01:c5:86:27:ce btif_dm_create_bond BOND_STATE_NONE 16:19:49.534 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 16:19:54.920 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 16:20:03.107 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDED Bond Events: Total Number of events: 2 Time address Function State 16:19:56.189 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 16:20:04.378 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDED </pre>
<p>After the process</p>	<pre> Bond Events: Total Number of events: 4 Time BD_ADDR Function State 16:19:49.533 48:01:c5:86:27:ce btif_dm_create_bond BOND_STATE_NONE 16:19:49.534 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 16:19:54.920 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 16:20:03.107 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDED Bond Events: Total Number of events: 2 Time address Function State 16:19:56.189 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 16:20:04.378 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDED </pre>
<p>What is visible after the process?</p>	<p>We see no change in the artefact.</p>
<p>Is it as expected?</p>	<p>No, we expected to see changes in the artefact.</p>

Bluetooth in Digital mobile Forensics
Communication between two Android devices

8.13 Test 13: Forgetting Device and attempted to Reconnect

Test Case	Both devices (A&B) are connected. We forget the connection on device A and try to reconnect with device B.
Devices	Motorola Nexus 6 & OnePlus 7T Pro (44:80:eb:f0:ad:8f & 48:01:c5:86:27:ce)
Procedure	For this test, we did not reboot the devices. At the start of the test, both devices were connected. On the OnePlus, we forgot the connection (see Attachment 7: Forgetting a paired device on Android). Afterwards, we tried to connect both devices again via the Nexus 6. We did not interact with the request (this is done in the next test case, test 14).
Monitored devices	Nexus 6 & OnePlus
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	<pre> Bond Events: Total Number of events: 4 Time BD_ADDR Function State 16:19:49.533 48:01:c5:86:27:ce btif_dm_create_bond BOND_STATE_NONE 16:19:49.534 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 16:19:54.920 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 16:20:03.107 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDED </pre> <pre> Bond Events: Total Number of events: 2 Time address Function State 16:19:56.189 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 16:20:04.378 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDED </pre>
What is expected?	<p>We did not expected to see a change on the Nexus 6 when the OnePlus forgot the connection, with the exception of the bond state changing to none. When the Nexus 6 tries to connect again, we expected first to see a change in bond state, followed by the creation of a bond request with twice the function “<i>bond_state_changed</i>” to be called.</p> <p><i>timestamp – BDaddr of OnePlus – change state – none</i> <i>timestamp – BDaddr of OnePlus – change state – bonding</i> <i>timestamp – BDaddr of OnePlus – create bond – none</i> <i>timestamp – BDaddr of OnePlus – change state – bonding</i> <i>timestamp – BDaddr of OnePlus – change state – bonding</i></p> <p>We expected to see on the OnePlus the removal of the bond followed by a change in bond state. Once the Nexus 6 tries to bond again, we expect to see the same as we saw in the other tests when the Nexus sent a request to bond to the OnePlus.</p>

Bluetooth in Digital mobile Forensics

Communication between two Android devices

	<p><i>timestamp – BDaddr of Nexus 6 – remove bond – none</i> <i>timestamp – BDaddr of Nexus 6 – change state – none</i> <i>timestamp – BDaddr of Nexus 6 – change state – bonding</i></p>
<p>After forgetting the connection</p>	<pre> Bond Events: Total Number of events: 4 Time BD_ADDR Function State 16:19:49.533 48:01:c5:86:27:ce btif_dm_create_bond BOND_STATE_NONE 16:19:49.534 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 16:19:54.920 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 16:20:03.107 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDED Bond Events: Total Number of events: 4 Time address Function State 16:19:56.189 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 16:20:04.378 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDED 16:24:33.539 44:80:eb:f0:ad:8f btif_dm_remove_bond BOND_STATE_NONE 16:24:33.556 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_NONE </pre>
<p>After the process (when request is send)</p>	<pre> Bond Events: Total Number of events: 5 Time BD_ADDR Function State 16:19:49.533 48:01:c5:86:27:ce btif_dm_create_bond BOND_STATE_NONE 16:19:49.534 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 16:19:54.920 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 16:20:03.107 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDED 16:28:21.553 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING Bond Events: Total Number of events: 5 Time address Function State 16:19:56.189 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 16:20:04.378 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDED 16:24:33.539 44:80:eb:f0:ad:8f btif_dm_remove_bond BOND_STATE_NONE 16:24:33.556 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_NONE 16:28:22.825 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING </pre>
<p>What is visible after the process?</p>	<p>We see that the Nexus 6 does not know that when the OnePlus forgot the connection. When the Nexus 6 tries to reconnect, we see a change in state to bonding. We see the same on the OnePlus when the request arrives.</p> <p>We can clearly see on the OnePlus what function is called when we forget the request (<i>btif_dm_remove_bond</i>) and that afterwards the function <i>bond_state_changed</i> has been called, changing the state to none.</p>
<p>Is it as expected?</p>	<p>No, we expected to see a change on the Nexus 6 when the bond was removed on the OnePlus. When the request to reconnect was sent, we expected to see the creation of a new request, but we do not see this. We see only once the change in bond state to bonding.</p> <p>This behaviour is remarkably different than what we expected, bases on the previous tests.</p>

Bluetooth in Digital mobile Forensics
Communication between two Android devices

8.14 Test 14: Accepting the reconnection request

Test Case	Device A accepts the request of device B.
Devices	Motorola Nexus 6 & OnePlus 7T Pro (44:80:eb:f0:ad:8f & 48:01:c5:86:27:ce)
Procedure	We accepted the reconnect request from the previous test. Thus the devices were not rebooted.
Monitored devices	Nexus 6 & OnePlus
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	<pre> Bond Events: Total Number of events: 5 Time BD_ADDR Function State 16:19:49.533 48:01:c5:86:27:ce btif_dm_create_bond BOND_STATE_NONE 16:19:49.534 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 16:19:54.920 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 16:20:03.107 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDED 16:28:21.553 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING </pre> <pre> Bond Events: Total Number of events: 5 Time address Function State 16:19:56.189 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 16:20:04.378 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDED 16:24:33.539 44:80:eb:f0:ad:8f btif_dm_remove_bond BOND_STATE_NONE 16:24:33.556 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_NONE 16:28:22.825 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING </pre>
What is expected?	<p>On both devices we expect to see a change in bond state to bonded twice.</p> <p><i>timestamp – BDaddr of OnePlus – change state – bonded</i> <i>timestamp – BDaddr of OnePlus – change state – bonded</i></p> <p><i>timestamp – BDaddr of Nexus 6 – change state – bonded</i> <i>timestamp – BDaddr of Nexus 6 – change state – bonded</i></p>
After the process	<pre> Bond Events: Total Number of events: 7 Time BD_ADDR Function State 16:19:49.533 48:01:c5:86:27:ce btif_dm_create_bond BOND_STATE_NONE 16:19:49.534 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 16:19:54.920 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 16:20:03.107 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDED 16:28:21.553 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 16:28:35.306 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDED 16:28:36.435 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDED </pre>

Bluetooth in Digital mobile Forensics
Communication between two Android devices

	<pre>Bond Events: Total Number of events: 7 Time address Function State 16:19:56.189 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 16:20:04.378 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDED 16:24:33.539 44:80:eb:f0:ad:8f btif_dm_remove_bond BOND_STATE_NONE 16:24:33.556 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_NONE 16:28:22.825 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 16:28:36.575 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDED 16:28:37.687 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDED</pre>
What is visible after the process?	<p>On the Nexus 6 we see twice a change in bond state to bonded. The same is visible on the OnePlus. When we compare the timestamps, we see that that both are approximately 1 second apart. This is within margin of error in time between the two devices and may be linked to latency with Bluetooth communication. The fact we see twice the bond state changing to bonded indicates that both devices know when the other device has accepted the connection, following our conclusion of Test 9: OnePlus sends request – accepted.</p>
Is it as expected?	<p>Yes, we see the expected behaviour.</p>

8.15 Test 15: Sending a file from the OnePlus

Test Case	Both devices are connected with each other via Bluetooth. We transfer a photo over Bluetooth from device A to device B.
Devices	Motorola Nexus 6 & OnePlus 7T Pro (44:80:eb:f0:ad:8f & 48:01:c5:86:27:ce)
Procedure	Both devices were connected via Bluetooth before the start of the test. We share a photo (see Attachment 8: Photo used in testing to send between devices) from the OnePlus' gallery to the Nexus 6.
Monitored devices	Nexus 6 & OnePlus
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	<pre>Bond Events: Total Number of events: 0</pre> <pre>Bond Events: Total Number of events: 0</pre>
What is expected?	<p>On the Nexus 6 we expect to see the arrival of a special bond request for file sharing. We expected to see a change in bond state to bonded and transfer the file. Once the file is transferred, we expected to see a change in bond state to none.</p> <p><i>timestamp – BDaddr of OnePlus – change state – bonding</i></p>

Bluetooth in Digital mobile Forensics

Communication between two Android devices

	<p><i>timestamp – BDaddr of OnePlus – change state – bonded</i> <i>timestamp – BDaddr of OnePlus – change state – receiving</i> <i>timestamp – BDaddr of OnePlus – change state – none</i></p> <p>On the OnePlus we expected to see the creation of a special bond request for file transferring. Followed by twice a change in bond state before an indication of the file transfer. After the file transfer, we expect to see the bond state to be set to none.</p> <p><i>timestamp – BDaddr of Nexus 6 – create transfer bond – none</i> <i>timestamp – BDaddr of Nexus 6 – change state – bonding</i> <i>timestamp – BDaddr of Nexus 6 – change state – bonded</i> <i>timestamp – BDaddr of Nexus 6 – change state – transferring</i> <i>timestamp – BDaddr of Nexus 6 – change state – none</i></p>
<p>During the process</p>	<pre>Bond Events: Total Number of events: 2 Time BD_ADDR Function State 14:45:33.218 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 14:45:33.842 48:01:c5:86:27:ce bond_state_changed BOND_STATE_NONE</pre> <pre>Bond Events: Total Number of events: 2 Time address Function State 14:45:32.850 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 14:45:33.487 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_NONE</pre>
<p>After the process</p>	<pre>Bond Events: Total Number of events: 2 Time BD_ADDR Function State 14:45:33.218 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 14:45:33.842 48:01:c5:86:27:ce bond_state_changed BOND_STATE_NONE</pre> <pre>Bond Events: Total Number of events: 2 Time address Function State 14:45:32.850 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 14:45:33.487 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_NONE</pre>
<p>What is visible after the process?</p>	<p>When we chose the receiving device of our photo, the two devices went to the bonding state. When the device has accepted the incoming request to receive a photo, the state was set to none.</p>
<p>Is it as expected?</p>	<p>No, we expected to see a new behaviour when the photo was transmitting and when the photo was received the change in bond state to none.</p>

