BIASes and BLUFFS against Reproducible Exploits

Elisaveta Lavrentieva

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2024

Abstract

We investigate the reproducibility of Bluetooth Impersonation AttackS (BIAS) and Bluetooth Forward and Future Secrecy (BLUFFS) attacks in our dissertation. Using a Raspberry Pi 3 Model B and a CYW920819M2EVB-01 evaluation board, we are able to reproduce BIAS and successfully attack target devices with it. We analyse the packets captured with various BIAS attacks, and compare them to a regular Bluetooth connection. We show the difficulties in implementing BLUFFS, but confirm that it is likely reproducible.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Elisaveta Lavrentieva*)

Acknowledgements

- Myself for getting through this
- Michio Honda, my supervisor who helped me gain confidence in my abilities to do independent research and guided me along the way
- My family for helping me get the critical piece of hardware that I needed to be able to do this at all
- Jinx, my flatmate's cat
- Chibi, my family's cat
- My friends that listened to me rant about why the attacks aren't working, and were there when I finally got it working
- The corvids outside my window that come and eat the crackers I put out for them
- My laptop and phone for not breaking from all of the Bluetooth exploits I put them against

Table of Contents

Chapter 1

Introduction

Phones, computers, and smartwatches are a few examples of devices with Bluetooth capabilities that we use everyday. They often record sensitive data such as who we call, where we go to for work, or what our heart rate currently is, all to improve their services and our quality of life. With 5 billion Bluetooth devices shipped in 2023, of which 1.22 billion are categorised as data transfer devices and 563 million as location services devices, it is vital to question the security and privacy of modern Bluetooth devices. [1]

Despite the widespread popularity of Bluetooth in our daily lives, there is a worrying lack of research into the safety of Bluetooth and the potential implications of it. The official Bluetooth SIG allows people to report security vulnerabilities they find with the Bluetooth protocol to them, yet only 1 official security notice was made in 2023, and 2 in 2022. [2] While one may consider this to be the result of the Bluetooth protocol being secure, the more probable reason is that not enough security and privacy research is being done into it. 28,961 CVEs were published in 2023, making the 2023 Bluetooth security notice the *only* CVE to be officially recognised by the Bluetooth SIG in that year. [3]

1.1 Goals

While the discovery of new vulnerabilities is important, it is almost useless to report if it cannot be reproduced by others, since it would be impossible to test if patches against the attack work or not. [4] Over the course of this dissertation, we selected two Bluetooth vulnerabilities, BIAS and BLUFFS, that were recognised by the Bluetooth SIG to replicate. [5] [6]

We chose these two because the creator of these exploits, Daniele Antonioli, has released the code to assist others in replicating the attacks. [7] [8] However, the CYW920819EVB-02 evaluation board that was used for the original BIAS proof of concept has been discontinued and is no longer available to purchase. [9] Because of this, there is demand for the original code to be ported to a newer version of the EVB-02 board, which was a primary motivator for us to aim to implement BIAS and BLUFFS.

There is also a lack of guidance on how to implement BIAS in its entirety. The BIAS

paper and code provide simple instructions on how to perform the attack with the files provided in the repository, but it fails to explain how one can patch the attack device to impersonate a device that the repository does not include. The BLUFFS code repository does not contain step-by-step instructions, but it does say what files to run depending on what the user wishes to do.

While the papers give a methodology on how they setup and ran their experiments, we cannot assume that people will read through and fully understand them. It is important that people that wish to recreate these attacks can understand why they are following these steps, and so they can potentially extend on the attack.

The goals of this dissertation are listed as follows:

- Understand and break down the BIAS and BLUFFS attacks
- Adapt the BIAS and BLUFFS code to implement it on an evaluation board that is available to the general public
- Reproduce BIAS and BLUFFS with an accessible setup, and analyse how the attacks worked
- Provide an in-depth guide on how we ran BIAS and BLUFFS

1.2 Contributions

As a result of this dissertation, we have:

- Reimplemented the BIAS attack for the CYW920819M2EVB-01 evaluation board
- Created a guide on how to implement the BIAS attack on a Raspberry Pi 3 Model B and a CYW920819M2EVB-01 evaluation board.
- Created new impersonation files for BIAS and fixed the original code to work with Python 3
- Explained why BLUFFS is not a feasible attack for our setup

Our efforts in reproducing BIAS for the CYW920819M2EVB-01 resulted in the original author of the attack being interested in our guide. [7] Our work is also planned to be incorporated into an open-source framework that tests devices for Bluetooth vulnerabilities, developed by members at ETH Zürich.

1.3 Overview

We split the remaining parts of our dissertation into 5 distinct chapters.

- Chapter 2 introduces the Bluetooth protocol, Secure Simple Pairing, and the Link Manager Protocol. We also introduce the toolkits we use, and explain why we chose to not implement other known Bluetooth attacks.

- Chapter 3 provides a simplified explanation of how the BIAS and BLUFFS vulnerabilities work.
- Chapter 4 explains how we implemented the BIAS attack on the EVB-01 board and successfully used it against an IdeaPad 5. We compare Wireshark logs between a regular Bluetooth connection, failed BIAS attacks, and a successful BIAS attack. We explain the differences we made from the original implementation of BIAS and discuss the potential impacts and limitations of the attack.
- Chapter 5 goes into detail about the types of difficulties we encountered when getting BIAS to work. It also explains why implementing BLUFFS was infeasible to achieve with the resources we had.
- Chapter 6 discusses the impact our project has had on the reproducibility of BIAS and BLUFFS, and we discuss the potential implications our project may have.
- Chapter 7 briefly summarises our project and discusses potential future work that can be done.

Chapter 2

Background

In this chapter, we introduce concepts behind the Bluetooth protocol that are necessary to understand BIAS and BLUFFS, particularly piconets, Secure Simple Pairing, and the Link Manager Protocol. We then introduce BlueZ, InternalBlue, and Wireshark as the tools we use in our methodology. Finally, we explain why we chose to reproduce BIAS and BLUFFS instead of other Bluetooth security exploits.

2.1 Bluetooth

2.1.1 Piconets

Bluetooth networks take the form of piconets, which are defined as at least two devices that are connected by the same physical channel. One device in the piconet is classed as a **central**, and up to seven other devices are called **peripherals**. [11] Figure 2.1 shows three examples of piconets, though we note that example **c** is classed as a scatternet due to two of the centrals being connected.

Devices are allowed to switch roles in a piconet, for example when a peripheral needs to connect to a different peripheral. Centrals and peripherals complete different tasks when performing some protocols, but for our dissertation we only concern ourselves with their roles in establishing sessions. [10]

2.1.2 Secure Simple Pairing

Secure Simple Pairing is the stage that two Bluetooth devices that are unauthenticated to one another must complete so that they can encrypt all future sessions between themselves. Figure 2.2 shows a simplified diagram with the 5 phases: Public key exchange, authentication stage 1, authentication stage 2, Link Key calculation, and encryption.

We explain the separate phases in more detail below:

Phase 1: Initiating device A sends its public key PK_A to device B. Once B receives PK_A, it then responds back to A with its public key PK_B.

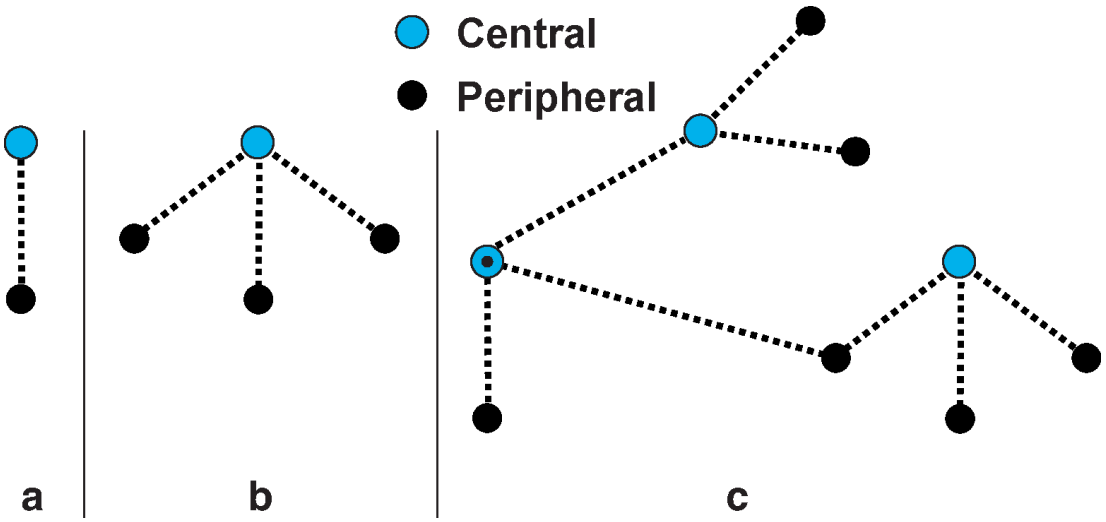


Figure 2.1: Possible configurations of a piconet. [10]

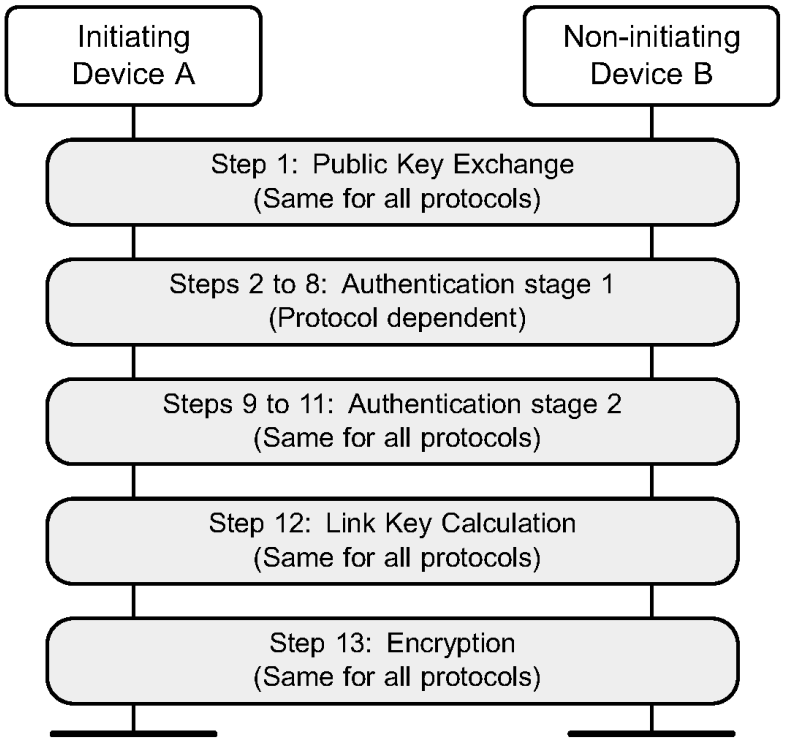


Figure 2.2: The steps of secure simple pairing [12]

		Initiator				
		Display Only	Display YesNo	Keyboard Only	NoInput NoOutput	Keyboard Display
Responder	Display Only	Just Works	Just Works	Passkey Entry ●	Just Works	Passkey Entry ●
	Display YesNo	Just Works	Numeric Comparison	Passkey Entry ●	Just Works	Numeric Comparison
	Keyboard Only	Passkey Entry ●	Passkey Entry ●	Passkey Entry ●	Just Works	Passkey Entry ●
	NoInput NoOutput	Just Works	Just Works	Just Works	Just Works	Just Works
	Keyboard Display	Passkey Entry ●	Numeric Comparison	Passkey Entry ●	Just Works	Numeric Comparison

● Responder displays, Initiator inputs
● Initiator displays, Responder inputs
● Initiator inputs and Responder inputs

Figure 2.3: Table of the resulting authentication protocol depending on the IOCaps. [13]

Phase 2: The process diverges depending on what type of authentication protocol the devices must complete. The authentication protocols are decided on by the input and output capabilities (**IOCaps**) of the devices. Figure 2.3 shows what authentication protocol will be selected based on what IOCaps the devices have. We do not need to understand the specifics of the separate authentication methods for BIAS and BLUFFS, so we have left them out of this explanation.

