Brno University of Technology
Faculty of Information Technology

Project - BZA

Implementation of Bluetooth Impersonation AttackS (BIAS)

Firc Anton (xfirca00)

1 Introduction

An article published in May 2020, presented new attacks to be performed on Bluetooth devices. The attacks exploit a vulnerabilities in legacy authentication procedure and secure authentication procedure that allow impersonation attacks [1]. The goal of this work is to implement attacks described in [1] and prove their technical feasibility.

The main contributions of this work might be summarized as follows:

- Technical feasibility of BIAS was evaluated, and the BIAS toolkit provided by authors of [1] simplifies the execution, so even less experienced users are capable to reproduce it.
- I present that the most demanding part of the BIAS is constructing correct impersonation file.
- I present a modification to InternalBlue source codes needed to run the framework with latest CYW920819EVB-02 development board.
- An tutorial for execution was presented for other people looking into reproducing BIAS or further researching this topic.

The principle of BIAS is discussed in section 2. The BIAS toolkit for automated attacks execution is discussed in section 3. Section 4 presents the hardware used to execute BIAS. Step-by-step description on how the attack was executed is presented in section 5. The retrieval of impersonation file content, essential for successful attack implementation, is discussed in section 6. Finally section 7 concludes all of the findings of this work.

2 Essence of BIAS

BIAS represents a set of attacks on Legacy and Secure authentication. All of the attacks exploit overly permissive master role [1]. Following sections describe each of the possible attacks.

2.1 Attacks on Legacy Secure connections

The master sends an authentication challenge, slave responds with a message calculated using the agreed long-term key [1]. Master does not need to possess the long-term key to authenticate slave‘s response as the Bluetooth standard does not require to use the authentication procedure mutually [1]. This way, master is able to be connected to any slave device without knowing the long-term key.

The contrary scenario puts the attacking device into a role of a slave. The slave needs to possess the long-term key to respond to master's authentication challenge [1]. However, Bluetooth's role switch procedure might be misused. The attacker initiates the role swap procedure, which ensures him a master role before the authentication procedure [1]. The authentication procedure than follows the same scenario as discussed before. This is possible, because every standard compliant device must agree on the role switching in any state of the connection, even before authentication [1].

2.2 Downgrade attacks on Secure connections

Secure connections use stronger cryptographic primitives than Legacy Secure connections, and are currently considered the most secure way to pair and establish

secure connections [1]. Unfortunately, all the tested devices were vulnerable to the downgrade attack [1]. While the secure connections require mutual authentication procedure, the Bluetooth standard allows the devices to switch to the less secure Legacy Secure connections even when both of the devices support safer Secure connections or were paired using Secure connections [1]. After the downgrade to Legacy Secure connections, the attack follows the scenario described before in section 2.1.

2.3 Reflection attacks on Secure connections

The reflection attack works on a different base as the ones described before. In the reflection attack, the attacker tricks victim into answering his own authentication challenge and giving response to the attacker [1]. The attacker then authenticates by sending the response back to the victim [1]. The standard again does not provide any mean to prevent this attack, however this type of attack is not implemented within [1].

3 BIAS toolkit

Apart from the technical specification of the impersonation attacks, [2] provides even an automated toolkit for implementing the attacks on Legacy secure connections and downgrade attacks on secure connections.

The toolkit contains a modified Bluetooth firmware for CYW920819EVB-02 development board, enabling to modify the authentication procedures, thus to enable the role switching and authenticating slave automatically [1]. The authors retrieved addresses for patching the ROM with custom firmware using reverse engineering [1].

After collecting enough information from the reverse engineering process, the authors developed the `BIAS` toolkit that automates the attacks and published it on GitHub[1] [1]. The toolkit takes impersonation and attack file as input, generates the impersonation script and finally configures the attack device to impersonate the target device [1].

The *attack file* contains information about the attack device (dev-board), such as the name of `hci` interface, and the addresses of the functions that are needed to be patched [1]. The *impersonation file* contains info about the device we want to impersonate, such as Bluetooth name and address and Secure connections support [1].

4 Used hardware

To implement BIAS, a Linux laptop and a CYW920819EVB-02 development board from Cypress is needed [1]. The board itself includes a CYW20819 SoC, that implements Bluetooth 5.0 and support for Secure Connections [1]. The development board is connected to the host (laptop in our case) via USB using the UART interface [1].

The Linux laptop used to perform these attacks is a Lenovo IdeaPad 700-15ISK with Intel Core I7–6700HQ CPU in a default configuration.

[1] https://github.com/francozappa/bias

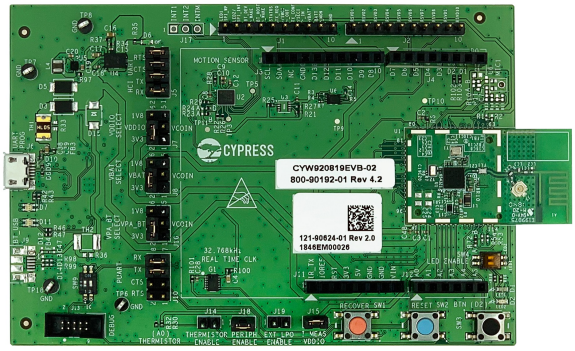

Figure 1: The CYW920819EVB-02 development board. The CYW20819 SoC is visible in the white-dashed region [1]. Retrieved from: https://www.cypress.com/documentation/development-kitsboards/cyw920819evb-02-evaluation-kit

5 Implementing the attack

Attack implementation follows instructions from the GitHub repository [2] of the `BIAS` toolkit. The whole process might be generalized into following steps:

1. Patching the Linux kernel to parse board diagnostic messages
2. setting up the board (controller) with the laptop (host)
3. starting InternalBlue and monitoring the Bluetooth interface
4. patching the ROM of the dev-board and impersonating a Bluetooth device

5.1 Kernel patching

Patching the Linux kernel to enable diagnostic message parsing is the first important step in performing BIAS. The GitHub repository comes with modified Bluetooth drivers for the vanilla Linux 4.14.111[2].