Bluetooth in Digital mobile Forensics
Communication between two Android devices

8.16 Test 16: Sending a file from the Nexus 6

Test Case	We share a photo from device B to device A. The devices are not connected with each other.
Devices	Motorola Nexus 6 & OnePlus 7T Pro (44:80:eb:f0:ad:8f & 48:01:c5:86:27:ce)
Procedure	We shared a photo from the Nexus 6 gallery with the OnePlus over Bluetooth. The devices were not connected and not present in the list of known Bluetooth devices. The photo can be found in Attachment 8: Photo used in testing to send between devices.
Monitored devices	Nexus 6 & OnePlus
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	<pre>Bond Events: Total Number of events: 0</pre> <pre>Bond Events: Total Number of events: 0</pre>
What is expected?	<p>We expected to see the creation of a special bond request on the Nexus 6, followed by twice a change in bonding state to bonding followed by a change to bonded, before transferring the photo. After the photo is transferred, we expected to see a change in bond state to none.</p> <p><i>timestamp – BDaddr of OnePlus – create transfer bond – none</i> <i>timestamp – BDaddr of OnePlus – change state – bonding</i> <i>timestamp – BDaddr of OnePlus – change state – bonding</i> <i>timestamp – BDaddr of OnePlus – change state – bonded</i> <i>timestamp – BDaddr of OnePlus – change state – transferring</i> <i>timestamp – BDaddr of OnePlus – change state – none</i></p> <p>On the OnePlus we expected to see the arrival of the request, followed by a change in bond state to bonding. Then a change to bonded followed by an indication that a file is being transferred. After the transfer, we expected that the bond state is set to none.</p> <p><i>timestamp – BDaddr of Nexus 6 – change state – bonding</i> <i>timestamp – BDaddr of Nexus 6 – change state – bonded</i> <i>timestamp – BDaddr of Nexus 6 – change state – receiving</i> <i>timestamp – BDaddr of Nexus 6 – change state – none</i></p>

Bluetooth in Digital mobile Forensics
Communication between two Android devices

<p>During the process</p>	<pre> Bond Events: Total Number of events: 2 Time BD_ADDR Function State 14:53:47.305 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 14:53:47.871 48:01:c5:86:27:ce bond_state_changed BOND_STATE_NONE </pre> <pre> Bond Events: Total Number of events: 2 Time address Function State 14:53:46.405 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 14:53:46.947 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_NONE </pre>
<p>After the process</p>	<pre> Bond Events: Total Number of events: 2 Time BD_ADDR Function State 14:53:47.305 48:01:c5:86:27:ce bond_state_changed BOND_STATE_BONDING 14:53:47.871 48:01:c5:86:27:ce bond_state_changed BOND_STATE_NONE </pre> <pre> Bond Events: Total Number of events: 2 Time address Function State 14:53:46.405 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_BONDING 14:53:46.947 44:80:eb:f0:ad:8f bond_state_changed BOND_STATE_NONE </pre>
<p>What is visible after the process?</p>	<p>When we chose our receiving device, on both device, the bonding state was set to bonding. When we accepted the incoming request to receive a photo, the bond state was set to none.</p>
<p>Is it as expected?</p>	<p>No, we expected to see a new behaviour when the photo was transmitting and when the photo was received, the change in bond state to none. We also expected to see the creation of a bond request since the two devices were not connected at the start of this test.</p>

Bluetooth in Digital mobile Forensics
Communication between smartphone and audio player

9 Communication between smartphone and audio player

Since testing the behaviour of the artefact when two mobile phones that are trying to reconnect is not easily observable, we test this case with a Bluetooth enabled audio player. For this test-case we used the UUV Airdot wireless in-ear headphones.

Between each testcase we reboot the smartphone.

9.1 Bonding the audio player with the smartphone

Test Case	An Android device is paired with the audio player
Devices	Motorola Nexus 6 & UUV Airdot wireless in-ear headphones (44:80:eb:f0:ad:8f & ca:71:06:fe:ed:a3)
Procedure	The Nexus 6 Bluetooth settings were opened in order to be able to pair with the Airdots. When the Airdots showed up, we connected the devices.
Monitored devices	Nexus 6
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	<pre>Bond Events: Total Number of events: 0</pre>
What is expected?	We expected to see the creation of a pairing request, followed by a change in bond state. We expected to see a similar behaviour as we saw between two smartphones. Once paired, we expected to see the bond state to be set to bonded. <i>timestamp – BDaddr of audio player – create bond – none</i> <i>timestamp – BDaddr of audio player – change state – bonding</i> <i>timestamp – BDaddr of audio player – change state – bonding</i> <i>timestamp – BDaddr of audio player – change state – bonded</i>
After the process	<pre>Bond Events: Total Number of events: 4 Time BD_ADDR Function State 15:04:40.817 ca:71:06:fe:ed:a3 btif_dm_create_bond BOND_STATE_NONE 15:04:40.820 ca:71:06:fe:ed:a3 bond_state_changed BOND_STATE_BONDING 15:04:43.889 ca:71:06:fe:ed:a3 bond_state_changed BOND_STATE_BONDING 15:04:44.659 ca:71:06:fe:ed:a3 bond_state_changed BOND_STATE_BONDED</pre>
What is visible after the process?	We can see the timestamp when we created the bond request and the MAC-address of the receiver (audio player). We can see that twice the change in bond state to bonding and when paired, the change in state to bonded.
Is it as expected?	Yes, we expected to see this behaviour.

Bluetooth in Digital mobile Forensics
Communication between smartphone and audio player

9.2 Reconnecting the audio player with the smartphone

Test Case	The audio player is powered on and is connection with the smartphone
Devices	Motorola Nexus 6 & UVV Airdot wireless in-ear headphones (44:80:eb:f0:ad:8f & ca:71:06:fe:ed:a3)
Procedure	The Airdots were powered on and connected automatically with the Nexus 6.
Monitored devices	Nexus 6
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	<code>Bond Events: Total Number of events: 0</code>
What is expected?	We expected to see the bond state changing to bonded. <i>timestamp – BDaddr of audio player – change state – bonded</i>
After the process	<code>Bond Events: Total Number of events: 0</code>
What is visible after the process?	We see no change in the artefact.
Is it as expected?	No, we expected to see a change in the bond state to bonded.

9.3 Forgetting the audio player

Test Case	Smartphone forgets the audio player being bonded
Devices	Motorola Nexus 6 & UVV Airdot wireless in-ear headphones (44:80:eb:f0:ad:8f & ca:71:06:fe:ed:a3)
Procedure	Via the advanced settings in the Bluetooth menu of the Nexus 6 we forgot the connection with the Airdots (see Attachment 7: Forgetting a paired device on Android).
Monitored devices	Nexus 6
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	<code>Bond Events: Total Number of events: 0</code>

Bluetooth in Digital mobile Forensics

Communication between smartphone and audio player

What is expected?	<p>We expected to see the removal of the bond and the bond state to be none.</p> <p><i>timestamp – BDaddr of audio player – remove bond – none</i> <i>timestamp – BDaddr of audio player – remove bond – none</i></p>
After the process	<pre>Bond Events: Total Number of events: 2 Time BD_ADDR Function State 15:09:59.597 ca:71:06:fe:ed:a3 btif_dm_remove_bond BOND_STATE_NONE 15:09:59.633 ca:71:06:fe:ed:a3 bond_state_changed BOND_STATE_NONE</pre>
What is visible after the process?	<p>We can see the removal on the bond and the bond state being set to none.</p>
Is it as expected?	<p>Yes, we expected this behaviour.</p>

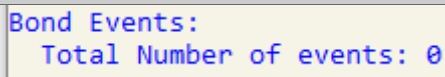
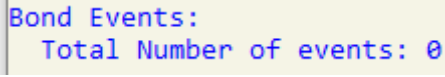
Bluetooth in Digital mobile Forensics

Communication between an Android smartphone and smartwatch

10 Communication between an Android smartphone and smartwatch

This test is to observe what happens when an Android smartphone and a Wear OS smartwatch reconnect with each other and if we can see this in the discovered artefact.

It was mainly preformed since we wanted to know what happens with known devices, especially devices that communicate over Bluetooth and have a variety of functions like smartwatches and the infotainment system of a car. We wanted to know if we could see the devices reconnecting in the artefact that we are studying. Since we have no car at our disposal that has such infotainment system, we conducted this test with a smartwatch.

Test Case	Smartwatch and smartphone are connected with each other but out of range. The two devices are back in range and connect automatically with each other.
Devices	OnePlus 7T Pro & Fossil Q Marshal (48:01:c5:86:27:ce & fc:45:96:6a:35:65)
Procedure	Both devices were connected before the start of the test. At the start of the test, both devices were out of range of each other. We bring them back into each other's range and they connect automatically.
Monitored devices	OnePlus
How is it monitored?	<code>\$ adb shell dumpsys bluetooth_manager</code>
What is monitored?	Bond events
Start scenario	
What is expected?	We expected to see the bond state change to bonded. <i>timestamp – BDaddr of smartwatch – change state – bonded</i>
After the process	
What is visible after the process?	Nothing is visible in the section of Bond Events.
Is it as expected?	No, we expected to see a change in bond state when the two devices reconnected.

11 Artefact lifespan

The value of an artefact for DFIR heavily depends on the lifespan of that artefact. When an artefact, for example, has a lifespan of a few minutes, the value of that artefact is negligible.

To determine the value of *dumpsys*, we analysed the lifespan of the aforementioned modules in chapter 6: The Value of *dumpsys*, namely the modules *accounts*, *content* and *Bluetooth_manager*.

11.1 Accounts

The module *accounts*, we saw all the accounts known on a device with their respective services and applications. These entries stay present as long the app is installed. If an app is installed, but not signed in (cfr. Figure 13 - Example of registered accounts on a device, entry WhatsApp), we can see the presence of the application without an account.

These observations translate to a valuable artefact in a DFIR investigation. Usually when an application is used, the user signs in the first time that he/she uses the application. Afterwards, the user stays logged in. When a device is confiscated, the possibility that the user is not signed in to an app, is slim.

11.2 Content

The module *content* includes mainly the synchronisation history of the services with the known accounts. Only the most recent synchronisation is recorded. When this synchronisation was, depends on the service that is present.

These (control) data stays present, which means that it is a valuable source for a DFIR.

11.3 Bluetooth_manager

Since *Bluetooth_manager* is a module within *dumpsys* that is specific to one service, namely the Bluetooth service, we expect an influence to the artefacts of *Bluetooth_manager* when the service is stopped. Stopping this service can be done in different ways. One could switch off Bluetooth, aeroplane mode could be enabled or the device could be powered off or restarted.

We observed that when the Bluetooth process was stopped, *Bluetooth_manager* is resetted. This means that all information contained in the *dumpsys* module *Bluetooth_manager* is lost, except for, among other things⁴, the list of bonded devices.

On the OnePlus, we observed a different behaviour compared to the Nexus 6. When the OnePlus entered a power saving state when the device wasn't in use, for example during the night on battery power when its user is sleeping and not using the device, the Bluetooth process was stopped, causing *Bluetooth_manager* to reset. This behaviour is observed even when the device was connected over Bluetooth with a smartwatch. Due to the limited number of devices, we were unable to verify if this behaviour is caused by new power saving

⁴ Not all section of *Bluetooth_manager* have been researched. Not all sections contained relevant data for our research. This also means that no attention was given to these sections when researching the lifespan of the sections we were interested in.

Bluetooth in Digital mobile Forensics

Artefact lifespan

measurements in Android 10, or if it is a characteristic of the device itself, caused by an optimisation performed on the software of the OnePlus. Further research concerning this matter has to be undertaken. Preferably with devices running Android 10.

The value of the studied artefact, the section "*Bond Events*" within the *dumpsys* module *Bluetooth_manager*, heavily depends on the device. Nevertheless it is definitely worth of considering investigating when conducting a DF investigation.

12 Critical reflection on the tests

There are some remarks about our testing procedure to be made, and why certain choices were made.