Phase 3: All processes converge back to following the same steps again, and each device computes a new value E. Device A sends B its computed E_A, which B then checks to see if it can compute the same value as E_A. If successful, then B sends A its computed E_B, which A then checks in a similar fashion.

Phase 4: At this point the devices compute the **Link Key**. The part we stress here is that there is no possibility for a user that is eavesdropping can know the Link Key from passively sniffing the connection between A and B.

Phase 5: The final phase has the two devices compute the encryption key with which to encrypt all future communications between themselves.

2.1.3 Link Manager Protocol

The Link Manager dictates how Bluetooth devices connect to and disconnect from one another. It contains the **Link Manager Protocol** (LMP), which sends **Packet Data Units** (PDU). [15]

All LMP PDUs have an **opcode** at the start of its message to inform the receiving device what the body of the packet contains. Each LMP PDU is one of either structure depending on what type of opcode it has:

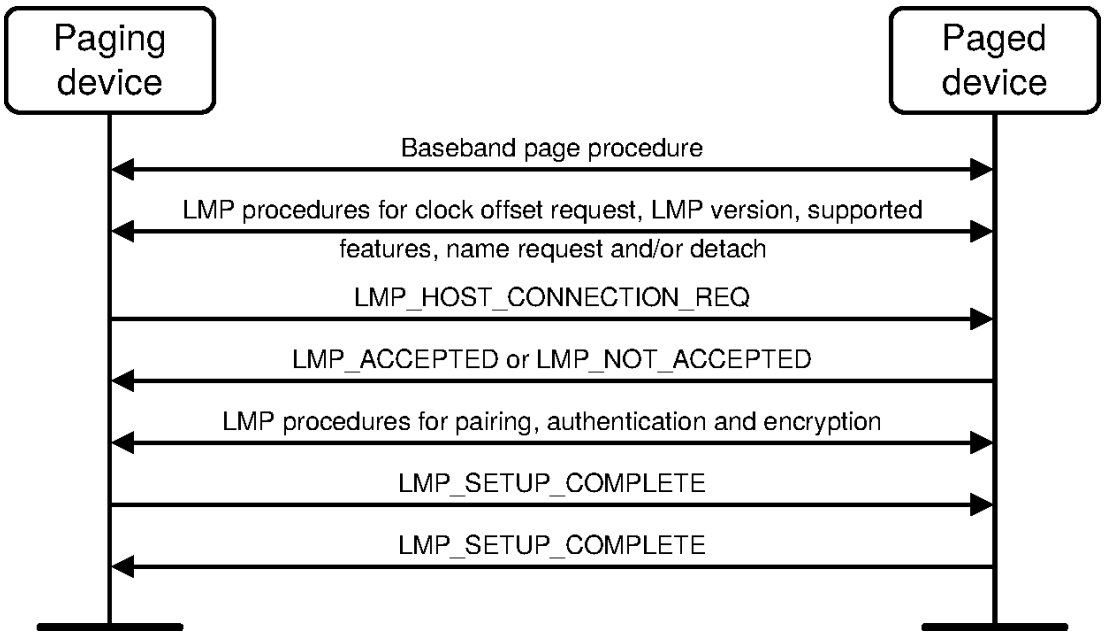


Figure 2.4: LMP connection establishment steps. Either the central or peripheral can start a connection. [14]

- 1 bit transaction ID, a 7 bit opcode, followed by up to 17 bytes of payload.
- 1 bit transaction ID, a 15 bit extended opcode, followed by up to 16 bytes of payload.

When a device wishes to connect to a device, it follows the LMP procedures to establish a connection between themselves. Figure 2.4 shows what LMP packets are sent between the devices when establishing a connection. [14]

2.2 Toolkits

We give a brief overview of the three tools we use in our methodology. We do not go into depth how these tools work as we only need to understand what they do, and how we can use them to our advantage.

2.2.1 BlueZ

BlueZ is an open source implementation of the Bluetooth protocol for Linux devices. It also installs commands that we use when gathering information about the target device to impersonate. `bluetoothctl` provides command-line access to BlueZ, allowing us to perform actions such as making our Bluetooth device pairable, or to scan the nearby environment for any discoverable devices. [16]

Other commands we use from BlueZ include: `hcitool`, `btmon`, and `btattach`.

2.2.2 InternalBlue

InternalBlue was created as part of Dennis Mantz's Masters thesis, designed to be a low-level Bluetooth experimentation framework for security research purposes. [17] It features powerful commands such as sending hand-crafted LMP packets over a specified connection, or overwriting parts of the memory of our Bluetooth chip.

The code developed for BIAS and BLUFFS use InternalBlue for its low-level capabilities, though the attacks were created when InternalBlue was written in Python 2. InternalBlue has since been rewritten in Python 3, and we encounter issues that we discuss in Chapter 5.

2.2.3 Wireshark

Wireshark is a popular open-source network packet analyser that provides a user-friendly interface to easily search through captured packets and look at their contents. [18] We use Wireshark to record Bluetooth packets we receive during our tests, and then to investigate whether an attack attempt was successful or not. We also install a plugin to allow Wireshark to read capture LMP packets, made by Classen et. al. [19]

2.3 Bluetooth Attacks

We chose the BIAS and BLUFFS vulnerabilities instead of other vulnerabilities because they were made by the same author, have code also made by the same author, and have been officially recognised by the Bluetooth SIG. [5] [6]

We chose not to implement other vulnerabilities that fit this criteria due to time constraints, and the high likelihood that they would require a different setup to what was needed for BIAS and BLUFFS. Some vulnerabilities also built off of what BIAS and BLUFFS did, meaning we had to understand those two attacks before we could approach new vulnerabilities.

Finally, we also wanted to implement vulnerabilities that were discovered in the past 5 years, as vulnerabilities that were found after that time frame were less likely to be exploitable as devices and the Bluetooth protocol were patched to defend against them.

Some attacks we do not explain in the following chapter but we considered implementing are: Method Confusion attack, [13] Blacktooth, [20] BRAKTOOTH, [21] and BLESA. [22]

2.4 Previous Work

To our knowledge, there has only been 1 other successful attempt at implementing BIAS on an evaluation board other than the CYW920819EVB-02. [23] It uses the CYW920735Q60EVB-01 evaluation board, which is still available for purchase. However, it does not explain the steps they took to get the attack to work, but they do say how to get the information necessary to create an impersonation file. We improve on

this work by explaining how we got the CYW920819M2EVB-01 evaluation board to work with all of the tooling needed. We also explore the differences between a regular Bluetooth connection and a BIAS connection.

Blacktooth implements BIAS using the EVB-02 as part of its chain of attack to achieve remote command execution on the victim device, showing that BIAS was reproducible when the EVB-02 was available to purchase. [20] We did not find any papers that used a device other than the EVB-02 to implement BIAS during our research.

Chapter 3

BIAS and BLUFFS

We introduce the Bluetooth vulnerabilities, BIAS and BLUFFS, that we focus our project on. We provide a high-level overview of them along with the phases involved. All details on the attacks in this chapter come from Antonioli et. al. [24][25]

3.1 Bluetooth Impersonation AttackS (BIAS)

Bluetooth Impersonation AttackS, more commonly known as BIAS, is a security vulnerability in the Bluetooth Protocol that allows an attacker to 'impersonate' a device and then connect to a victim without knowing the Link Key between the victim and the impersonated device. Figure 3.1 depicts how BIAS can allow Charlie, the attacker, to communicate with Alice or Bob, the victims, of which have paired together before the attack. If communicating with Bob, then Charlie impersonates Alice, and vice versa.

Before Charlie can perform BIAS, they must patch their Bluetooth chip with the same Bluetooth address, name, LMP feature profiles, LMP version and subversion, company ID, and device class as the victim they want to impersonate. We explain this process more in Chapter 4 as the information gathering is a valid part of the Bluetooth protocol, and not a vulnerability caused by BIAS.

3.1.1 Legacy Secure Connections BIAS

BIAS exploits the lack of mutual authentication during the Simple Pairing process with Legacy Secure Connections. Charlie can be either a central or a peripheral device at the start of the pairing process, as BIAS follows the same simplified steps:

1. Begin a connection with the victim
2. **Peripheral only**: Switch roles to become the central device
3. Send the victim a challenge to solve as part of the authentication process
4. Finish pairing

Charlie never solves a challenge from the victim, because the victim does not ever check if Charlie solves it. If Charlie starts off as a peripheral device, they must perform a role switch to become the central device because then they do not need to solve the challenge they send to the victim.

3.1.2 Secure Connections BIAS

There are two types of BIAS attacks for Secure Connections, those being the **downgrade** and the **reflection** attacks. The downgrade attacks exploit the ability to change from Secure Connections to Legacy Secure Connections if one user does not support Secure Connections. The simplified steps for BIAS downgrade attacks are as follows:

1. Begin a connection with the victim
2. Declare that Secure Connections are not supported
3. Downgrade to Legacy Secure Connections
4. Follow the steps for Legacy Secure Connections BIAS

The BIAS reflection attacks exploits the ability to switch roles once the challenges for the authentication procedure have been sent. The simplified steps for these attacks are as follows:

1. Begin a connection with the victim
2. **Peripheral only**: Switch roles to become the central device
3. Exchange challenges with the victim to begin the authentication process
4. While waiting for the victim to solve the first challenge, switch roles to become the peripheral device
5. Receive the response from the victim, and then send it back

At the point when Charlie and the victim exchange challenges, this begins phase 3 of the Secure Simple Pairing procedure as explained in Subsection 2.1.2. The victim sends Charlie E_V as the response to the challenge. At this point the victim would expect to receive E_C from Charlie, but because Charlie changed roles the victim must change as well. This makes the victim want E_V in response instead, and so Charlie sends it back and the pairing procedure successfully completes.

As we will be checking our implementation of BIAS with the Secure Connections downgrade peripheral attack in Chapter 4, we present a more in-depth explanation for the attack in Figure 3.2.

3.2 BLUetooth Forward and Future Secrecy (BLUFFS) Attacks

BLUFFS was discovered in November 2023, and builds off of BIAS and a previous attack by Antonioli et. al called KNOB.

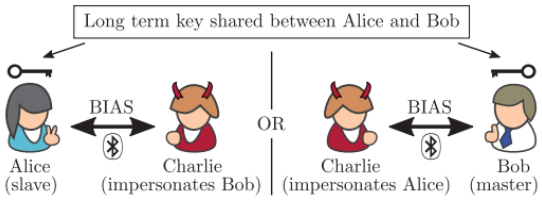


Figure 3.1: A simple diagram showing the end results of BIAS. [24]

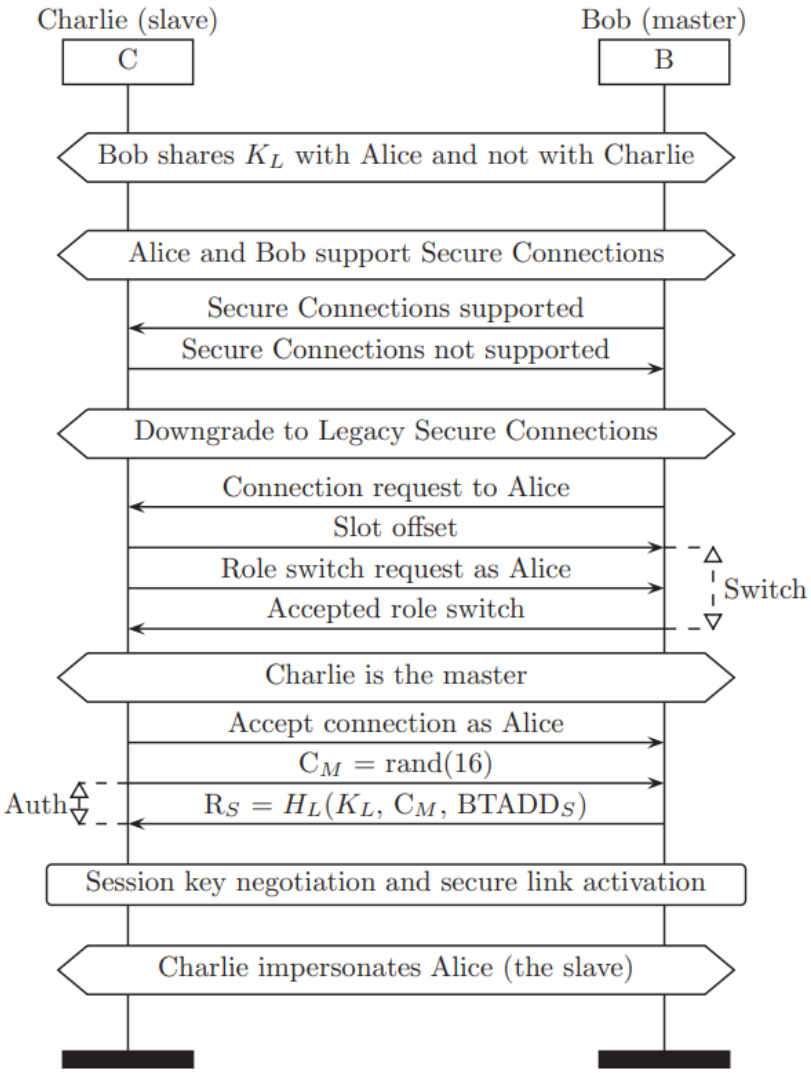


Figure 3.2: Process of the BIAS peripheral attack which we implement. This exploits downgrading from Secure Connections to Legacy Secure Connections. [24]

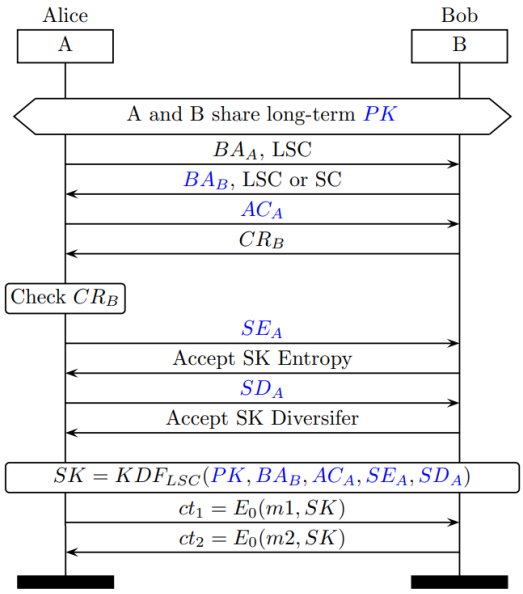


Figure 3.3: Session Key derivation during Simple Pairing. [25]

The KNOB attack can be succinctly explained as forcing the encryption key K used by the victims to have an entropy of 1 byte, meaning that Charlie can quickly brute force K by checking it against 512 pre-computed values. [26] This exploit trivialises Bluetooth encryption, but companies swiftly patched devices against the exploit. [27] This attack is not essential to implement BLUFFS, which is why we chose to not implement KNOB.

BLUFFS manages to break both the forward and future secrecy guarantees made by the Bluetooth protocol, enabling Charlie to decrypt any messages sent between the victims Alice and Bob no matter whether they are from the past or being actively sniffed. It does this by manipulating the values used in the variables when deriving a Session Key for the new connection between Alice and Bob.

Figure 3.3 shows which variables are used to establish the Session Key *SK*. *PK* is the long-term Pairing Key *BA* is the Bluetooth Address, *CA* is a challenge, *CR* is the response to the challenge, *SE* is the entropy of *SK*, and *SD* is the the *SK* diversifier. The purpose of *SK* is to protect both past and future connections if *SK* is discovered, as a new Session Key is for every connection.

Charlie would perform the following actions if they were targeting Bob:

1. Start a connection with Bob, where Charlie impersonates Alice using the same techniques as in BIAS.
2. During the Session Key establishment phase, Charlie sends a **constant** AC_C to Bob and ignores CR_B.
3. *SE* is set as the lowest possible entropy, which is 1 if Bob is vulnerable to KNOB, or 7 if not.
4. *SD* is also constant, which makes Bob negotiate an *SK* that can be brute-forced.

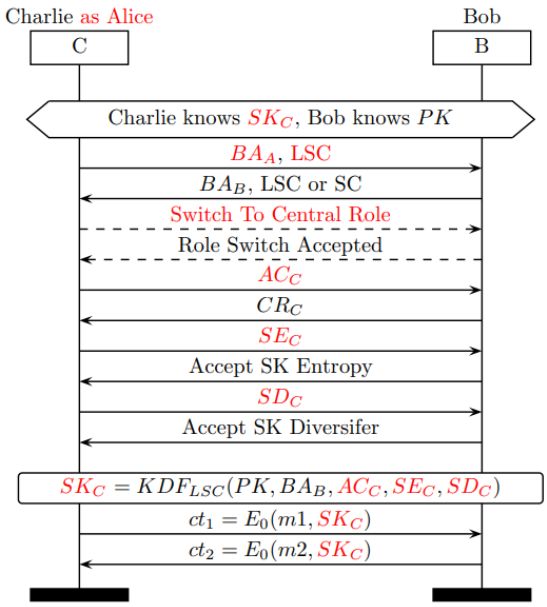


Figure 3.4: Session Key derivation, but Charlie manipulates the values used for deriving SK. [25]

5. Charlie brute forces SK, which can be done offline.
6. Once brute forced, Charlie can now decrypt all future and past messages between Bob and Alice.

Unfortunately, the time it would take for Charlie to brute-force SK using commercial equipment would take several weeks. [25] We discuss this issue more in Chapter 5.

Chapter 4

Implementing BIAS

This chapter explains the implementation of the BIAS attack by using a CYW920819M2-EVB-01 evaluation board and a Raspberry Pi 3 Model B, and discusses the differences between the original implementation and this one.

4.1 Setup

To install InternalBlue with all of its functionalities, we need `pwntools`, a Python 3 library for exploitation development and Capture The Flag competitions. [28] However, since `pwntools` is not available for the Raspbian OS, we installed Ubuntu Server 22.04.4 LTS on our Raspberry Pi 3 to easily install `pwntools` and hence InternalBlue.

To run the attack, we need to recon essential information from the device we impersonate. As explained in Section 3.1, we must gather the device's Bluetooth address, name, LMP version and subversion, company ID, LMP features, and device class. These can all be obtained by querying the device and reading the responses given, but we are unable to read all of the response packets with the basic Linux kernel on a Raspberry Pi, because the packets begin with the `0x07` prefix, which the kernel marks as a Broadcom diagnostic packet. The packet then is sent down to a management pipe instead of the HCI pipe that the other packets go down which InternalBlue, Wireshark, and `btmon` watch, meaning we miss essential packets. [29]

To capture the diagnostic Bluetooth packets, we patch the 4.14 Linux kernel. Using the kernel patching files by Antonioli, we remove all of the code that allows diagnostic parsing and add a new file called `h4_recv.h`. [30] This new file comes from the Android kernel's Bluetooth files that acts as a generic Bluetooth driver helper. [31] This lets the kernel accept Bluetooth packets with a `0x07` prefix. When we attach the CYW board later in the process, we can then execute the following command:

```
echo 1 | sudo tee /sys/kernel/debug/bluetooth/hci1/vendor_diag
```

This makes the Broadcom chip send out diagnostic messages, which includes LMP packets that we can now read with the patched kernel. [29]

4.2 Differences from the Original Implementation

We use the Linux kernel 4.14.111 to minimise divergence from the original implementation. However, there were issues when compiling the kernel that are unrelated to the patches made, meaning it was necessary to change other files in the kernel for it to compile successfully. The files changed are: `scripts/dtc/dtc-lexer-lex.c`, `scripts/dtc/dtc-lexer-lex.c_shipped` [32], `include/linux/bitfield.h` [33], and `.config` [34]. This should not affect the implementation of BIAS, as none of the changes affect the Bluetooth modules.

One of the major challenges we encountered with the original paper was that an essential piece of equipment, the CYW920819EVB-02 from Infineon, was discontinued. [9] We found that the CYW920819M2EVB-01 is a suitable replacement because it can still be purchased and it has the same memory addresses as the original board. We go into more detail about this challenge in Chapter 5.

InternalBlue does not have a firmware file dedicated to the EVB-01, but does for the EVB-02. This is because the EVB-02 has the chip identifier `0x220c`, whereas the EVB-01 has the chip identifier `0x2305`. InternalBlue uses the chip identifier to either load the specialised firmware files with addresses for the memory, or to load the generic one. Some functions require addresses to be defined to let them be called by InternalBlue, which means the generic firmware file cannot use those functions. To enable InternalBlue to change the firmware and send HCI commands to the EVB-01, we copied the `fw_0x220c.py` file to a new `fw_0x2305.py` file. Now, when we start up InternalBlue, it will match the chip identifier `0x2305` to the new file and we can use the specialised functions.

The original BIAS files were written in Python 2 since InternalBlue was written in Python 2 when the first proof of concept was made, but InternalBlue has since changed over to using Python 3 and strongly recommends people to use the new version. Every change to the original BIAS files made can be found in the materials provided. These are also discussed more in Chapter 5.

4.3 Implementation

In this section, we show an example of the BIAS attack where we impersonate a WH-CH510 headset to have an IdeaPad 5 connect to the attack device instead of the actual WH-CH510 device.

4.3.1 Setting Up the CYW Board

We can mount the CYW evaluation board with diagnostics mode enabled by running:

```
btattach -B /dev/(file where CYW is attached) -S 115200 -P bcm
```

This command changes the attack device's Bluetooth controller to the CYW board, sets the baud rate to 115200 Bd, and sets the protocol that talks to the board to a Broadcom protocol. [35] While `btattach` usually detects what protocol to use to talk to the