We didn't use the Tsurugi VM for the tests with the OnePlus. This is because Linux has no supporting drivers at the moment of testing for this device. This forced us to use the Windows O.S. of the host machine (Dell XPS). There is a benefit to use a Windows VM instead. Since we had a separate machine to communicate with each device, it was better manageable to communicate to the devices using adb. If both devices were connected to the same machine, it could have been a potential point of error. We could communicate to the wrong device.

Another remark that we have to make is the following. We saw a noticeable difference in the information we received out of the output of Bluetooth_manager between the two Android devices. We could extract more information out of the dump from the OnePlus compared to the one of the Nexus 6. Since we had only a limited number of devices, that both run almost vanilla Android, it could be beneficial to compare the output with other devices. Potentially other manufactures made changes on how Bluetooth works on their device and could some artefacts behave differently. We dive deeper in the differences in the dump between the two test devices in chapter 14 - Differences between Android versions.

Of all the sections that are present in the output of Bluetooth_manager, we focused us only on bond events. We didn't look for other changes in the output in any test case. The test with unexpected behaviour, especially those where we saw no change in the artefact, may had an influence on other data of the output of Bluetooth_manager. It might be possible that useful or interesting data could be found in these scenarios elsewhere in Bluetooth_manager. As we set out to determine if an artefact could be found when a connection attempt was conducted, this was out of scope of the current research and we did not investigate this possibility.

It might be noticed that there were no tests between the Nexus 6 and iPhone regarding sending a file between these over Bluetooth. We could not find an obvious way to do this. That is the reason that there were no test cases regarding this matter.

We do not expect noteworthy changes caused by using adb that could impact the systems integrity.

During our research, we did not look into the filesystem and file structure of the devices.

As mentioned earlier, both Android devices are my personal devices. They did not receive a reset before testing. This was done to recreate a real investigation. The benefit in our research is that we discovered the possibilities of dumpsys, e.g. how all accounts that are known on the device could be found as mentioned in chapter 6 - The Value of dumpsys.

⁵ Vanilla Android: stock Android. The OnePlus' OxygenOS has a few extra features compared to vanilla Android, but is not heavily modified compared to devices of some other manufactures.

13 Discussion of test results

In this chapter, we discuss the various results. We make a generalization of the various scenarios and describe the corresponding behaviour of the section *Bond Events* within the module *Bluetooth_manager* of *dumpsys*. We'll describe both master⁶ and slave⁷ devices separately. Each scenario will be clarified with the help of a generalized example.

13.1 Time-out of a request

13.1.1 Master device

```
timestamp – slave address – btif_dm_create_bond – BOND_STATE_NONE
timestamp – slave address – bond_state_changed – BOND_STATE_BONDING
timestamp – slave address – bond_state_changed – BOND_STATE_BONDING
```

+30s

```
timestamp – slave address – bond_state_changed – BOND_STATE_none
```

When encountering this scenario, it means that you are observing an artefact of a master device who sent a request that timed out.

13.1.2 Slave device

```
timestamp – master address – bond_state_changed – BOND_STATE_BONDING
```

+30s

```
timestamp – master address – bond_state_changed – BOND_STATE_NONE
```

When encountering this scenario, it means that you are observing an artefact of a slave device who received a request that timed out.

13.2 Cancellation of a request

13.2.1 Master device

```
timestamp – slave address – btif_dm_create_bond – BOND_STATE_NONE
timestamp – slave address – bond_state_changed – BOND_STATE_BONDING
timestamp – slave address – bond_state_changed – BOND_STATE_BONDING
```

+<30s

```
timestamp – slave address – Invalid value – BOND_STATE_BONDING
timestamp – slave address – bond_state_changed – BOND_STATE_NONE
```

When encountering this scenario, it means that you are observing an artefact of a master device who sent a request and cancelled that request.

⁶ **Master device:** a.k.a. sender, the device issuing the change in bond state, e.g. issuing a bond request.

⁷ **Slave device:** a.k.a. receiver, the device receiving the connection requests. See Figure 1 - Bluetooth Handshake.

13.2.2 Slave device

timestamp – master address – bond_state_changed – BOND_STATE_BONDING
 +<30s
timestamp – master address – bond_state_changed – BOND_STATE_NONE

When encountering this scenario, it means that you are observing an artefact of a slave device who received a request that got cancelled by the master.

13.3 Denial of a request

When a request was denied, we see a different behaviour depending on the version of the Android O.S. that the device is running.

13.3.1 Master device

timestamp – slave address – btif_dm_create_bond – BOND_STATE_NONE
timestamp – slave address – bond_state_changed – BOND_STATE_BONDING
timestamp – slave address – bond_state_changed – BOND_STATE_BONDING
 +30s/+<30s
timestamp – slave address – bond_state_changed – BOND_STATE_NONE

During our testing, we saw strange behaviour on the master when the request was denied. When the slave was running Android 7.1, we saw an identical behaviour to a time-out on the master who was running Android 10. When the slave was running Android 10 and the master was running Android 7.1, we saw that when the slave had rejected the request, a change in bond state to none. This indicates that there is a difference in Bluetooth communication.

13.3.2 Slave device

Older⁸ versions of Android:

timestamp – master address – bond_state_changed – BOND_STATE_BONDING
timestamp – master address – Invalid value – BOND_STATE_BONDING
timestamp – master address – bond_state_changed – BOND_STATE_NONE

Newer⁹ versions of Android:

timestamp – master address – bond_state_changed – BOND_STATE_BONDING
timestamp – master address – Invalid value – BOND_STATE_BONDING
timestamp – master address – bond_state_changed – BOND_STATE_NONE
timestamp – master address – btif_dm_remove_bond – BOND_STATE_NONE
timestamp – master address – bond_state_changed – BOND_STATE_NONE
timestamp – master address – bond_state_changed – BOND_STATE_NONE

On newer versions of Android we see that when the request has been declined, the bond is removed. Afterwards we see twice the change in bond state, indicating that a slave running a newer version of Android will notify the master that the request has been rejected. Expected is that this will influence the behaviour of the master and create a new scenario that the artefact

⁸ In our testing, this older version of Android was Android 7.1 (Nexus 6).

⁹ In our testing, the testing was done on the most recent version of Android, Android 10, at that time.

could reside in when the master is running a newer version of Android. It is not expected that the variation in Bluetooth version would influence this, since a master running an older version of Android knows immediately that the request was denied. Further research to this behaviour is recommended with devices running newer versions of Android, and if possible, with different Bluetooth versions.

If one of these scenarios is encountered, it means that you are observing an artefact of a slave device who received a request that was denied.

13.4 Acceptance of a request

13.4.1 Master device

```
timestamp – slave address – btif_dm_create_bond – BOND_STATE_NONE
timestamp – slave address – bond_state_changed – BOND_STATE_BONDING
timestamp – slave address – bond_state_changed – BOND_STATE_BONDING
timestamp – slave address – bond_state_changed – BOND_STATE_BONDED
(timestamp – slave address – bond_state_changed – BOND_STATE_BONDED)
```

When encountering this scenario, it means that you are observing an artefact of a master device who sent a request that got accepted.

13.4.2 Slave device

```
timestamp – master address – bond_state_changed – BOND_STATE_BONDING
timestamp – master address – bond_state_changed – BOND_STATE_BONDED
(timestamp – master address – bond_state_changed – BOND_STATE_BONDED)
```

When encountering this scenario, it means that you are observing an artefact of a slave device who received a request that was accepted.

13.4.3 Remarks

As we saw a difference between Android versions when a request was rejected, we see a similar difference when a request has been accepted. The only difference we saw is that when the master is running a newer version of Android, the master has twice the entry where the bond state is changed to bonded and on the slave twice a change in bond state to bonded, hence we put the last line between the brackets.

13.5 Forgetting a connection

When forgetting a connection, this is only visible on the device that forgot the connection. This makes sense since it is possible that the other device is out of range.

```
timestamp – address of connected device – btif_dm_remove_bond – BOND_STATE_NONE
timestamp – address of connected device – bond_state_changed – BOND_STATE_NONE
timestamp – address of connected device – bond_state_changed – BOND_STATE_NONE
```

As you can observe in the pseudocode, the change in bond state is present twice. At this moment, we don't have a possible explanation why this is and what the function of it is.

Bluetooth in Digital mobile Forensics

Discussion of test results

When the forgotten device tries to reconnect with the device who forgot the connection, this will fail, and a new connection request will be created (see Test 13: Forget Device and attempt to reconnect). A possible explanation is that the pin code used in the original request is no longer valid for a connection between the two devices.

13.6 File transfer

When a file is transferred between two devices, we see the same behaviour in the artefact of the two devices.

timestamp – address of the other device – bond_state_changed – BOND_STATE_BONDING
timestamp – address of the other device – bond_state_changed – BOND_STATE_NONE

The changes in the bond state happens within a second.

13.7 General remarks

We observed that when a change in bond state is conducted, the master device has twice a change in bond state. For example, when a new connection request is conducted, the master will have twice the entry *timestamp – slave address – bond_state_changed – BOND_STATE_NONE*. When comparing the timestamp of the second entry to the entries on the slave device, the slave will have the entry *timestamp – master address – bond_state_changed – BOND_STATE_BONDING* around that timestamp. We can deduce that the second entry on the master device is when the slave has received the request and its bond state is being changed to be able to bond. There is some variation in the timestamp to be expected due to the fact we are communicating wirelessly and that the devices are not time synced to the millisecond. A variation in timestamp around the one to two second mark is thus expected.

It is suspected that the differences mentioned between the different devices are due to the difference in Android versions and not the difference in Bluetooth version. This is found on the observation that the behaviour on the Nexus 6 changed compared to the tests with the iPhone.

With the test scenarios where we simulated an incorrect pin, we wanted to know if something went wrong during a connection request, what the behaviour of the artefact was. Since we accepted the request on the sender and rejected it on the receiver, we had in essence the same scenario as when the request was denied.

13.8 Value of the artefact

As stated previously, the value of an artefact for DFIR depends heavily on its lifespan. In chapter 11 - Artefact lifespan, we determined its value based on the lifespan of the artefact. We did not determine if the information that the artefact contains is valuable.

To decide if the information of our described Bluetooth artefact has a value during an investigation, we will compare this to a real-life case that has been described in the paper of Panagiotis Andriotis, George Oikonomou and Theo Tryfonas from the Bristol Cryptography Group at the University of Bristol [10].

They described a case where images were classified as child pornography. The images were transferred via Bluetooth from a mobile phone to a computer. Since the question that we set out to answer in this thesis was whether we could find an artefact indicating if an attempt to

Bluetooth in Digital mobile Forensics

Discussion of test results

connect over Bluetooth could be found, this case is applicable to our described Bluetooth artefact. To send an image via Bluetooth, a Bluetooth connection must be conducted (see Test 15: Sending a file from the OnePlus and Test 16: Sending a file from the Nexus 6).