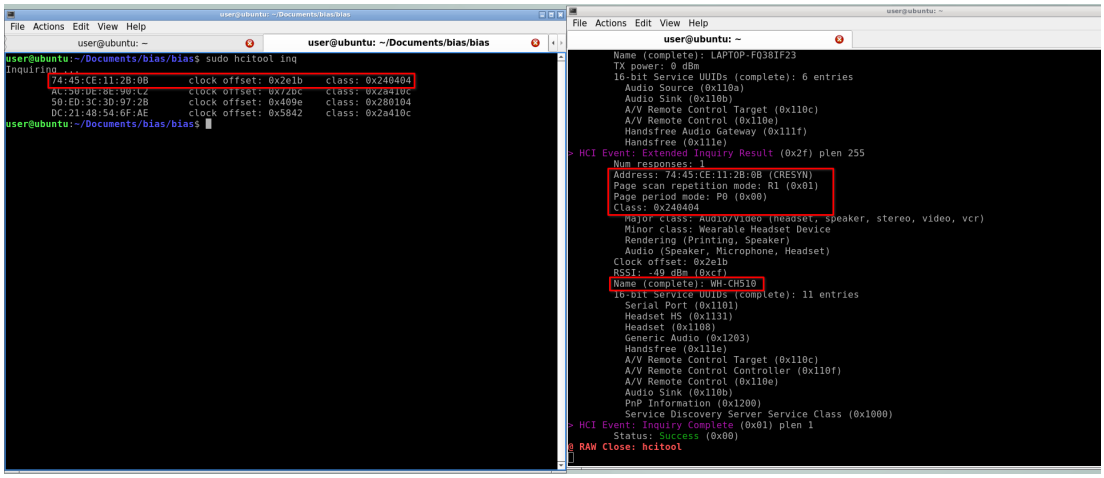


Figure 4.1: `hcitool inq` results with necessary data highlighted

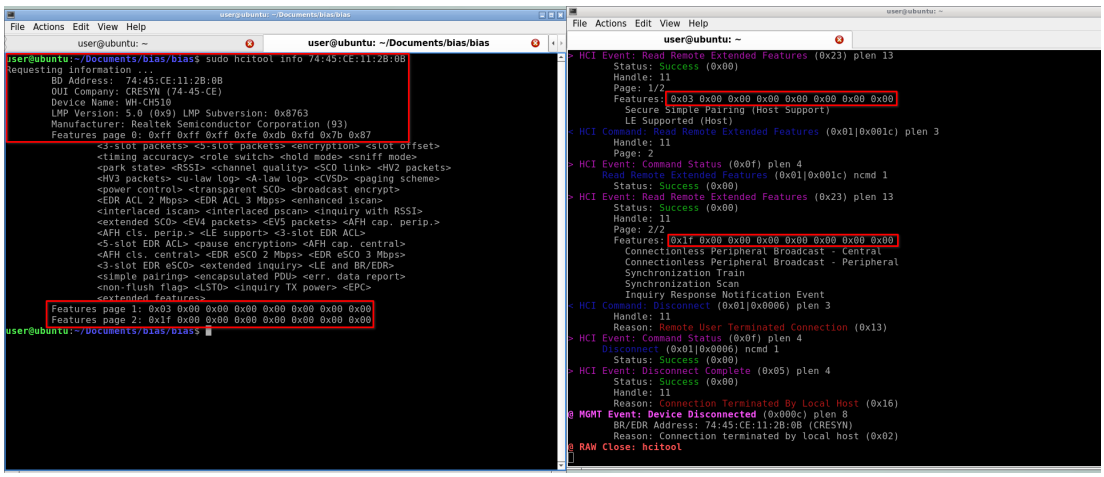


Figure 4.2: `hcitool info` results with necessary data highlighted

Bluetooth chip, the CYW board appears like a Cypress chip to it, so it will not use the Broadcom protocol we require for the diagnostic messages. The chip is fully compatible with the Broadcom protocol, so we can explicitly define it when mounting the board.

We now can run the command in section Section 4.1 to enable the diagnostic messages to be sent.

4.3.2 Device Impersonation

We never need to connect to the WH-CH510 to gather the necessary information about it. However, we do need it to be discoverable for our attack device to find it and query it the necessary packets.

First, we need to have `btmon` running on the side to show the packets being captured. Next, we can run `hcitool inq` to have our device search for nearby devices. Once completed, we get the Bluetooth address and class of the devices it finds. In `btmon` we can see similar information as well as the name of the device, as visible in Figure 4.1

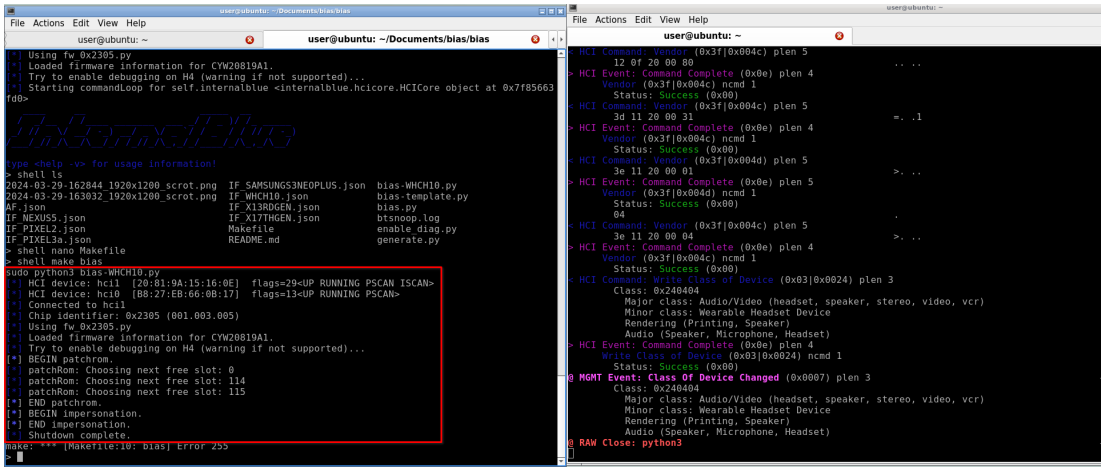


Figure 4.3: InternalBlue patching the CYW board to match the WH-CH510

To get the rest of the data values we need, we can run `hcitool info (address)`. This will show the LMP feature pages, LMP version and subversion, and the manufacturer, as seen in Figure 4.4.

We can now create an impersonation file for the BIAS code to use when changing the CYW firmware. It is important to write the device class value in little endian order in the file instead of the big endian order that is visible from `btmon` and `hcitool`, otherwise the CYW board will not appear like the impersonated device's class. We have made 2 impersonation files available, one for the WH-CH510 and another for the Pixel 3a.

We keep the lmin and lmax values as `07` as we did not implement the KNOB attack, so we cannot know whether the impersonated device can create a 1 byte session key or not. However, it may be possible to test the KNOB attack during BIAS by setting the lmin value to `01` and seeing if it will pair to the victim, since if the victim is patched against KNOB then it should reject the connection request.

Before we move onto the next steps, we decided to reattach the CYW board as mentioned in section 4.3.1 but without the `-P bcm` flag as we found that some victim devices have difficulty connecting to the attack device with this flag set. We also need to start up `bluetoothctl` and set the Bluetooth controller to be discoverable before we patch it, as if we set it after we patch our device then `bluetoothctl` will change the class back to its actual value, destroying the integrity of the impersonation of the attack device.

From this point we follow the steps as outlined by the original BIAS Github repository. We start up InternalBlue and initiate Wireshark from it, then we run `make generate` to create the BIAS python file and `shell make bias` inside InternalBlue to patch the device.

As seen by `btmon` in Figure 4.3, InternalBlue is accessing parts of the memory of the CYW board and rewriting the original Bluetooth values with the ones we specified in the impersonation file. When it is done, the board is virtually identical to the impersonated device to the point where if we exit InternalBlue and try to start it up again, it will trip an error warning as the impersonated device does not use a Broadcom or Cypress chip, making the tool exit early and forcing us to start from the beginning of the BIAS attack.

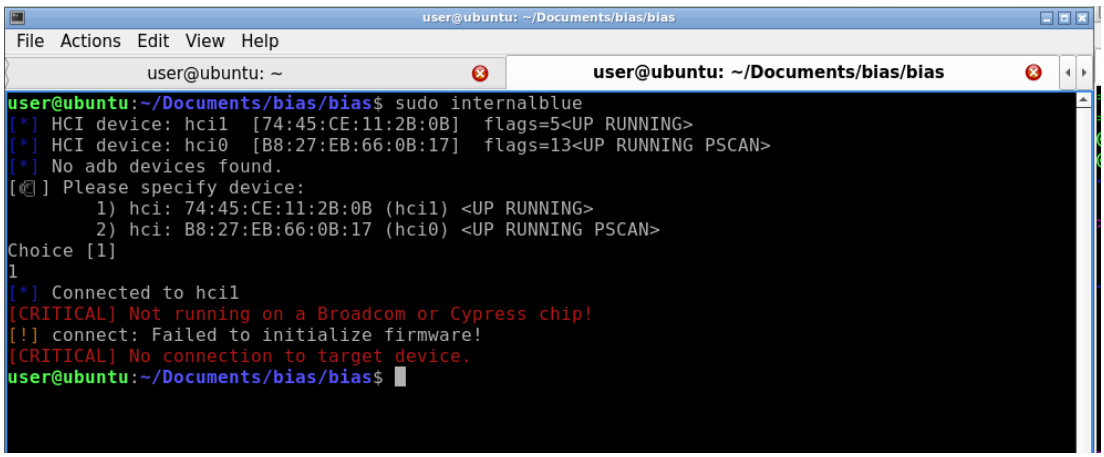


Figure 4.4: InternalBlue cannot be started again after the CYW board is patched

4.3.3 Running the Attack

The attack device is now ready to connect to the IdeaPad 5. Before we do so, we pair the WH-CH510 to the IdeaPad 5 to ensure that they trust each other, and then we turn off the WH-CH510 to prevent any interference from it during the connection process.

The IdeaPad 5 attempts to connect to the WH-CH510. At this point, it will find our attack device and see that it has the same information values as the ones stored in the IdeaPad 5's trusted devices list. This tricks the victim into believing that the attack device is actually the original WH-CH510, and connects to it without the attack device ever authenticating itself to it. This is the BIAS peripheral impersonation attack working.

4.4 Comparing Wireshark Logs

To show that this is the BIAS attack working, we compare the Wireshark PCAP files between a normal Bluetooth connection, a failed BIAS attempt, and a successful BIAS attempt. The full Wireshark logs can be found in the materials. The normal Bluetooth connection is between a Pixel 3a and the EVB-01 board, where the EVB-01 is unaltered. The BIAS attempts are between a victim IdeaPad 5 and the EVB-01 board, of which is impersonating the same Pixel 3a from the normal Bluetooth connection logs.

To keep the figures shorter and simpler to read, we filter out all captured LMP packets and only show the packets in the time frame that is relevant to the discussion. All of the Wireshark logs can be found in their entirety in the materials provided. We were not able to record Wireshark logs of a regular Bluetooth connection between the EVB-01 and the IdeaPad 5, which we explain in Section 5.1.

Figure 4.5, Figure 4.7, and Figure 4.9 are provided to draw attention to the packets we compare in the following sections. Red represents the final Simple Pairing protocol packet, blue represents the Link Key packets, green shows if the target device accepted the pairing, and yellow shows the IOCaps packets.

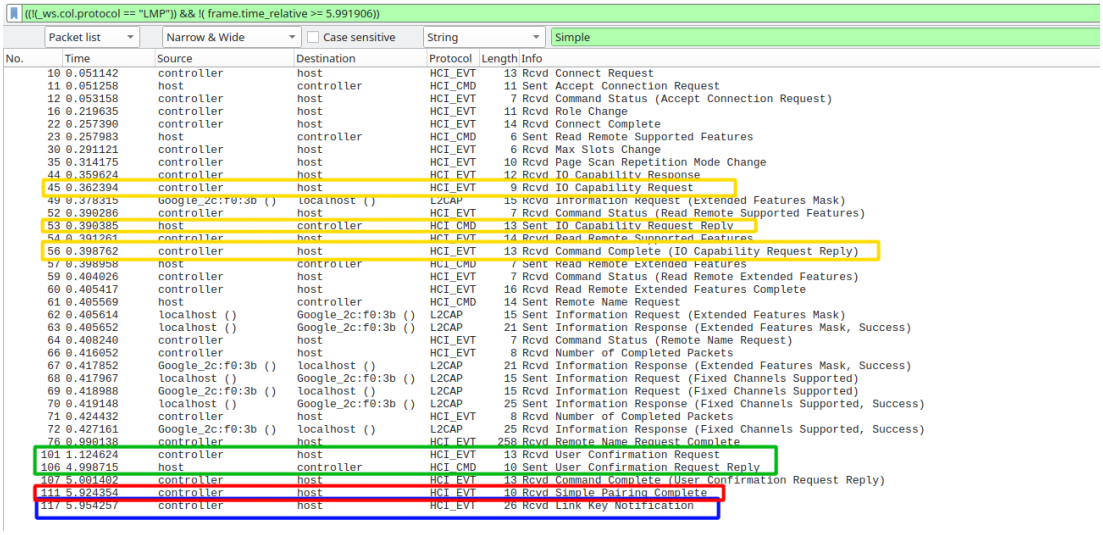


Figure 4.5: Annotated shortened Wireshark logs of a regular Bluetooth pairing protocol between a Pixel 3a and the EVB-01 board. Refer to Section 4.4 for a guide on the colours.

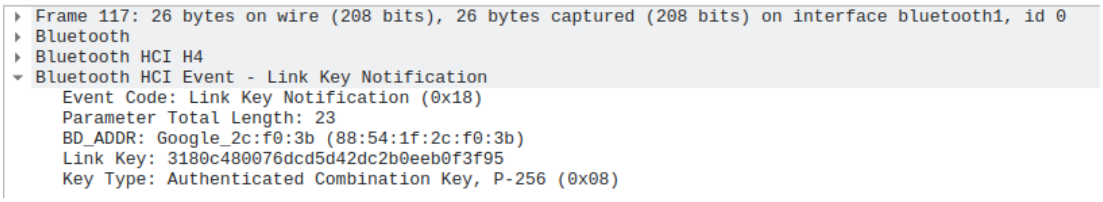


Figure 4.6: Contents of the highlighted Link Key packet in Figure 4.5.

4.4.1 Regular Bluetooth Connection between the EVB-01 and a Pixel 3a

In Figure 4.5, we present the Wireshark logs of a Pixel 3a connecting to the EVB-01 using the Bluetooth protocol. The Pixel 3a is paired to the EVB-01 board beforehand to keep the Wireshark logs relevant to the lifetime a Bluetooth connection only.

This connection follows the LMP procedure for establishing a connection, explained in Subsection 2.1.3. One key difference in Figure 4.5 from Figure 4.7 and Figure 4.9 is that the Link Key packet is only received **after** the Simple Pairing protocol is completed. Figure 4.6 shows the contents of the packet. We stress that this packet contains the Link Key to encrypt all future messages.

4.4.2 Successful BIAS attack between an Impersonated Pixel 3a and an IdeaPad 5

Figure 4.7 shows a successful BIAS peripheral attack from our experiments. The EVB-01 impersonates a Pixel 3a that is trusted by the victim IdeaPad 5. We note that the Link Key is negotiated immediately after the connection was accepted, and there is no packet containing the Link Key after the Simple Pairing protocol is completed. The

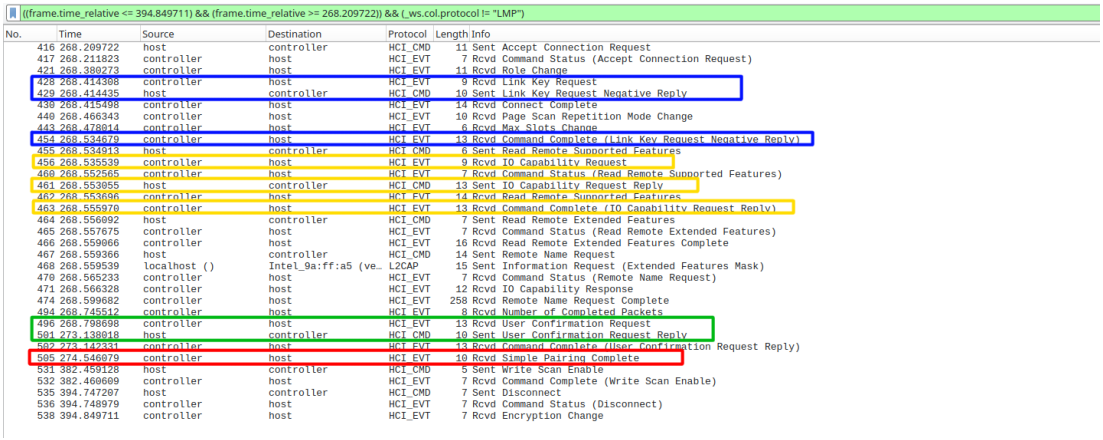


Figure 4.7: Annotated shortened Wireshark logs of a successful BIAS peripheral attack. The CYW board impersonates a Pixel 3a that the IdeaPad 5 victim trusts. Refer to Section 4.4 for a guide on the colours.

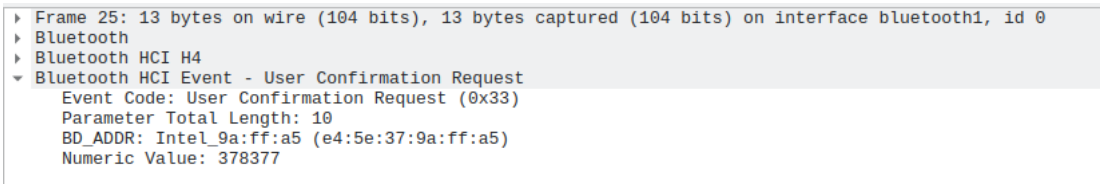


Figure 4.8: Contents of the first User Confirmation Request packet in Figure 4.7. `0x33` means the sender has authenticated the other user.

connection is sustained until we terminate the connection from the EVB-01.

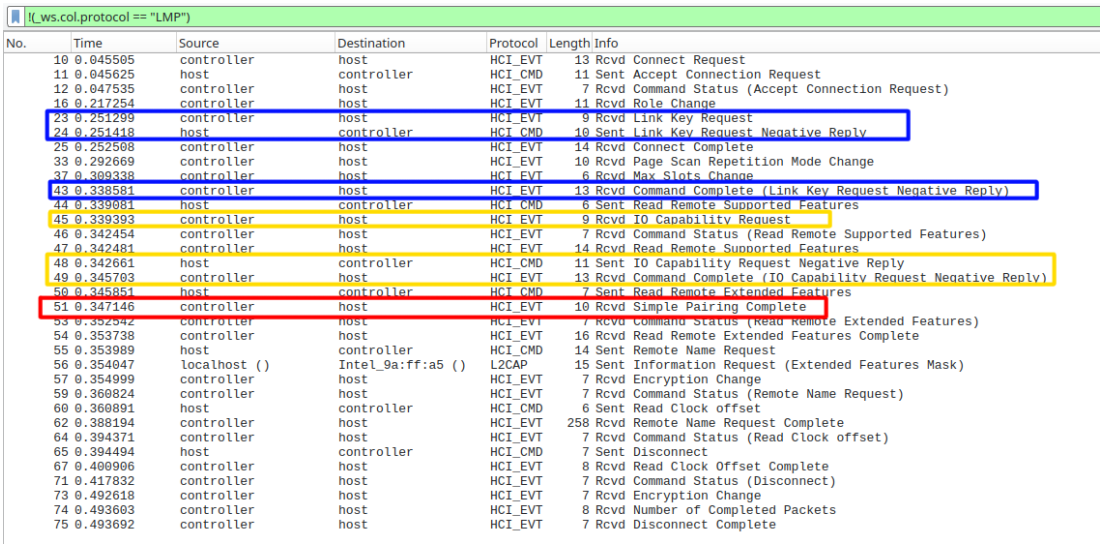
We follow the LMP procedure like before. As the EVB-01 is patched to have the same information as the Pixel 3a that the IdeaPad 5 requests, the IOCaps match the details the IdeaPad 5 has stored for its profile on the Pixel 3a. This leads into the authentication phase of the procedure, which the EVB-01 responds that it correctly verified the PIN it obtained in Figure 4.8, though it never actually checks the PIN.

The IdeaPad 5 believes that the EVB-01 is actually the Pixel 3a as it claimed to have calculated the same number as the PIN. The Simple Pairing procedure is then completed as both parties are satisfied with the calculated number, and the BIAS peripheral attack is successfully performed. At no point during this connection do we receive a Link Key value like that of Figure 4.6. Because of this, the connection between the IdeaPad 5 and the EVB-01 never gets encrypted, despite how the final packet in Figure 4.7 says that the encryption is turned off once the EVB-01 is disconnected.

4.4.3 Failed BIAS attacks between an Impersonated Pixel 3a and an IdeaPad 5

We present two examples of a failed BIAS attempt in this section, and we explain why they failed.

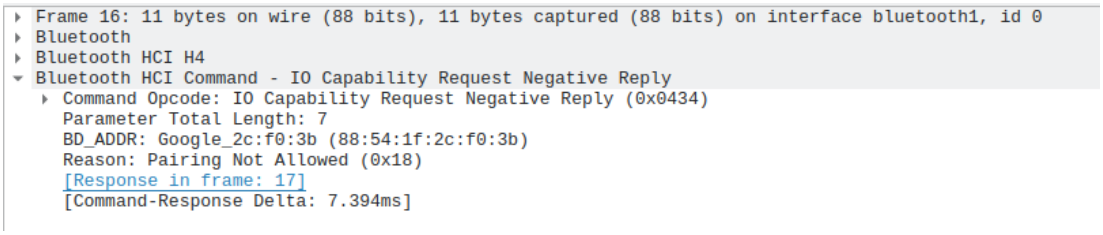
In this example, we patched the EVB-01 in the same way as before, but we did not set



!(_ws.col.protocol == "LMP")

No.	Time	Source	Destination	Protocol	Length	Info
10	0.045505	controller	host	HCI_EVT	13	Rcvd Connect Request
11	0.045625	host	controller	HCI_CMD	11	Sent Accept Connection Request
12	0.047535	controller	host	HCI_EVT	7	Rcvd Command Status (Accept Connection Request)
16	0.217254	controller	host	HCI_EVT	11	Rcvd Role Change
23	0.251299	controller	host	HCI_EVT	9	Rcvd Link Key Request
24	0.251418	host	controller	HCI_CMD	10	Sent Link Key Request Negative Reply
25	0.252508	controller	host	HCI_EVT	14	Rcvd Connect Complete
33	0.292669	controller	host	HCI_EVT	10	Rcvd Page Scan Repetition Mode Change
37	0.309338	controller	host	HCI_EVT	6	Rcvd Max Slots Change
43	0.338581	controller	host	HCI_EVT	13	Rcvd Command Complete (Link Key Request Negative Reply)
44	0.339081	host	controller	HCI_CMD	6	Sent Read Remote Supported Features
45	0.339393	controller	host	HCI_EVT	9	Rcvd IO Capability Request
46	0.342454	controller	host	HCI_EVT	7	Rcvd Command Status (Read Remote Supported Features)
47	0.342481	controller	host	HCI_EVT	14	Rcvd Read Remote Supported Features
48	0.342661	host	controller	HCI_CMD	11	Sent IO Capability Request Negative Reply
49	0.345703	controller	host	HCI_EVT	13	Rcvd Command Complete (IO Capability Request Negative Reply)
50	0.345851	host	controller	HCI_CMD	7	Sent Read Remote Extended Features
51	0.347146	controller	host	HCI_EVT	10	Rcvd Simple Pairing Complete
53	0.352542	controller	host	HCI_EVT	7	Rcvd Command Status (Read Remote Extended Features)
54	0.353738	controller	host	HCI_EVT	16	Rcvd Read Remote Extended Features Complete
55	0.353989	host	controller	HCI_CMD	14	Sent Remote Name Request
56	0.354047	localhost ()	Intel_9a:ff:a5 ()	L2CAP	15	Sent Information Request (Extended Features Mask)
57	0.354999	controller	host	HCI_EVT	7	Rcvd Encryption Change
59	0.360824	controller	host	HCI_EVT	7	Rcvd Command Status (Remote Name Request)
60	0.360891	host	controller	HCI_CMD	6	Sent Read Clock offset
62	0.388194	controller	host	HCI_EVT	258	Rcvd Remote Name Request Complete
64	0.394371	controller	host	HCI_EVT	7	Rcvd Command Status (Read Clock offset)
65	0.394494	host	controller	HCI_CMD	7	Sent Disconnect
67	0.400906	controller	host	HCI_EVT	8	Rcvd Read Clock Offset Complete
71	0.417832	controller	host	HCI_EVT	7	Rcvd Command Status (Disconnect)
73	0.492618	controller	host	HCI_EVT	7	Rcvd Encryption Change
74	0.493603	controller	host	HCI_EVT	8	Rcvd Number of Completed Packets
75	0.493692	controller	host	HCI_EVT	7	Rcvd Disconnect Complete

Figure 4.9: Annotated shortened Wireshark logs of a failed BIAS peripheral attack. The CYW board impersonates a Pixel 3a that the IdeaPad 5 victim trusts, but the attack does not work. Refer to Section 4.4 for a guide on the colours.



```
▸ Frame 16: 11 bytes on wire (88 bits), 11 bytes captured (88 bits) on interface bluetooth1, id 0
▸ Bluetooth
▸ Bluetooth HCI H4
▾ Bluetooth HCI Command - IO Capability Request Negative Reply
  ▸ Command Opcode: IO Capability Request Negative Reply (0x0434)
    Parameter Total Length: 7
    BD_ADDR: Google_2c:f0:3b (88:54:1f:2c:f0:3b)
    Reason: Pairing Not Allowed (0x18)
    [Response in frame: 17]
    [Command-Response Delta: 7.394ms]
```