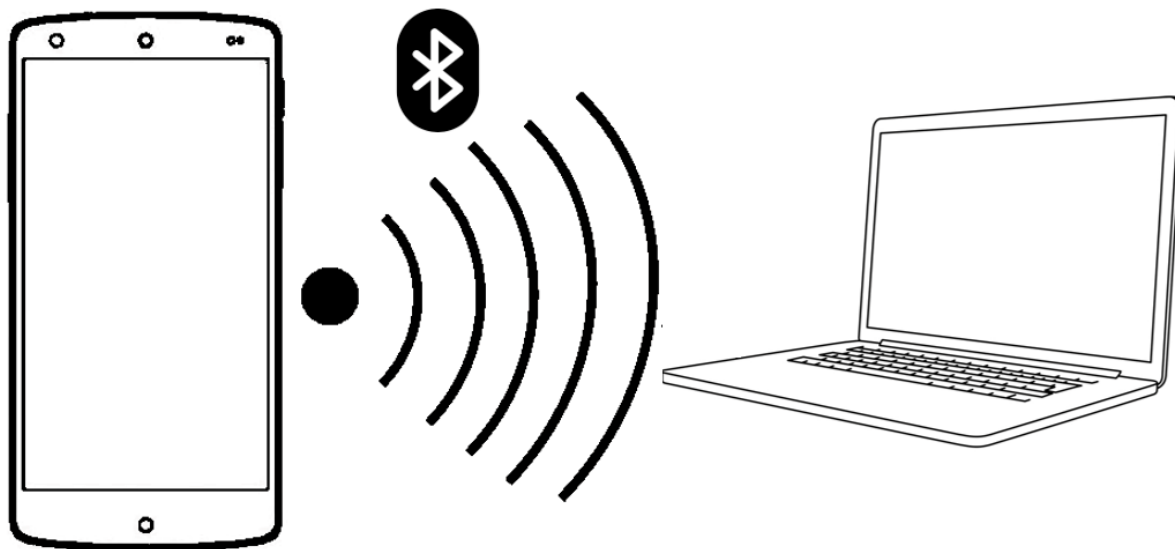


Figure 14 - Visualisation of the transferring process over Bluetooth

As the performed tests showed, we could in fact prove if a file was sent via Bluetooth. The typical properties of the artefact when transferring a file deviates greatly compared to other observed behaviours of the artefact. This means that yes, our found artefact is valuable in a digital forensic investigation.

14 Differences between Android versions

As mentioned in the previous chapter, there are some differences between the two Android versions that were used in our testing.

It can be suspected that the Bluetooth service on Android 10 is more advanced compared to Android 7.1. Furthermore, there are some differences visible in *Bluetooth_manager*. Some subtle differences, for example the difference between “BD-addr” and “address” in the studied artefact where the address of the other device could be seen. Other, larger differences, that we observed were that the OnePlus had all music streaming services present in *Bluetooth_manager*.

This section, where these streaming services are visible, is called “Media Players”, see Figure 15 - Output of the section “Media Players” of the dumpsys module *Bluetooth_manager*. As you can see, even the last played song is present. We’ve observed this artefact and noticed that the asterisk indicates which service was last active. It defaults to Google Play Music. As you can see in the attachment, it has defaulted to the default service, but knows what song was last streamed via Spotify. The visible song was playing at that moment via the Dell XPS (the laptop that was used during testing, see chapter 3 - Test Devices) and the OnePlus knew this. Probably due to Spotify Connect, but has not been verified.

The benefit of this artefact is that when certain Bluetooth speakers are connected to the device, the music resumes automatically where the music was stopped during the last session of the service.

```
Media Players:
  #2: MediaPlayerInfo com.google.android.googlequicksearchbox (as 'Google') Type = 1, S
  *#3: MediaPlayerInfo com.google.android.music (as 'Google Play Muziek') Type = 1, Subt
  #4: MediaPlayerInfo com.google.android.youtube (as 'YouTube') Type = 1, SubType = 0,
  #5: MediaPlayerInfo com.shazam.android (as 'Shazam') Type = 1, SubType = 0, Status =
  #6: MediaPlayerInfo com.spotify.music (as 'Spotify') Type = 1, SubType = 0, Status =

(as 'Google') Type = 1, SubType = 0, Status = 0 Feature Bits [40 41 42 44 45 47 48 58 59 62 65 67 68]
y Muziek') Type = 1, SubType = 0, Status = 0 Feature Bits [40 41 42 44 45 47 48 58 59 62 65 67 68]
) Type = 1, SubType = 0, Status = 0 Feature Bits [40 41 42 44 45 47 48 58 59 62 65 67 68]
1, SubType = 0, Status = 0 Feature Bits [40 41 42 44 45 47 48 58 59 62 65 67 68]
1, SubType = 0, Status = 2 Feature Bits [40 41 42 44 45 47 48 58 59 62 65 67 68]

Feature Bits [40 41 42 44 45 47 48 58 59 62 65 67 68] Controller: null
ature Bits [40 41 42 44 45 47 48 58 59 62 65 67 68] Controller: MediaController (com.google
; [40 41 42 44 45 47 48 58 59 62 65 67 68] Controller: null
12 44 45 47 48 58 59 62 65 67 68] Controller: null
12 44 45 47 48 58 59 62 65 67 68] Controller: MediaController (com.spotify.music@6b798cf)

ller: null
: MediaController (com.google.android.music@bc3f52e) null

(com.spotify.music@6b798cf) We Will Rock You - Remastered, Queen, News Of The World (Deluxe Remastered Version)
```

Figure 15 - Output of the section “Media Players” of the dumpsys module *Bluetooth_manager*

Since this output is very long (wide), separate screenshots were made and placed under each other. The output for each “Media Player” is one line.

15 Conclusion

15.1 What did we discover within this dissertation?

Mobile digital forensics is a fast evolving domain within an already fast evolving world. For digital forensic investigators it is cat-and-mouse game to keep up with the changes in the world of mobile devices. The flavours that each manufacturer implements on the fast evolving base is not helping with this. It's a hunt for the forensic researchers to find a tool that is able to assist them with as many as possible devices in different scenarios.

The discovery of the potential that *dumpsys* has is in that aspect unseen. *Dumpsys* has over a hundred different modules, each specialized in a certain aspect of an Android device. A very important one is the module *account* where we are able to see all accounts and the corresponding service of that account that are known on the device.

In our case study we observed one section of such module and were able to determine what happens in different scenarios where Bluetooth is used. Since that module, *Bluetooth_manager*, also contains what known device correspondents to what Bluetooth address, we can easily and quickly identify connected devices.

We started from a live investigation, but found a way quickly that we were able to examine the information via a copy of the data, a.k.a. post-mortem.

15.2 What is the impact on the current state of Mobile Digital Forensics?

Currently, the first step that is undertaken when a mobile device is confiscated is to make sure it is unable to communicate to the outside. This means that all networking options of the device are turned off. This is done by either enabling aeroplane mode or switching the device off. The intent of these actions are to prevent a remote wipe of the device and destroy potential proof.

Unfortunately, valuable information is lost by doing so as e.g. the entries in *Bluetooth_manager* are reset as a consequence. This means that the current procedures during a confiscation are destroying data. From this point of view, there are issues and questions that can be raised. When a forensic investigation is conducted, the goal is to work as forensically sound as possible. This mean that altering the subject is reduced to a minimum and that data is preserved. Destruction of data needs to be prevented, no matter what. This is where the current state of digital forensics, to be more specific, the seizure procedure falls short.

A new seizure procedure need to be put in place. The solution is very simple. When seizing a device, a copy of the information from *dumpsys* can be made, preferably from each module separately. When this has been successfully done, the device can preferably be placed in a Faraday enclosure. Only when this is not possible, it could be turned off or enable aeroplane mode, as done currently. It goes without saying that preferably the copy is made whilst the device is already in a Faraday cage. By doing so, the possibility that the device is wiped during the copy process is eliminated.

An example script to extract all information out of the *dumpsys* services can be found in Attachment 9: Example script to extract all information from the *dumpsys* modules. In this attachment we also describe what a possible improved seizure procedure could be.

16 Bibliography

- [1] Statcounter, "Mobile Operating System Market Share Februari 2020," February 2020. [Online]. Available: <https://gs.statcounter.com/os-market-share/mobile/worldwide>.
- [2] Statista, "Mobile Operating Systems' Market Share Worldwide," January 2020. [Online]. Available: <https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/>.
- [3] hostingtribunal.com, "Mobile and Desktop Operating System Market Share," 2020. [Online]. Available: <https://hostingtribunal.com/blog/operating-systems-market-share/#gref>.
- [4] J. Padgette, R. Smithbey, J. Bahr, L. Chen, M. Batra, K. Scarfone and M. Holtmann, "Guide to Bluetooth," NIST, Gaithersburg, 2017.
- [5] Lynxbee, "Understanding Bluetooth Basics – Pairing and Handshaking process," 2020. [Online]. Available: <https://www.lynxbee.com/understanding-bluetooth-basics-pairing-and-handshaking-process/>.
- [6] M. McLaughlin, "How to Root Your Android Phone," Lifewire, 16 March 2020. [Online]. Available: <https://www.lifewire.com/how-to-root-your-android-phone-121676>.
- [7] Android Developers, "Android Debug Bridge (ADB)," 9 March 2020. [Online]. Available: <https://developer.android.com/studio/command-line/adb>.
- [8] Tsurugi, "About us," March 2020. [Online]. Available: https://tsurugi-linux.org/about_us.php.
- [9] Tsurugi, "Your DFIR Linux Distribution," 9 March 2020. [Online]. Available: <https://tsurugi-linux.org/>.
- [10] P. Andriotis, G. Oikonomou and T. Tryfonas, "Forensic Analysis of Wireless Networking Evidence of Android Smartphones," University of Bristol, Bristol, 2012.
- [11] Android Developers, "Logcat command-line tool," 9 March 2020. [Online]. Available: <https://developer.android.com/studio/command-line/logcat>.
- [12] Android Developers, "dumpsys," 9 March 2020. [Online]. Available: <https://developer.android.com/studio/command-line/dumpsys>.
- [13] Linux Freedom, "Tsurugi Linux," 9 March 2020. [Online]. Available: <http://linuxfreedom.com/tsurugi/>.
- [14] M. Rutnik, "What is stock Android?," 20 January 2019. [Online]. Available: <https://www.androidauthority.com/what-is-stock-android-845627/>.
- [15] D. Tobok, "What Is Digital Forensics?," 15 Februari 2018. [Online]. Available: <https://cytelligence.com/resource/what-is-digital-forensics/>.
- [16] I. Spais, "Introduction to Digital Forensics," 18 December 2019. [Online]. Available: <https://www.cipsec.eu/content/introduction-digital-forensics>.

Bluetooth in Digital mobile Forensics

Bibliography

- [17] M. Sonntag, *Introduction to Computer Forensics*, Linz: Johannes Kepler University, 2012.
- [18] P. Marjie T. Britz, *Computer Forensics and Cyber Crime - An Introduction*, Clemson: Pearson, 2013.
- [19] J. Lyle, *Digital Forensics or Your trail is easier to follow than you think*, NIST, 2015.
- [20] S. L. Ksander, *Introduction to Digital Forensics*, West Lafayette, Indiana: Purdue University, 2020.
- [21] K. Kent, S. Chevalier, T. Grance and H. Dang, "Guide to Integrating Forensic Techniques into Incident Response," NIST, Gaithersburg, 2006.
- [22] P. Greg Gogolin, "Digital Forensics Explained," Taylor & Francis Group, Boca Raton, 2013.
- [23] J. Fruhlinger, "What is digital forensics? And how to land a job in this hot filed," 25 January 2019. [Online]. Available: <https://www.csoonline.com/article/3334396/what-is-digital-forensics-and-how-to-land-a-job-in-this-hot-field.html>.
- [24] M. Al-Hadadi and A. Al Shidhani, "Smartphone Forensics Analysis: A Case Study," *International Journal of Computer and Electrical Engineering*, vol. Vol. 5, no. No. 6, pp. 576-580, December 2013.
- [25] Guru99, "What is Digital Forensics? History, Process, Types, Challenges," 19 April 2020. [Online]. Available: <https://www.guru99.com/digital-forensics.html>.
- [26] XDA Developers, "What is ADB? How to Install ADB, Common Uses, and Advanced Tutorials," 2020. [Online]. Available: <https://www.xda-developers.com/what-is-adb/>.
- [27] Information Security and Forensics Society (ISFS), "Computer Forensics - Part 1: An Introduction to Computer Forensics," april 2004. [Online]. Available: http://www.isfs.org.hk/publications/ComputerForensics_part1.pdf.
- [28] P. Coeck, Interviewee, *Digital Forensic Researcher*. [Interview]. March 2020.
- [29] Bluetooth SIG, "Bluetooth," 2020. [Online]. Available: [bluetooth.com](https://www.bluetooth.com).
- [30] Bluetooth SIG, "Bluetooth Core Specification v5.0," Bluetooth SIG, Kirkland, Wachington, 2016.
- [31] Bluetooth SIG, "Our history," 2020. [Online]. Available: <https://www.bluetooth.com/about-us/our-history/>.
- [32] IEEE, "802.15.1-2002 - IEEE Standard for Telecommunications and Information Exchange Between Systems - LAN/MAN - Specific Requirements - Part 15: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks ...," 14 June 2002. [Online]. Available: <https://ieeexplore.ieee.org/document/1016473>.
- [33] IEEE, "802.15.1-2005 - IEEE Standard for Information technology-- Local and metropolitan area networks-- Specific requirements-- Part 15.1a: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Wireless Personal Area Networks (WPAN)," 14 June 2005. [Online]. Available: <https://ieeexplore.ieee.org/document/1490827>.

Bluetooth in Digital mobile Forensics

Bibliography

- [34] M. Woolley, "Bluetooth Core Specification Version 5.2," Bluetooth SIG, Kirkland, Wachingoton, 2020.
- [35] S. Vafaei and M. Henney, "Bluetooth Versions Comparison & Profiles," RTINGS.com, 6 July 2017. [Online]. Available: <https://www.rtings.com/headphones/learn/bluetooth-versions-comparison-profiles>.
- [36] David, "What's the Difference between Bluetooth Versions 2.x, 3.x, 4.x and 5.x," The Droid Guy, 9 April 2020. [Online]. Available: <https://thedroidguy.com/whats-difference-between-bluetooth-versions-2-x-3-x-4-x-5-x-1065792>.
- [37] J. Flynt, "What's the Difference between Bluetooth Versions 2, 3, 4 and 5?," 3DInsider, 12 January 2019. [Online]. Available: <https://3dinsider.com/bluetooth-versions/>.
- [38] I. A. Nguyen, "Bluetooth 1.0 vs 2.0 vs 3.0 vs 4.0 vs 5.0 - How They Compare," 18 April 2018. [Online]. Available: <https://www.semiconductorstore.com/blog/2018/Bluetooth-1-0-vs-2-0-vs-3-0-vs-4-0-vs-5-0-How-They-Differ-Symmetry-Blog/3147/>.
- [39] SSH.com, "Root User," SSH.com, 2020. [Online]. Available: <https://www.ssh.com/iam/user/root/>.
- [40] Urban Dictionary, "Urban Dictionary: Bricked," 2020. [Online]. Available: <https://www.urbandictionary.com/define.php?term=bricked>.

Overview of attachments

- 1 List of all services that can be issued with *dumpsys* on Nexus 6
- 2 Output of the command “*adb logcat | grep – i bluetooth*”
- 3 Generalisation of the applied test procedure
- 4 Pop-up when sending a request on Android
- 5 Pop-up when receiving a request on Android
- 6 Pop-up on Nexus 6 when pin is incorrect
- 7 Forgetting a paired Bluetooth device on Android
- 8 Photo used in testing to send between devices
- 9 Example script to extract all information from the *dumpsys* modules

Attachment 1: List of all services that can be issued with "dumpsys" on Nexus 6

```
nick@XPS-Tsurugi:~$ adb shell dumpsys -l
Currently running services:
  AtCmdFwd
  DockObserver
  SurfaceFlinger
  accessibility
  account
  activity
  alarm
  android.security.keystore
  appops
  appwidget
  assetatlas
  audio
  backup
  battery
  batteryproperties
  batterystats
  bluetooth_manager
  carrier_config
  clipboard
  cneservice
  commontime_management
  connectivity
  connectivity_metrics_logger
  connmetrics
  consumer_ir
  content
  contexthub_service
  country_detector
  cpuinfo
  dbinfo
  device_policy
  deviceidle
  devicestoragemonitor
  diskstats
  display
  display.qservice
  dns_listener
  dreams
  drm.drmManager
  dropbox
  ethernet
  gfxinfo
  gpu
  graphicsstats
  hardware_properties
  imms
  ims
  input
  input_method
  iphonesubinfo
  isms
  isub
  jobscheduler
```

Bluetooth in Digital mobile Forensics

Attachments

```
isub
jobscheduler
launcherapps
location
lock_settings
media.audio_flinger
media.audio_policy
media.camera
media.camera.proxy
media.codec
media.drm
media.extractor
media.player
media.radio
media.resource_manager
media.sound_trigger_hw
media_projection
media_router
media_session
meminfo
midi
mount
netpolicy
netstats
network_management
network_score
network_time_update_service
nfc
notification
otadexopt
package
permission
persistent_data_block
phone
pinner
power
print
processinfo
procstats
recovery
restrictions
rttmanager
samplingprofiler
scheduling_policy
search
sensorservice
serial
servicediscovery
shortcut
simphonebook
sip
soundtrigger
statusbar
telecom
telephony.registry
textservices
```

Bluetooth in Digital mobile Forensics

Attachments

```
textservices
trust
uimode
updatelock
usagestats
usb
user
vibrator
voiceinteraction
vrmanager
wallpaper
webviewupdate
wifi
wifip2p
wifiscanner
window
nick@XPS-Tsurugi:~$ █
```

Bluetooth in Digital mobile Forensics

Attachments

Attachment 2: output of the command `adb logcat | grep -i bluetooth`

```

03-28 00:57:22.985 1567 D BluetoothAdapter: STATE_ON
03-28 00:57:22.988 1567 D BluetoothLeScanner: onClientRegistered() - status=0 clientIf=6 mClientIf=0
03-28 00:57:25.010 1567 D BluetoothAdapter: STATE_ON
03-28 00:57:25.981 9953 V BluetoothDiscoveryReceiver: Received: android.bluetooth.adapter.action.DISCOVERY_STARTED
03-28 00:57:25.992 9953 V BluetoothDiscoveryReceiver: Received: android.bluetooth.adapter.action.DISCOVERY_FINISHED
03-28 00:58:20.209 1409 I BluetoothRemoteDevices: state12newState0
03-28 00:58:20.238 1567 I TrustAgent.Tracker: [BluetoothConnectionTracker] Bluetooth connect broadcast for OnePlus 7T Pro 4
8:01:C5:86:27:CE
03-28 00:58:21.025 1409 I BluetoothBondStateMachine: bondStateChangeCallback: Status: 0 Address: 48:01:C5:86:27:CE newState
e: 1
03-28 00:58:21.027 1409 I BluetoothBondStateMachine: sspRequestCallback: [B@337e35b name: [B@973b7f8 cod: 5898764 pairingV
ariant 0 passkey: 447915
03-28 00:58:21.035 9953 W BluetoothEventManager: CachedBluetoothDevice for device 48:01:C5:86:27:CE not found, calling rea
dPairedDevices().
03-28 00:58:21.037 9953 E BluetoothEventManager: Got bonding state changed for 48:01:C5:86:27:CE, but we have no record of
that device.
03-28 00:58:21.038 1409 I BluetoothBondStateMachine: Bond State Change Intent:48:01:C5:86:27:CE OldState: 10 NewState: 11
03-28 00:58:21.038 1409 I BluetoothBondStateMachine: Entering PendingCommandState State
03-28 00:58:21.157 9953 W ContextImpl: Calling a method in the system process without a qualified user: android.app.Contex
tImpl.startActivity:767 android.content.ContextWrapper.startActivity:356 android.content.ContextWrapper.startActivity:356 com.and
roid.settings.bluetooth.BluetoothPairingRequest.onReceive:74 android.app.ActivityThread.handleReceiver:3040
03-28 00:58:21.157 9953 W ContextImpl: Calling a method in the system process without a qualified user: android.app.Contex
tImpl.startActivity:779 android.app.ContextImpl.startActivity:768 android.content.ContextWrapper.startActivity:356 android.conten
t.ContextWrapper.startActivity:356 com.android.settings.bluetooth.BluetoothPairingRequest.onReceive:74
03-28 00:58:21.182 864 I ActivityManager: START u0 {act=android.bluetooth.device.action.PAIRING_REQUEST flg=0x100000000 cm
p=com.android.settings/.bluetooth.BluetoothPairingDialog (has extras)} from uid 1000 on display 0
03-28 00:58:21.192 9953 D LocalBluetoothManager: setting foreground activity to null
03-28 00:58:21.198 1409 I BluetoothAdapterProperties: Discoverable Timeout:120
03-28 00:58:21.199 1409 I BluetoothAdapterProperties: Scan Mode:21
03-28 00:58:21.428 864 I ActivityManager: Displayed com.android.settings/.bluetooth.BluetoothPairingDialog: +228ms
03-28 00:58:51.305 1409 I BluetoothBondStateMachine: bondStateChangeCallback: Status: 9 Address: 48:01:C5:86:27:CE newState
e: 0
03-28 00:58:51.306 1409 D BluetoothAdapterProperties: Failed to remove device: 48:01:C5:86:27:CE
03-28 00:58:51.307 1409 E BluetoothRemoteDevices: state12newState1
03-28 00:58:51.307 1409 I BluetoothBondStateMachine: Bond State Change Intent:48:01:C5:86:27:CE OldState: 11 NewState: 10
03-28 00:58:51.324 1409 D HidService: getHidService(): returning com.android.bluetooth.hid.HidService@444ad74
03-28 00:58:51.348 1409 I BluetoothBondStateMachine: StableState(): Entering Off State
03-28 00:59:45.043 9953 I BluetoothPairingDialog: Pairing dialog canceled
03-28 00:59:45.186 9953 D LocalBluetoothManager: setting foreground activity to non-null context
03-28 00:59:45.236 1409 I BluetoothAdapterProperties: Discoverable Timeout:120
03-28 00:59:45.236 1409 I BluetoothAdapterProperties: Scan Mode:23
03-28 01:00:23.886 1567 D BluetoothAdapter: STATE_ON
03-28 01:01:20.497 9953 D LocalBluetoothManager: setting foreground activity to null
03-28 01:01:20.505 1409 I BluetoothAdapterProperties: Discoverable Timeout:120
03-28 01:01:20.507 1409 I BluetoothAdapterProperties: Scan Mode:21
03-28 01:01:56.123 864 I BluetoothManagerService: Message: 20
03-28 01:01:56.123 864 D BluetoothManagerService: Added callback: android.bluetooth.IBluetoothManagerCallbackStub$Proxy@
cc07c7f:true
03-28 01:01:56.125 864 D BluetoothManagerService: Message: 30
03-28 01:02:45.373 1938 D BluetoothA2dp: Proxy object connected
03-28 01:03:05.331 864 D BluetoothManagerService: Message: 31

```

Attachment 3: Generalization of the applied test procedure

Generalization of a
Test Procedure
when using
Android Debug
Bridge's Dumpsys

adb dumpsys

Introduction

This document describes an generalization of a possible test procedure for the different modules within *adb dumpsys*. This is specific to Android devices.