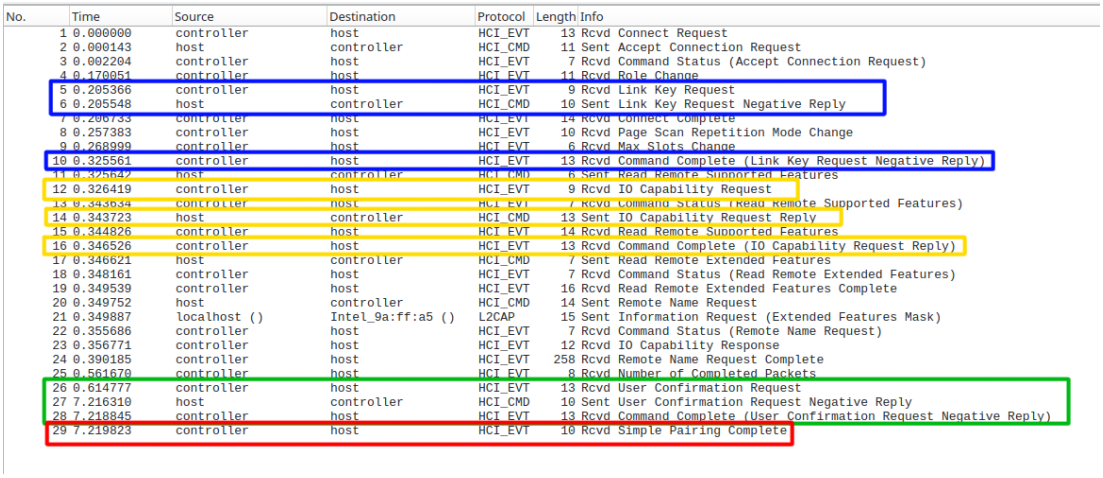
Figure 4.10: Contents of the sent IOCaps packet in Figure 4.9.

the EVB-01 to be pairable. Figure 4.9 shows the Wireshark logs of the failed connection. Because of this mistake, when the IdeaPad 5 requests for the EVB-01's IOCaps, the EVB-01 refuses to provide them and returns the packet shown in Figure 4.10. As a result, the LMP protocol dictates that the connection establishment attempt should be stopped, and the Simple Pairing packet contains the code `0x05`, meaning that the PIN or Link Key was incorrect.

In our other example of a failed BIAS attempt, we correctly set the EVB-01 to be pairable. Figure 4.11 shows the logs of the attempt. This time, the EVB-01 sends the correct IOCaps, but fails to send the correct User Confirmation reply. Taking a closer look at the LMP packets sent during this time in Figure 4.12, we see in packet 371 that the Numeric Comparison failed.

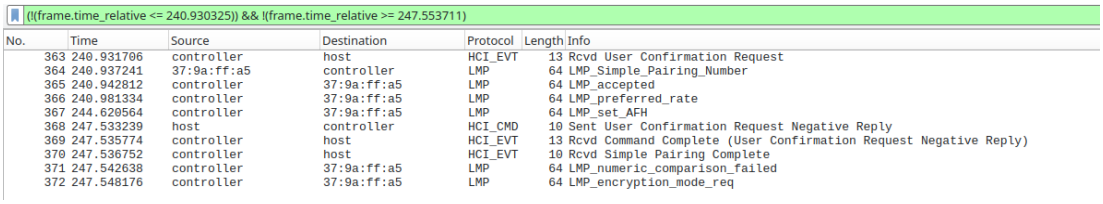
Investigating the body of the User Confirmation negative response in Figure 4.13, we find that the error code is `0x2d`. This error code means the Quality of Service parameters were rejected by the EVB-01, of which we cannot control ourselves.

We followed the steps we listed out in Chapter 4 that resulted in successful BIAS attacks. The Quality of Service parameters that resulted in this error are decided on in the Baseband protocol, which is out of scope for this project. We assume that



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	controller	host	HCI_EVT	13	Rcvd Connect Request
2	0.000143	host	controller	HCI_CMD	11	Sent Accept Connection Request
3	0.002204	controller	host	HCI_EVT	7	Rcvd Command Status (Accept Connection Request)
4	0.170051	controller	host	HCI_EVT	11	Rcvd Role Change
5	0.205366	controller	host	HCI_EVT	9	Rcvd Link Key Request
6	0.205548	host	controller	HCI_CMD	10	Sent Link Key Request Negative Reply
7	0.206733	controller	host	HCI_EVT	14	Rcvd Connect Complete
8	0.257383	controller	host	HCI_EVT	10	Rcvd Page Scan Repetition Mode Change
9	0.268999	controller	host	HCI_EVT	6	Rcvd Max Slots Change
10	0.325561	controller	host	HCI_EVT	13	Rcvd Command Complete (Link Key Request Negative Reply)
11	0.325642	host	controller	HCI_CMD	6	Sent Read Remote Supported Features
12	0.326419	controller	host	HCI_EVT	9	Rcvd IO Capability Request
13	0.343634	controller	host	HCI_EVT	7	Rcvd Command Status (Read Remote Supported Features)
14	0.343723	host	controller	HCI_CMD	13	Sent IO Capability Request Reply
15	0.344826	controller	host	HCI_EVT	14	Rcvd Read Remote Supported Features
16	0.346526	controller	host	HCI_EVT	13	Rcvd Command Complete (IO Capability Request Reply)
17	0.346621	host	controller	HCI_CMD	7	Sent Read Remote Extended Features
18	0.348161	controller	host	HCI_EVT	7	Rcvd Command Status (Read Remote Extended Features)
19	0.349539	controller	host	HCI_EVT	16	Rcvd Read Remote Extended Features Complete
20	0.349752	host	controller	HCI_CMD	14	Sent Remote Name Request
21	0.349887	localhost ()	Intel_9a:ff:a5 ()	L2CAP	15	Sent Information Request (Extended Features Mask)
22	0.355686	controller	host	HCI_EVT	7	Rcvd Command Status (Remote Name Request)
23	0.356771	controller	host	HCI_EVT	12	Rcvd IO Capability Response
24	0.390185	controller	host	HCI_EVT	258	Rcvd Remote Name Request Complete
25	0.561670	controller	host	HCI_EVT	8	Rcvd Number of Completed Packets
26	0.614777	controller	host	HCI_EVT	13	Rcvd User Confirmation Request
27	7.216310	host	controller	HCI_CMD	10	Sent User Confirmation Request Negative Reply
28	7.218845	controller	host	HCI_EVT	13	Rcvd Command Complete (User Confirmation Request Negative Reply)
29	7.219823	controller	host	HCI_EVT	10	Rcvd Simple Pairing Complete

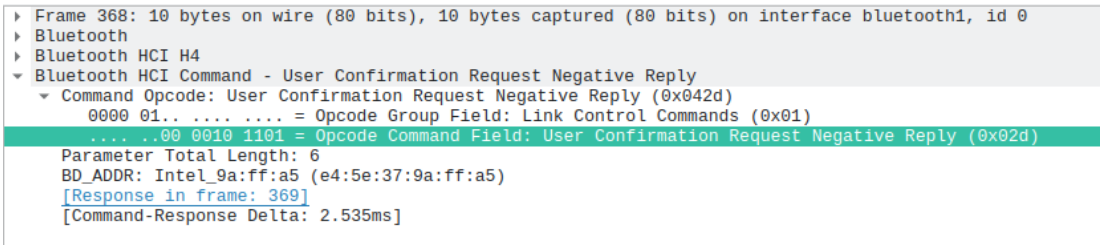
Figure 4.11: Annotated shortened Wireshark logs of a failed BIAS peripheral attack, but with the EVB-01 correctly patched and pairable.



(!(frame.time_relative <= 240.930325)) && !(frame.time_relative >= 247.553711)

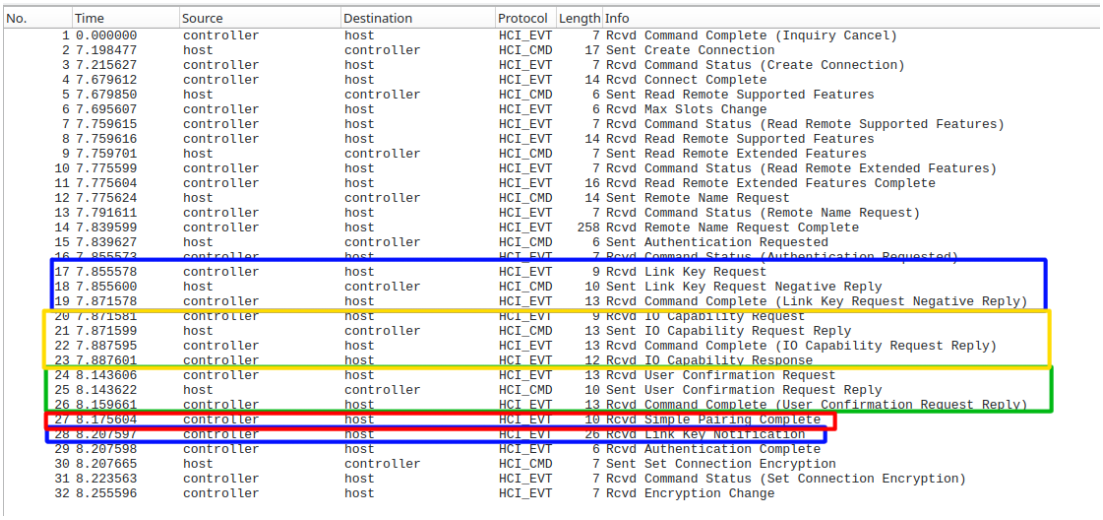
No.	Time	Source	Destination	Protocol	Length	Info
363	240.931706	controller	host	HCI_EVT	13	Rcvd User Confirmation Request
364	240.937241	37:9a:ff:a5	controller	LMP	64	LMP_Simple_Pairing_Number
365	240.942812	controller	37:9a:ff:a5	LMP	64	LMP_accepted
366	240.981334	controller	37:9a:ff:a5	LMP	64	LMP_preferred_rate
367	244.620564	controller	37:9a:ff:a5	LMP	64	LMP_set_AFH
368	247.533239	host	controller	HCI_CMD	10	Sent User Confirmation Request Negative Reply
369	247.535774	controller	host	HCI_EVT	13	Rcvd Command Complete (User Confirmation Request Negative Reply)
370	247.536752	controller	host	HCI_EVT	10	Rcvd Simple Pairing Complete
371	247.542638	controller	37:9a:ff:a5	LMP	64	LMP_numeric_comparison_failed
372	247.548176	controller	37:9a:ff:a5	LMP	64	LMP_encryption_mode_req

Figure 4.12: A closer look at the time frame in Figure 4.11 with LMP packets included.



```
▸ Frame 368: 10 bytes on wire (80 bits), 10 bytes captured (80 bits) on interface bluetooth1, id 0
▸ Bluetooth
▸ Bluetooth HCI H4
▾ Bluetooth HCI Command - User Confirmation Request Negative Reply
  ▾ Command Opcode: User Confirmation Request Negative Reply (0x042d)
      0000 01.. .... .... = Opcode Group Field: Link Control Commands (0x01)
      .... ..00 0010 1101 = Opcode Command Field: User Confirmation Request Negative Reply (0x02d)
    Parameter Total Length: 6
    BD_ADDR: Intel_9a:ff:a5 (e4:5e:37:9a:ff:a5)
    [Response in frame: 369]
    [Command-Response Delta: 2.535ms]
```