The described procedure was used in my bachelor dissertation where we covered one specific section of the module *Bluetooth_manager* of *dumpsys*.

We aimed to create a general procedure that can be used as a template for further investigation of the different modules of *dumpsys* and the info that can be found in these modules. An explanation about *adb* and *dumpsys* can be found in the aforementioned thesis.

Preparations

The first, and most important, decision that has to be made that will determine if the results are successful and usable or not is whether to start from a clean, factory reset device or not. In some cases, the data that has been gather during the use of device, is the data that you want to research. If a device is clean, this data is not yet present. Researching this data on a clean device will be unsuccessful.

When this decision has been made, the developer options need to be enabled on the test subject. USB-debugging has to be activated. A detailed description can be found in chapter 4 of my bachelor dissertation.

To connect with the test subject, the used computer needs to be able to run Android Debug Bridge and have this installed.

Once these conditions are fulfilled, you can determine what module(s) and section(s) of the module(s) will be monitored. This depends on your testcase.

Testing

When conducting these test, you first need to have a baseline. Before each testcase it is recommended to determine the state of each section that is monitored. When this has been performed, you can compare the state of the sections that you are monitoring during and after each one of the tests.

It's recommended to push the output from the *adb* command to a text file. This can easily be done as shown in the following example. Android and *adb* are Unix based. Most of the Unix commands work via *adb*. When this has been done, you can easily go back to review the results and discover potential changes in other sections that are applicable to the testcase that you originally did not monitor.

```
adb shell dumpsys [module] > file.txt
```

Naturally, you can specify the path where the file needs to be created if *adb* is not ran from the directory where you wish to save the file.

It's recommended to perform each test multiple times to make sure that the result is consistent. For example, you can decide that if a certain behaviour occurs for three consecutive tests, it will always occur.

During testing you also have to research the lifespan of the artefact that is being researched. Do certain processes influence the artefact? What happens to the artefact when the device is rebooted? What happens to the artefact when a certain process is stopped?

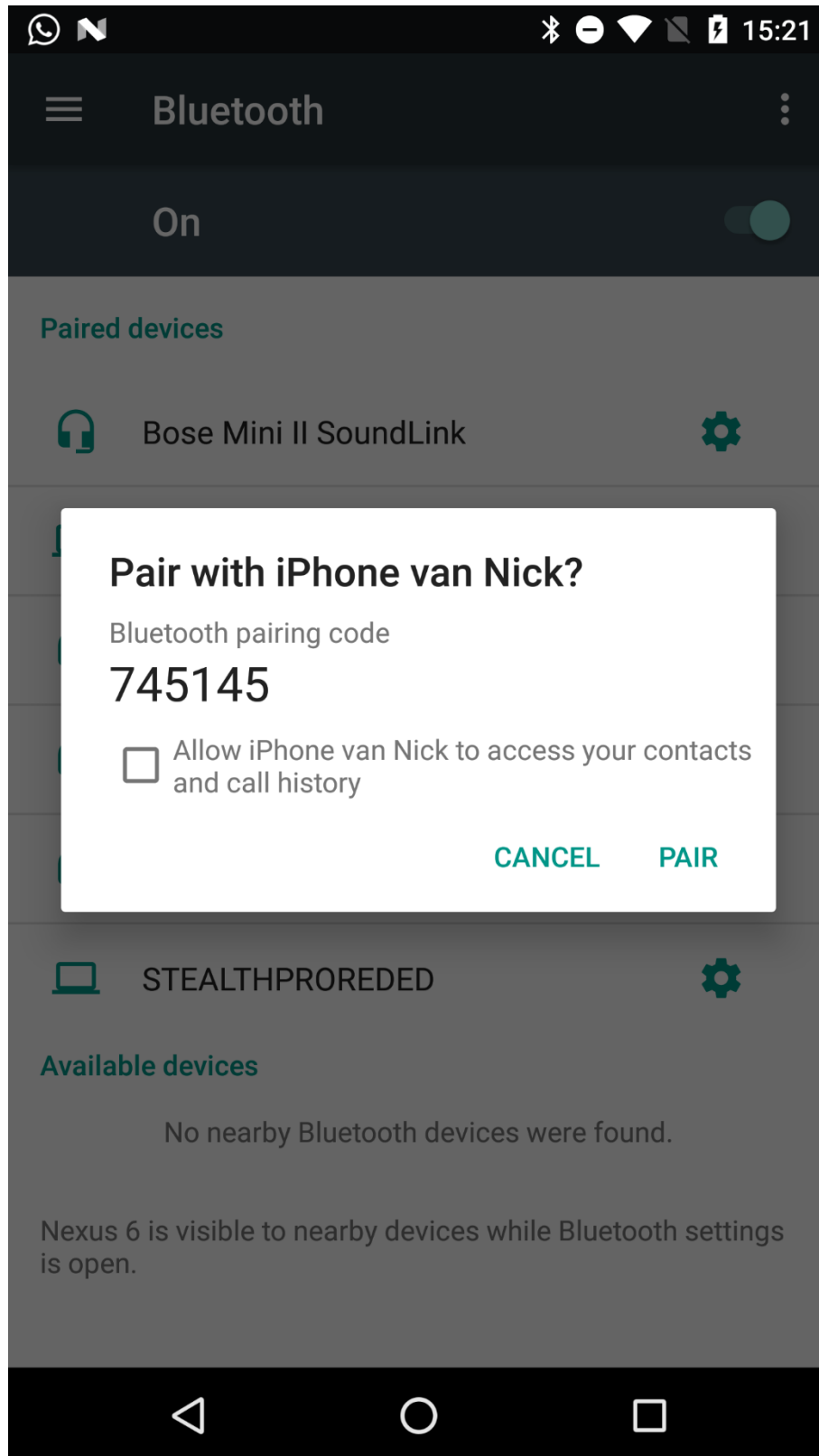
Reporting

After testing, the findings need to be reported. This can be done as you desire. A good rule of thumb is to create for each test individually an overview what has happened. This makes it clear for whoever wants to interpret your results or wants to know what happened in each testcase.

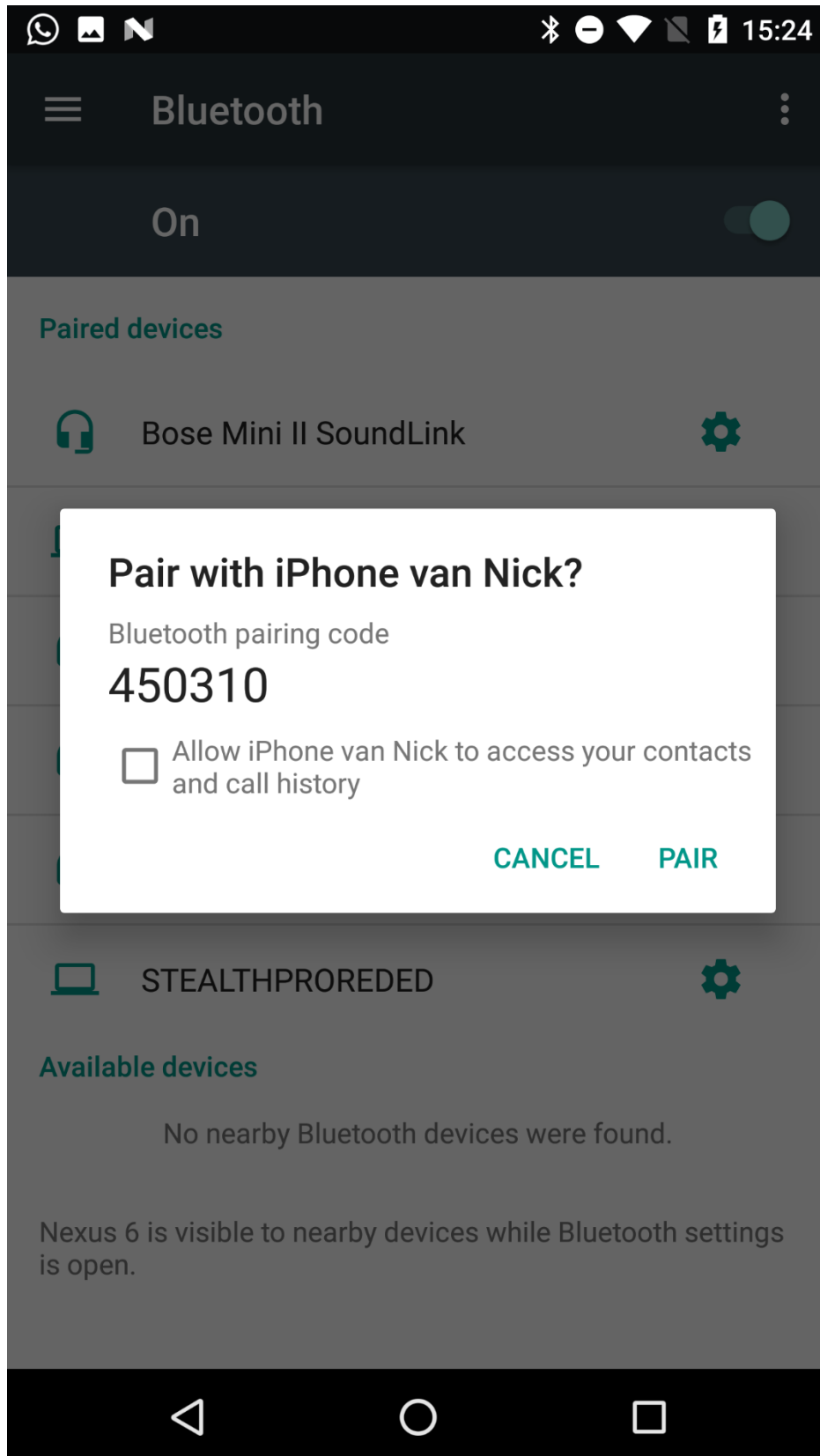
A possible overview could look like this:

Test Case	Description of the testcase.
Devices	<i>Identifiable names of the used devices during the test. If necessary with configuration details.</i>
Procedure	<i>Synopsis of the steps that were performed during the test.</i>
Monitored devices	<i>Identifiable name of the monitored device(s).</i>
How is it monitored?	<i>\$ adb shell dumpsys <module></i>
What is monitored?	<i>Section of the module that is monitored.</i>
Start scenario	<i>Screenshot of the monitored section(s) before testing.</i>
What is expected?	<i>Hypothesis of the expected outcome after testing. Clarified with pseudocode if possible.</i>
During the process	<i>Screenshot of the monitored section(s) during testing when applicable.</i>
After the process	<i>Screenshot of the monitored section(s) after testing.</i>
What is visible after the process?	<i>Detailed description of what is visible on the screenshots.</i>
Is it as expected?	<i>Yes/No. Motivation why it is or is not as expected.</i>