Figure 4.13: Contents of the User Confirmation Request Negative Reply packet in Figure 4.12.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	controller	host	HCI_EVT	7	Rcvd Command Complete (Inquiry Cancel)
2	7.198477	host	controller	HCI_CMD	17	Sent Create Connection
3	7.215627	controller	host	HCI_EVT	7	Rcvd Command Status (Create Connection)
4	7.679612	controller	host	HCI_EVT	14	Rcvd Connect Complete
5	7.679850	host	controller	HCI_CMD	6	Sent Read Remote Supported Features
6	7.695607	controller	host	HCI_EVT	6	Rcvd Max Slots Change
7	7.759615	controller	host	HCI_EVT	7	Rcvd Command Status (Read Remote Supported Features)
8	7.759616	controller	host	HCI_EVT	14	Rcvd Read Remote Supported Features
9	7.759701	host	controller	HCI_CMD	7	Sent Read Remote Extended Features
10	7.775599	controller	host	HCI_EVT	7	Rcvd Command Status (Read Remote Extended Features)
11	7.775604	controller	host	HCI_EVT	16	Rcvd Read Remote Extended Features Complete
12	7.775624	host	controller	HCI_CMD	14	Sent Remote Name Request
13	7.791611	controller	host	HCI_EVT	7	Rcvd Command Status (Remote Name Request)
14	7.839599	controller	host	HCI_EVT	258	Rcvd Remote Name Request Complete
15	7.839627	host	controller	HCI_CMD	6	Sent Authentication Requested
16	7.855573	controller	host	HCI_EVT	7	Rcvd Command Status (Authentication Requested)
17	7.855578	controller	host	HCI_EVT	9	Rcvd Link Key Request
18	7.855600	host	controller	HCI_CMD	10	Sent Link Key Request Negative Reply
19	7.871578	controller	host	HCI_EVT	13	Rcvd Command Complete (Link Key Request Negative Reply)
20	7.871581	controller	host	HCI_EVT	9	Rcvd IO Capability Request
21	7.871599	host	controller	HCI_CMD	13	Sent IO Capability Request Reply
22	7.887595	controller	host	HCI_EVT	13	Rcvd Command Complete (IO Capability Request Reply)
23	7.887601	controller	host	HCI_EVT	12	Rcvd IO Capability Response
24	8.143606	controller	host	HCI_EVT	13	Rcvd User Confirmation Request
25	8.143622	host	controller	HCI_CMD	10	Sent User Confirmation Request Reply
26	8.159661	controller	host	HCI_EVT	13	Rcvd Command Complete (User Confirmation Request Reply)
27	8.175604	controller	host	HCI_EVT	10	Rcvd Simple Pairing Complete
28	8.207597	controller	host	HCI_EVT	26	Rcvd Link Key Notification
29	8.207598	controller	host	HCI_EVT	6	Rcvd Authentication Complete
30	8.207665	host	controller	HCI_CMD	7	Sent Set Connection Encryption
31	8.223563	controller	host	HCI_EVT	7	Rcvd Command Status (Set Connection Encryption)
32	8.255596	controller	host	HCI_EVT	7	Rcvd Encryption Change

Figure 4.14: Annotated shortened Wireshark logs of the BIAS portion of the BLUFFS attack from Daniele. [8]

this may be to do with the baud rate we set the EVB-01 to when we first attach it in Subsection 4.3.1, but we did not look into this any further.

4.4.4 Validity of the Implementation

We discovered that the Wireshark logs of Danieli's BLUFFS attack contained the initial step where the attack device executed BIAS on the victim to connect to them. To determine if our implementation of BIAS was valid or not, we can compare our Wireshark logs to the BLUFFS logs to see if we obtain similar results. Figure 4.14 shows the relevant portions of the BLUFFS BIAS attack.

The key difference here is that the attack device receives the Link Key after Simple Pairing is finished in Danieli's logs, whereas in our logs in Figure 4.7 there is no packet containing the Link Key. In our comparisons of the two Wireshark logs, we could not find a difference in the captured packets between them that may have caused this discrepancy. The logs provided by Kozlowski also show a Link Key notification packet after the Simple Pairing procedure, but there is no explanation of why this might be. [23]

As we got the IdeaPad 5 to mistakenly pair to the EVB-01 when it was attempting to connect to its trusted Pixel 3a without the EVB-01 knowing the Link Key between them, we believe we managed to implement BIAS successfully. It may prove prudent to investigate why our implementation did not get the Link Key in future work.

4.5 Test Results

Following the steps listed out in Section 4.3, we tested BIAS against the IdeaPad 5 and Pixel 3a. We obtained impersonation files for these devices as well as the WH-CH10 headset. Table 4.1 shows which victims incorrectly connected to the EVB-01 that

Impersonated	IdeaPad 5	Pixel 3a	WH-CH510
IdeaPad 5	-	(not tested)	-
Pixel 3a	✓	-	-
WH-CH510	✓	✓	-

Table 4.1: A table showing the results of testing the BIAS peripheral attack against various devices. The left-hand column shows the devices the EVB-01 impersonated. The Pixel 3a was not tested against an impersonated IdeaPad 5 due to the EVB-01 breaking, explained in Section 5.1.

impersonated a trusted device of the victim.

We could not test BIAS against many victim devices since we only had access to our personal Bluetooth devices. Given more time and resources, we would have experimented if our implementation could perform the BIAS central attack, and we would test more everyday Bluetooth devices if they were vulnerable against BIAS.

4.6 Limitations

CVE-2020-10135 lists BIAS as having a base score of 5.4. No privileges or user interactions are required to perform BIAS, but it has a low impact on the confidentiality and integrity of the victim. [36] NVD also lists BIAS as having low attack complexity, which we argue against. Chapter 5 explains the difficulties we encountered while implementing BIAS, which drastically increases the attack complexity.

BIAS is not a realistic attack yet. After the attacker has the tooling set up, assuming they are using unmodified commercially available attack devices, they must be within a 35m range to the victim in a typical home or office environment. [37] If they do modify the device, for example by increasing the transmitting power or by attaching antennae to it to increase the range, then the device becomes more conspicuous.

The next challenge is capturing the Bluetooth packets of the device to impersonate. The minimum information the attacker must get is the device's Bluetooth address and name, as that is unique to each device. The rest of the information can be obtained beforehand if the attacker knows what device to impersonate. Without assuming that the device is discoverable, the attacker must sniff out Bluetooth packets over-the-air. The Ubertooth One hardware is specially designed for over-the-air packet sniffing [38], but it has been discontinued. [39]

Finally, BIAS is not reliable. Out of all of our attempts with our experiments, only **3** were successful. This means that the success rate for BIAS is very low. This was in a controlled setting where the attack device and the victim were next to each other, and the impersonated device was turned off. A realistic setting will have the victim and attacker be further apart, resulting in greater path loss because of the distance and obstacles between the devices. Plus, the real device the victim wishes to connect to is likely closer to the victim and is actively turned on, causing interference between the impersonated device and the attacker. We ran into this issue where the interference of

the impersonated device and the attacker confused the victim and made the victim stop responding, hence failing the BIAS attack attempt.

Chapter 5

Challenges

We list the challenges we encountered over the course of our project when implementing BIAS and BLUFFS, and we explore how these challenges may have impacted others' ability to reproduce the attacks. We also go into why BLUFFS was unachievable for this project with the resources we had.

5.1 Inaccessible Hardware

Daniele et. al listed out the equipment they used in their papers on BIAS and BLUFFS, which was a CYW920819EVB-02 evaluation board and a Linux laptop. As explained in Section 4.2, the original board was discontinued and impossible to purchase from third-party sellers, meaning we needed to find a new evaluation board to replace the EVB-02.

We learnt that the CYW920819M2-EVB-01 board had the same memory addresses as the EVB-02 from an answer to an open issue in the BIAS code repository. [7] Considering that the original BIAS paper located the specific EVB-02 memory addresses via dumping the ROM and RAM contents of the board, and then reverse engineering it in a disassembling and decompiler program called Ghidra. [24]

We did not want to go through this process ourselves if we used a different evaluation board, so we purchased the CYW920819M2-EVB-01. Any future work that looks to implement BIAS on a different evaluation board may need to go through the original process.

Unfortunately, through the process of implementing BIAS, our EVB-01's Bluetooth capabilities stopped functioning as expected. Over the course of our experimentation, we discovered that the EVB-01 could no longer sustain a Bluetooth connection once established as a peripheral. If the EVB-01 established the connection as a central, then the connection would be sustained. We could not successfully complete BIAS peripheral attacks anymore, and were unable to figure out why the EVB-01 Bluetooth's functionality stopped working. It is also due to the EVB-01 breaking that we could not implement BLUFFS, as BLUFFS requires patching the Bluetooth controller's memory using InternalBlue as well.

5.2 Incompatible Code

As mentioned in Subsection 2.2.2, the code for BIAS was made when InternalBlue was first created in Python 2, but InternalBlue is now written in Python 3. Since the BIAS code was released, there have been no updates to the code from June 2020 onwards, making it incompatible with the latest version of InternalBlue. [40] The BLUFFS main `bluffs.py` is also written in Python 2, resulting in it having the same issue.

To fix this compatibility issue, we ran `bias-template.py` and `generate.py` through Python's `2to3` command, which automatically converts Python 2 files into Python 3. We edited the output files as `2to3` did not catch the bytestrings and strings, which were incompatible with InternalBlue. All of these changes did not affect how the BIAS code worked, and so did not impact the reliability of the attack.

If we implemented BLUFFS as well, we would also change `bluffs.py` as it is written in Python 2. We would follow the same steps we did for BIAS, however `2to3` will be removed in Python version 3.13, so this method of porting the files may no longer be available in the future. [41]

5.3 Incompatible Libraries

We aimed to use the Raspberry Pi 3 as the computer for BIAS since it represents what most people that wish to reproduce BIAS have access to, and it can be purchased from online retailers for £33 at the time of writing. [42] If a user did not have a Raspberry Pi 3, but did have a newer version or a Linux computer of similar or better capabilities, they likely can implement BIAS following our explanation in Chapter 4 from Section 4.3 onwards.

As mentioned in Section 4.1, we installed Ubuntu OS on the Raspberry Pi 3 so we could install the `pwntools` library for InternalBlue. `pwntools` uses `binutils`, which in turn is not available for Raspbian OS.

5.4 Computing Power

Computing power was not an issue for BIAS as the attack does not require resource-intensive procedures. On the other hand, BLUFFS requires brute forcing an encryption key as a mandatory step in its attack.

The original BLUFFS paper relies on keeping the entropy SE of the Session Key (SK) as low as possible. In practice, $SE = 1$ if the victim device is also vulnerable to KNOB, or $SE = 7$ if the victim device is patched against KNOB. (Un)fortunately, Android pushed a patch against KNOB at the same time that the KNOB paper was published, resulting in our test victim devices being patched against the KNOB attack. [27]

This security patch forces our attack device to negotiate SK to have an entropy of $SE = 7$. With our setup of a single Raspberry Pi 3 Model B, brute forcing SK would take several weeks based on the estimations from Antonioli,[25] though it is more likely

to take at least a month considering that the Raspberry Pi 3 has far less computing power than the average modern computer today.

Considering the time frame we had for our dissertation, and how we had to implement BIAS first before we could move on to BLUFFS, it was not reasonable for us to spend the time it would take to crack SK. We also did not want to offload the calculations to distributed computing or a more powerful computer, as the aim of our dissertation is to recreate these Bluetooth vulnerabilities on a setup that could reasonably be obtained.

Chapter 6

Discussion

We discuss the implications of this dissertation and the importance of reproducible attacks. We consider the factors that made it harder or easier to reproduce BIAS and BLUFFS, and we take a step back to argue the reasons for and against increasing the reproducibility of security vulnerabilities such as BIAS and BLUFFS.

6.1 Reproducibility of BIAS

As shown by our implementation in Chapter 4, we were able to reproduce BIAS multiple times. Once we finished porting the original code to Python 3, made the impersonation files, and prepared the Raspberry Pi 3 and EVB-01, it took under 5 minutes to run each attempt of BIAS. We are confident that if we followed the steps in our implementation again, we would successfully run BIAS.