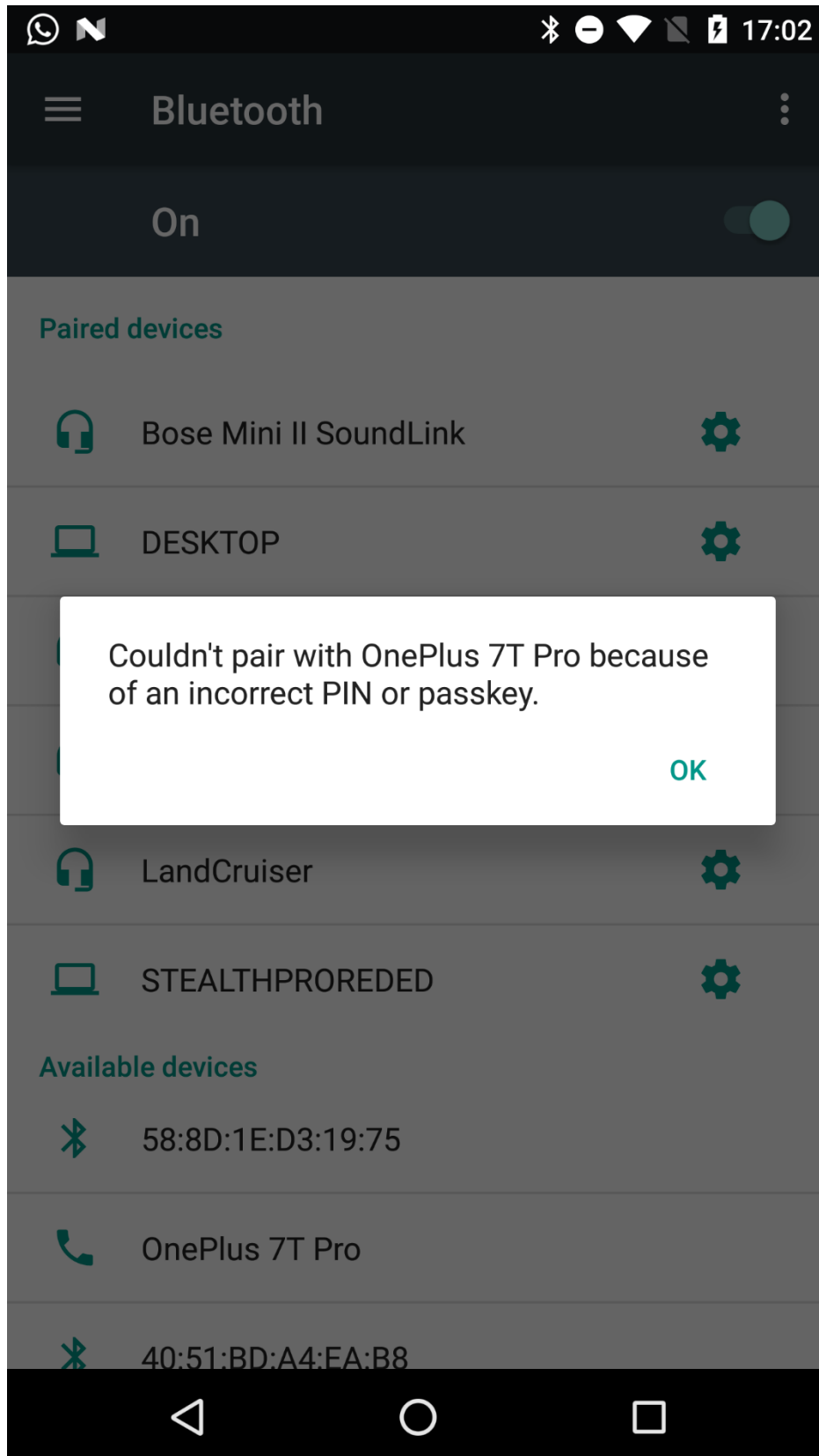
Attachment 4: Pop-up when sending a request on Android