Before our work, we argue that BIAS was extremely difficult to reproduce, especially so after the original EVB-02 board was discontinued. There are multiple Github issues in the BIAS code repository with questions about obtaining the necessary information to make impersonation files, difficulties with errors, and not knowing which evaluation boards could be used for BIAS. This is due to the original authors not explaining details that were assumed to be known by those wishing to run BIAS.

With this dissertation, we provide a relatively accessible setup and a thorough explanation on every step we took to get BIAS to work for us, and provide plausible explanations why we achieved the results we got during our tests. The reproducibility of BIAS has increased because of our work.

6.2 Reproducibility of BLUFFS

Unfortunately, we cannot say we achieved the same with BLUFFS. While we have completed the first key step to the BLUFFS attack, there is no reasonable way to brute force a 7-byte encryption key with an accessible setup.

Our work does show that BLUFFS is likely reproducible on the EVB-01 if we ignore

the computing constraint, as we can follow the same steps the original paper took. We would adapt the BLUFFS code repository to Python 3 to make it compatible with the latest version of InternalBlue, but the original paper used the same setup as from the BIAS attack.

6.3 Factors of Reproducible Attacks

Our main focus during the course of creating our BIAS implementation was reproducibility and accessibility. Over the course of our work and creating our guide, we found the points we factored in most were:

- **Cost**: The cost of purchasing the necessary equipment and the time spent following our steps must be reasonable.
- **Knowledge**: People should understand the reasoning behind each step we took, and should feel confident in searching for answers if they get stuck.
- **Execution**: The end result of our guide is to run BIAS, whether that be successfully or not.

While initially each of these factors were difficult to obtain as we struggled to get BIAS to work, as we refactored our steps we strove to lower the cost and knowledge needed to follow along our explanation.

6.4 Implications

With the severe impact that BIAS and BLUFFS can have on the security and privacy of Bluetooth, it was important that we considered the potential risks and benefits our project could have on others.

Making BIAS more reproducible brings up the clear risk that malicious people could use our work to harm others. While the likelihood of this happening does partially increase because of our project, we believe that the same can be said for developers that wish to patch Bluetooth devices against BIAS. As malicious people attempt to exploit BIAS against victims, the developers can test their devices to check if they can defend against the attack.

Our project also helps to educate people that may want to understand what parts of the Bluetooth protocol may be more vulnerable to exploits, or to understand the process behind reproducing security attacks. This could be to help them figure out how to stay safe from attacks by not letting their devices be discoverable unless necessary, or to ensure that future proof of concepts of security vulnerabilities are as easy to reproduce as possible.

Chapter 7

Conclusions

7.1 Summary

We implemented BIAS for the CYW920819M2EVB-01 evaluation board and Raspberry Pi 3 Model B. This dissertation is the first documented implementation of BIAS for an evaluation board other than the CYW920819EVB-02 and the CYW920735Q60EVB-01. We provide the first analysis of the packets captured for a BIAS attack, explaining why the attack works.

Our work has been recognised by the original author, and it is planned to be added to an open-source Bluetooth testing framework developed by members at ETH Zürich.

7.2 Future Work

- Implement BIAS for more Bluetooth chips. The main challenge is finding the correct patches of memory in the RAM to overwrite in InternalBlue, as this will require low-level reverse engineering of proprietary firmware. It is plausible that there are other Cypress chips that have the same memory addresses as our attack device. Collecting a list of evaluation boards that can run BIAS will be helpful in the future when the CYW920819M2EVB-01 is discontinued, and the same issue that motivated us to do this dissertation happens again.
- Test more devices against BIAS. We only had access to 3 Bluetooth devices, of which 2 could initiate Bluetooth connections. Testing devices that were released in the past couple years would prove to be a good investigation as to whether commercial devices are truly safe against BIAS or not. While the Bluetooth SIG provides patches against these Bluetooth attacks, they may not be mandatory because all versions of Bluetooth must be backwards compatible.
- Develop BIAS code that is more consistent, or understand why the BIAS attack fails. While we uncovered the reason some of our BIAS attempts failed, we do not know how to resolve them. This could be done alongside testing more devices against BIAS, since giving BIAS a higher success rate would make it easier to

test.

- Implement BLUFFS. This likely is not possible on the budget-friendly setup we aimed for, but there are still no other implementations of BLUFFS other than the original paper. We stress the importance of reproducing exploits, especially for one as severe as BLUFFS, but we do not have the resources to test BLUFFS.

Bibliography

[1] en-US. [Online]. Available: https://www.bluetooth.com/2024-market-update/.

[2] B. SIG, *Reporting security vulnerabilities*, en-US. [Online]. Available: https://www.bluetooth.com/learn-about-bluetooth/key-attributes/bluetooth-security/reporting-security/.

[3] [Online]. Available: https://www.cve.org/About/Metrics.

[4] D. Mu, A. Cuevas, L. Yang, *et al.*, "Understanding the reproducibility of crowd-reported security vulnerabilities," en,

[5] en-US. [Online]. Available: https://www.bluetooth.com/learn-about-bluetooth/key-attributes/bluetooth-security/bias-vulnerability/.

[6] en-US. [Online]. Available: https://www.bluetooth.com/learn-about-bluetooth/key-attributes/bluetooth-security/bluffs-vulnerability/.

[7] en. [Online]. Available: https://github.com/francozappa/bias.

[8] D. Antonioli, *Francozappa/bluffs*. [Online]. Available: https://github.com/francozappa/bluffs/tree/main/device.

[9] I. T. AG, *Cyw920819evb-02 - infineon technologies*, en. [Online]. Available: https://www.infineon.com/cms/en/product/evaluation-boards/cyw920819evb-02/.

[10] S. Bluetooth, *Part a architecture*. [Online]. Available: https://www.bluetooth.com/wp-content/uploads/Files/Specification/HTML/Core-54/out/en/architecture,-mixing,-and-conventions/architecture.html#UUID-18c95f05-5e3b-74ff-5ee8-66505f1f53e6.

[11] S. Bluetooth, *Part b baseband specification*. [Online]. Available: https://www.bluetooth.com/wp-content/uploads/Files/Specification/HTML/Core-54/out/en/br-edr-controller/baseband-specification.html.

[12] S. Bluetooth, *Part h security specification*. [Online]. Available: https://www.bluetooth.com/wp-content/uploads/Files/Specification/HTML/Core-54/out/en/br-edr-controller/security-specification.html.

[13] M. Von Tschirschnitz, L. Peuckert, F. Franzen, and J. Grossklags, "Method confusion attack on bluetooth pairing," en, in *2021 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA: IEEE, May 2021, pp. 1332–1347, ISBN: 978-1-72818-934-5. DOI: 10.1109/SP40001.2021.00013. [Online]. Available: https://ieeexplore.ieee.org/document/9519477/.

[14] S. Bluetooth, *Part c link manager protocol specification*. [Online]. Available: https://www.bluetooth.com/wp-content/uploads/Files/Specification/

HTML/Core-54/out/en/br-edr-controller/link-manager-protocol-specification.html.

[15] P. Dziwior, *Bluetooth - lmp*. [Online]. Available: http://www.dziwior.org/Bluetooth/LMP.html.

[16] Ubuntu, *Bluez commands*, en. [Online]. Available: https://ubuntu.com/core/docs/bluez/reference/commands.

[17] D. Mantz, "Internalblue–a bluetooth experimentation framework based on mobile device reverse engineering," 2018.

[18] Wireshark, *Chapter 1. introduction*. [Online]. Available: https://www.wireshark.org/docs/wsug_html_chunked/ChapterIntroduction.html.

[19] J. Classen, *Seemoo-lab/h4bcm$_w$ireshark$_d$issector*, C, Sep. 2023. [Online]. Available: https://github.com/seemoo-lab/h4bcm_wireshark_dissector.

[20] M. Ai, K. Xue, B. Luo, *et al.*, "Blacktooth: Breaking through the defense of bluetooth in silence," en, in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, Los Angeles CA USA: ACM, Nov. 2022, pp. 55–68, ISBN: 978-1-4503-9450-5. DOI: 10.1145/3548606.3560668. [Online]. Available: https://dl.acm.org/doi/10.1145/3548606.3560668.

[21] M. E. Garbelini, V. Bedi, S. Chattopadhyay, S. Sun, and E. Kurniawan, "Braktooth: Causing havoc on bluetooth link manager via directed fuzzing," en,

[22] J. Wu, Y. Nan, and V. Kumar, "Blesa: Spoofing attacks against reconnections in bluetooth low energy," en,

[23] M. Kozlowski, *Marcinguy/cve-2020-10135-bias*, Sep. 2023. [Online]. Available: https://github.com/marcinguy/CVE-2020-10135-BIAS.

[24] D. Antonioli, N. O. Tippenhauer, and K. Rasmussen, "Bias: Bluetooth impersonation attacks," en, in *2020 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA: IEEE, May 2020, pp. 549–562, ISBN: 978-1-72813-497-0. DOI: 10.1109/SP40000.2020.00093. [Online]. Available: https://ieeexplore.ieee.org/document/9152758/.

[25] D. Antonioli, "Bluffs: Bluetooth forward and future secrecy attacks and defenses," en, in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, Copenhagen Denmark: ACM, Nov. 2023, pp. 636–650, ISBN: 9798400700507. DOI: 10.1145/3576915.3623066. [Online]. Available: https://dl.acm.org/doi/10.1145/3576915.3623066.

[26] D. Antonioli, N. O. Tippenhauer, and K. Rasmussen, "The knob is broken: Exploiting low entropy in the encryption key negotiation of bluetooth br/edr," en,

[27] en. [Online]. Available: https://source.android.com/docs/security/bulletin/2019-08-01.

[28] Python, Apr. 2024. [Online]. Available: https://github.com/Gallopsled/pwntools.

[29] J. Classen, *Bluez: Linux bluetooth stack overview*. [Online]. Available: https://naehrdine.blogspot.com/2021/03/bluez-linux-bluetooth-stack-overview.html.

[30] D. Antonioli, *Bias/linux-4.14.111 at master · francozappa/bias*. [Online]. Available: https://github.com/francozappa/bias/tree/master/linux-4.14.111.

[31] M. Holtmann, *Android common kernel h4,ecv.h first commit*. [Online]. Available: https://android.googlesource.com/kernel/common/+/07eb96a5a7b083c988a2c7b06

[32] en. [Online]. Available: https://github.com/BPI-SINOVOIP/BPI-M4-bsp/issues/4.

[33] J. Kicinski, *Re: [patch 00/22] add support for clang lto - jakub kicinski*. [Online]. Available: https://lore.kernel.org/kernel-hardening/20200707095651.422f0b22@kicinski-fedora-pc1c0hjn.dhcp.thefacebook.com/.

[34] BitManipulator, *Answer to "attempting to compile kernel yields a certification error"*, Apr. 2021. [Online]. Available: https://unix.stackexchange.com/a/646758.

[35] [Online]. Available: https://man.archlinux.org/man/btattach.1.en.

[36] [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2020-10135.

[37] en-US. [Online]. Available: https://www.bluetooth.com/learn-about-bluetooth/key-attributes/range/.

[38] lisaparty, *Ubertooth one — ubertooth documentation*. [Online]. Available: https://ubertooth.readthedocs.io/en/latest/ubertooth_one.html.

[39] Straithe and Elizabeth, *Ubertooth retirement - great scott gadgets*, Dec. 2022. [Online]. Available: https://greatscottgadgets.com/2022/12-22-ubertooth-retirement/.

[40] D. Antonioli, *Commits francozappa/bias*. [Online]. Available: https://github.com/francozappa/bias/commits/master/.

[41] en. [Online]. Available: https://docs.python.org/3/library/2to3.html.

[42] en. [Online]. Available: https://www.rapidonline.com/raspberry-pi-3-model-b-1-quad-core-1-4ghz-1gb-ram-wifi-bluetooth-75-1005.