Attachment 5: Pop-up when receiving a request on Android

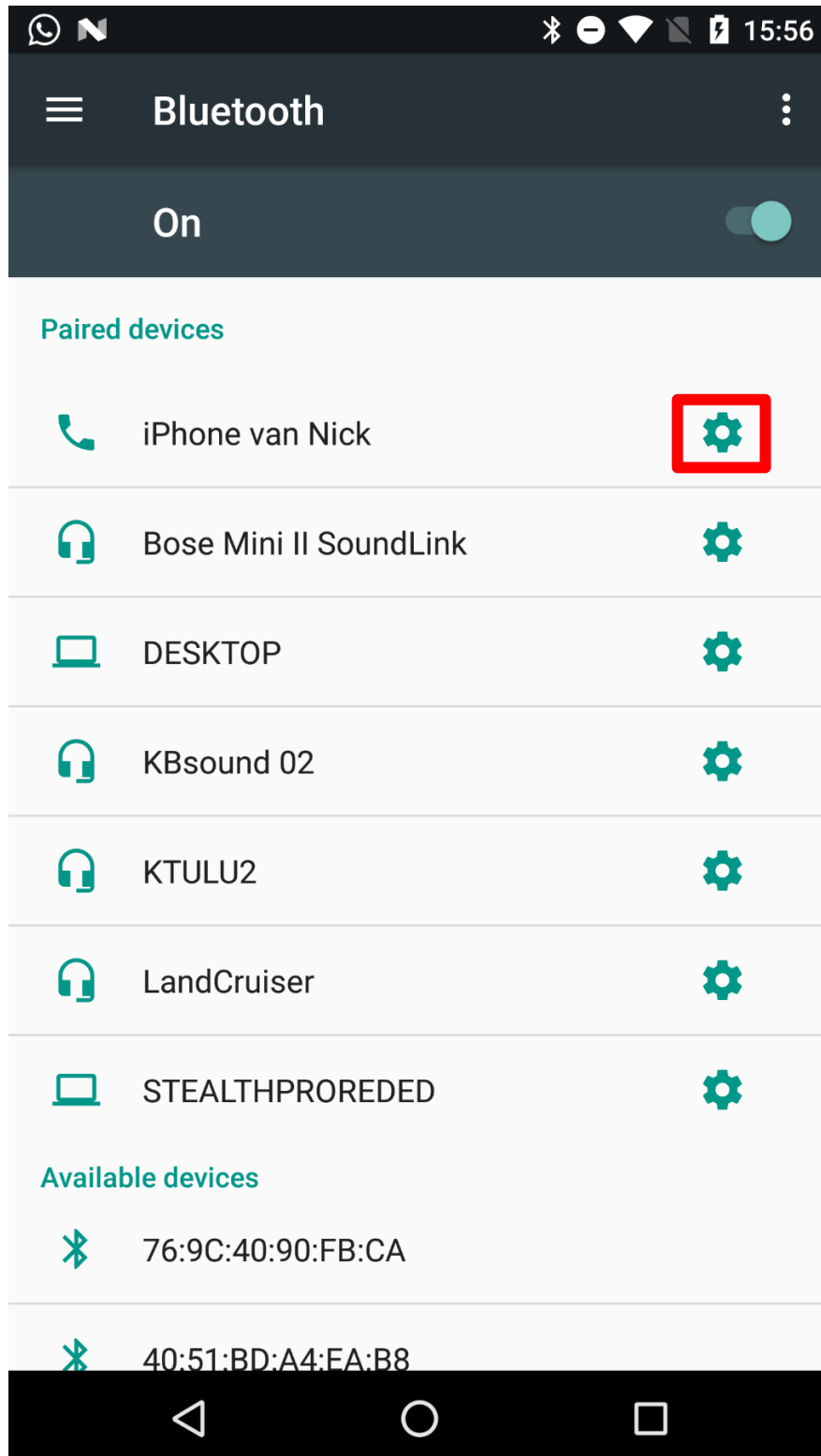


Attachment 6: Pop-up on Nexus 6 when pin is incorrect

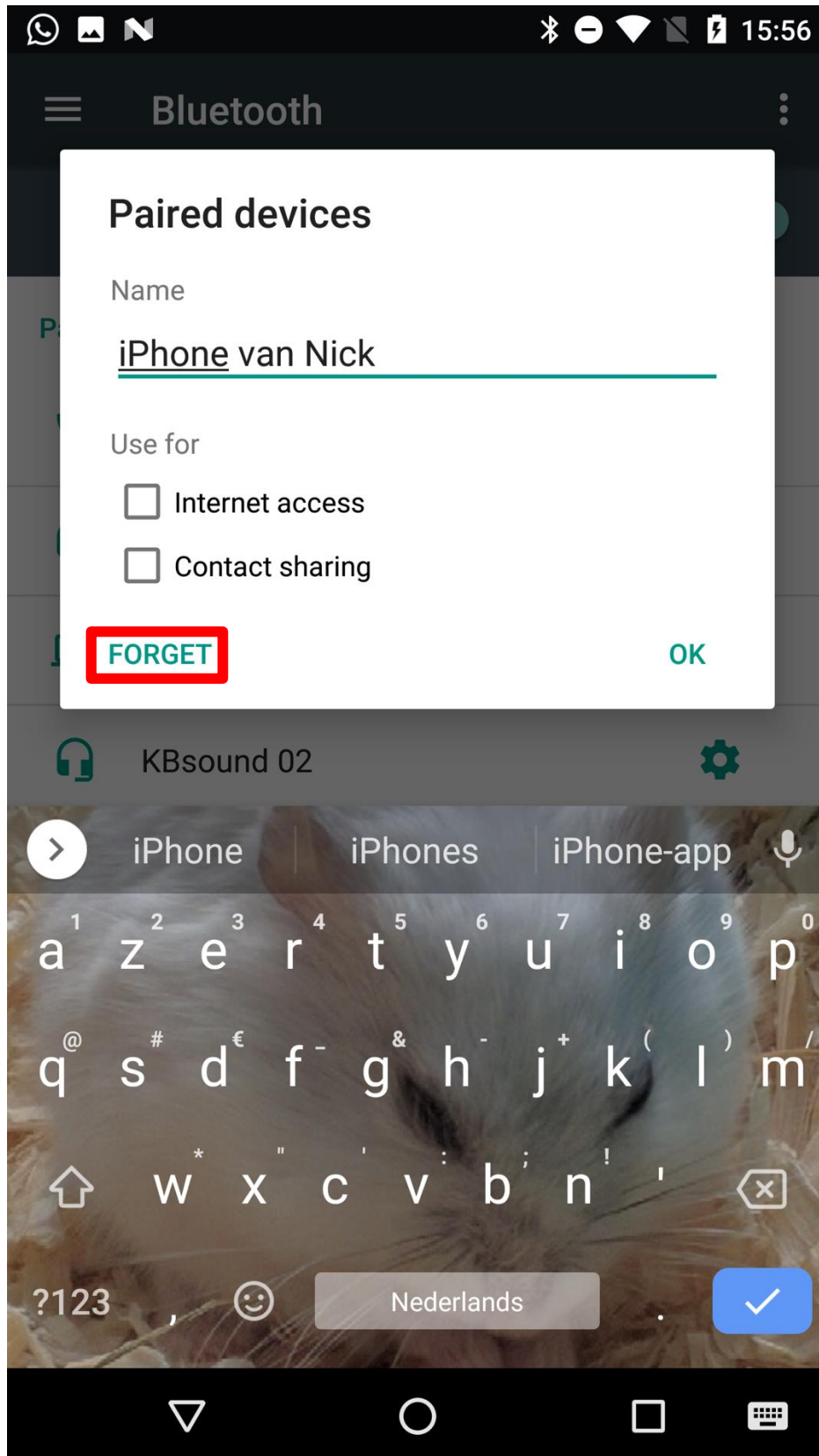


Attachment 7: Forgetting a paired device on Android

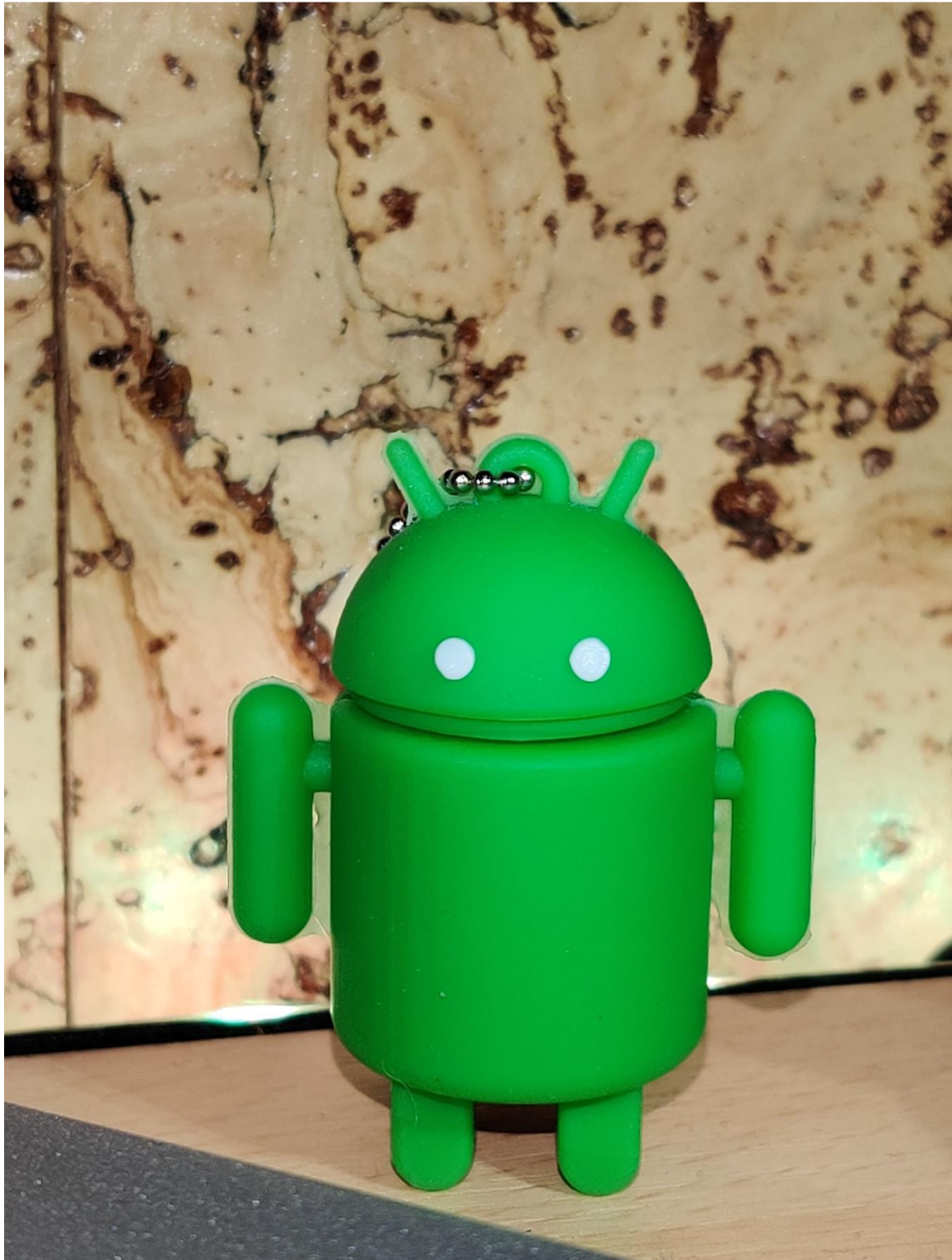
Step 1: In the Bluetooth settings, press the gear icon next to the device you want to go into the advanced settings.



Step 2: To forget the selected device, press "FORGET".



Attachment 8: Photo used in testing to send between devices



This photo was shot with the OnePlus 7T Pro. When sending the file from the Nexus 6 to the OnePlus, the received photo from the previous test was used.

Bluetooth in Digital mobile Forensics Attachments

Attachment 9: Example script to extract all information from the *dumpsys* modules

```

dumpsys.sh x
1 #!/bin/bash
2
3 Path="/home/pi/Documents/dumpsys"
4 DeviceList="/home/pi/Documents/dumpsys/device.txt"
5 DeviceListError="/home/pi/Documents/dumpsys/deviceError.txt"
6 NoDeviceBaseFile="/home/pi/Documents/DoNotTouch_ResourceFile4DumpsysScript2CheckIfDeviceIsConnected.txt"
7 BadStartupBaseFile="/home/pi/Documents/DoNotTouch_ResourceFile4DumpsysScript2CheckIfActive.txt"
8 DeviceName=""
9 services=()
10 log="/home/pi/Documents/dumpsys/log.txt"
11
12 #starting adb and check for connected devices
13 adb devices -l > /home/pi/Documents/dumpsys/device.txt 2> /home/pi/Documents/dumpsys/deviceError.txt
14 echo -e "[ `date` ] Started adb and checked for devices." >> "$log"
15 echo "Started adb @ `date`."
16
17 #Check if device is connected
18 if cmp -s $DeviceList $NoDeviceBaseFile || cmp -s $DeviceListError $BadStartupBaseFile; then
19     printf "No device is connected! \n"
20     sleep 5
21     bash /home/pi/Documents/dumpsys.sh
22     elif grep -wq "offline" "$DeviceList"; then
23         printf "Device is offline! Check if device is powered on and set USB settings/preferences to \File transfer"! \nThis can be found in the notifications tray.\n"
24         echo -e "[ `date` ] Device detected but is offline." >> "$log"
25         sleep 5
26     bash /home/pi/Documents/dumpsys.sh
27 #Check if device has USB-debugging enabled
28 elif grep -wq "unauthorized" "$DeviceList"; then
29     printf "Check RSA keys. Device did not approve connection for USB-debugging! \n"
30     echo -e "[ `date` ] Device detected but RSA keys do not match." >> "$log"
31     sleep 5
32     bash /home/pi/Documents/dumpsys.sh
33 else
34     echo "Retrieving device."
35     DeviceName=$(grep -oP '(?=<product:).*?(?=<model:)' $DeviceList)
36     #DeviceName=$(sed 's/.product:(.*)\s*/\1/' $DeviceList)
37     DeviceName=""
38     DeviceName+=$(grep -oP '(?=<model:).*?(?=<rsas:)' $DeviceList)
39     echo -e "[ `date` ] Device detected! Device is \"$DeviceName\"." >> "$log"
40     DeviceName+="@"
41     DeviceName+="'date'"
42
43 #creating dir for this device to save all information
44 mkdir "$Path"/"$DeviceName"
45 echo "Folder to store information has been created."
46 echo -e "[ `date` ] Directory \"$Path/$DeviceName\" to store data has been created." >> "$log"
47
48 #dump all dumpsys info in one large file and potential errors to another file
49 echo "General dump is being made." >> "$log"
50 echo -e "[ `date` ] General dump is being made."
51 adb shell dumpsys > "$Path"/"$DeviceName"/AAA_AllDumpsys.txt 2> "$Path"/"$DeviceName"/AAA_AllDumpsysErrorLog.txt #AAA_ is used to list the file first in the dir.
52 if grep -wq "unauthorized" "$Path"/"$DeviceName"/AAA_AllDumpsysErrorLog.txt; then
53     echo -e "[ `date` ] Error during the creation of the general dump. New Directory will be created." >> "$log"
54     bash /home/pi/Documents/dumpsys.sh
55 fi
56 echo "General dump has been made."
57 echo -e "[ `date` ] General dump has been created." >> "$log"

```

Bluetooth in Digital mobile Forensics

Attachments

```

58 #List of all dumpsys services and creation of array. Array will be used to dump the info of each service separately. First, whitespace is removed.
59 dumpsysServices=$(adb shell dumpsys -l
60 adb shell dumpsys -l >> "$Path"/"$DeviceName"/AAA_ListOfServices.txt
61 adb shell dumpsys -l >> "$Path"/"$DeviceName"/AAA_ListOfServices.txt
62 entries=$(eval "$dumpsysServices")
63 entriesWithoutLeadingWhitespaces=$(echo -e "${entries}" | tr -d '\s')
64 IFS=$'\n' read -r -d '' -a services <<< "$entriesWithoutLeadingWhitespaces"
65 echo "Services have been determined."
66 echo -e "[ `date` ] All services of dumpsys have been determined." >> "$log"
67
68 #Since we don't need the first entry in the array (the title text "Currently running services:"), we'll remove it for convenience.
69 services=("${services[@]:1}")
70
71 #Iterate over every service in the array and saving the dump of the service in a file.
72 for i in "${services[@]}";
73 do
74     adb shell dumpsys "$i" > "$Path"/"$DeviceName"/"$i".txt 2> "$Path"/"$DeviceName"/"$i"_error.txt
75     echo "$i has been dumped @ `date` ."
76     echo -e "[ `date` ] $i has been dumped." >> "$log"
77 done
78
79 echo "All dumpsys modules have been safed. They can be found at $Path/$DeviceName."
80 echo -e "[ `date` ] All dumpsys modules have been safed. They can be found at \"$Path/$DeviceName\"." >> "$log"
81
82 adb kill-server
83 echo "Stopped adb. You can now safely poweroff this device. Device will poweroff automatically."
84 shutdown
85 #optional, indication when a pi is used without interaction to know when the process is completed.
86
87 fi
88 exit 0
89

```

Side note:

This script was intended to run at start-up on a Raspberry Pi.

There are probably optimisations or cleaner code possible. The line *bash /home/pi/Documents/dumpsys.sh* & was added to the file */etc/rc.local* in order to start the script when the Raspberry Pi is booted.

Proposal for new seizure procedure

A potential seizure procedure is proposed. The context for this proposal is when a search in a suspects house is conducted. The detectives find an Android device belonging to the suspect. In order to keep the example simple, the suspect is in his house and willing to cooperate.

- *What is needed?*
 - Operational Raspberry Pi with the latest version of Rasbian installed
 - Power source for the Raspberry Pi
 - USB-C and micro-USB cables to connect Android device to the Raspberry Pi
 - Example script is present on the device and configured as mentioned in the side notes.

- *What needs to be done?*

The detectives invade the house and arrest the suspect. They find an Android phone in the pocket of the suspect. The suspect is willing to cooperate and gives the code to unlock the device. The detectives write it down¹⁰. Now, USB-debugging can be enabled as explained in chapter 4: Test Setup.

The Raspberry can now be powered on and the device can be connected. The aforementioned script will be executed and all modules of *dumpsys* will be copied. When the process is done, the Raspberry powers itself off, notifying the executor that the process is completed. The Android device is now placed in a Faraday Bag (and can be powered off) to transport it to the Digital Forensics Investigation Lab.

In the copied files, the Digital Forensics Investigators found in the module *accounts* an account that was used to sign-in onto a known chatroom where child pornography is exchanged. In the module *Bluetooth_manager* the investigators found the MAC-address of another device that was used in a different child pornography case, thus linking the two cases together.

¹⁰ The screen lock is not removed. Certain services and/or applications no longer work when there is no screen lock in place (think of Android Pay). When the screen lock has been removed. The officers/detectives have altered the device, thus not working forensically sound.