As a starting system to patch, I chose Ubuntu 18.04.1. After setting up the system, I downloaded the target Linux kernel, and merged the modified Bluetooth drivers with the downloaded kernel files. In order to get functioning system, a configuration file is needed to set the modules and other properties of kernel [3]. The configuration file may be manually created, or copied from the running Ubuntu distribution by running `cp -v /boot/config-$(uname -r) .config` in the downloaded kernel directory [3].

After copying the configuration file and modified drivers, kernel can be compiled and installed running following commands:

```
make -j $(nproc)
sudo make modules_install
sudo make install
```

The first command compiles the kernel using all available processors to speed up the process, the second command compiles all of the needed modules, and the last known finally installs the patched kernel [3]. When running the first command, you

[2]Available from: http://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.14.111.tar.xz

might be questioned about additional kernel settings, applying the default ones (pressing enter for each questions) worked well in my case. After successful installation, the system needs to be rebooted.

Please note that if you patched the kernel the same way, that the newer one is still available, the computer will try to boot the newer one. It is needed to select the Ubuntu with 4.14.111 Linux to boot in Grub menu.

5.2 Setting up the board

With the kernel patched, the board has to be configured as a `hci` device. In my case following commands set up the correct baud rate and connected the dev-board as a `hci` device [2, 4]:

```
sudo stty -F /dev/ttyUSBX 115200
sudo btattach -B /dev/ttyUSBX
```

To determine the correct `/dev/USB` mountpoint a script from StackExchange issue might be used[3].

To verify that the device is set-up correctly `hciconfig` might be executed. The dev-board should be present, and have MAC address assigned as shown in Figure 2.

```
anton@anton-Lenovo-ideapad-700-15ISK:~$ hciconfig
hci1:   Type: Primary  Bus: UART
        BD Address: 20:81:9A:10:00:00  ACL MTU: 704:20  SCO MTU: 64:10
        UP RUNNING PSCAN ISCAN
        RX bytes:869 acl:0 sco:0 events:61 errors:0
        TX bytes:3286 acl:0 sco:0 commands:61 errors:0

hci0:   Type: Primary  Bus: USB
        BD Address: AC:2B:6E:BD:6E:9B  ACL MTU: 1021:5  SCO MTU: 96:6
        DOWN
        RX bytes:1589 acl:0 sco:0 events:197 errors:0
        TX bytes:37771 acl:0 sco:0 commands:196 errors:0
```

Figure 2: Correct output of `hciconfig` command. Device `hci1` is the dev-board, device `hci0` is the notebook wireless chip.

5.3 Using InternalBlue to monitor the interface

After connecting the dev-board as a `hci` device, InternalBlue CLI might be used to monitor the selected interface and send packets. In order to use InternalBlue to execute BIAS, a python2 version is needed[4]. The CYW920819EVB-02 board recieved a firmware update, so modification to the InternalBlue source code might be needed. As shown in Figure 3, the board chip is identified as `0x2305`, and this version is unknown to the InternalBlue tool. This issue might be resolved by modifying the InternalBlue source code, where the `fw_0x220c.py` file content is copied into a new file named `fw_0x2305.py`. These files are located in `<python_dir>/site-packages/internalblue/fw`.

[3] https://unix.stackexchange.com/questions/144029/command-to-determine-ports-of-a-device-like-dev-ttyusb0

[4] https://github.com/seemoo-lab/internalblue/releases/tag/python2

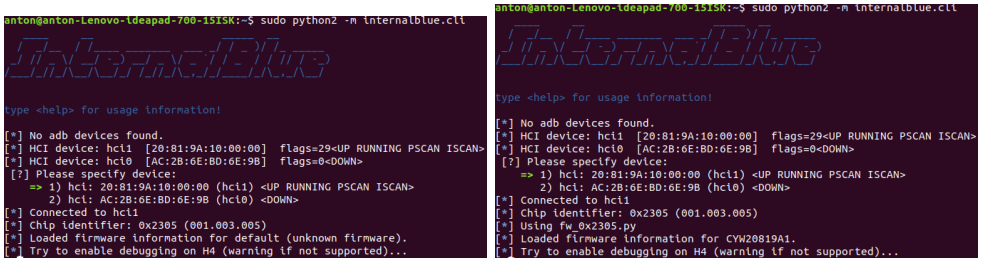

Figure 3: A screenshot from InternalBlue CLI before modifying the source code to match the new firmware version on the left, and a screenshot from the patched version.

Finally, to enable dev-board's diagnostic mode run `sudo python2 enable_diag.py`, that send a special LMP packet to the board. Before running this script, it is needed to set the correct `hci` index on line 4. This serves also as a check if the kernel was correctly patched, as this script will fail without the patch. To monitor the interface using wireshark, a `monitor start` command has to be executed from within the InternalBlue CLI.

5.4 Patching ROM and impersonation

The `BIAS` toolkit comes with two scripts prepared for patching the ROM and finally impersonating the target device. Firstly, `generate.py` script generates `bias.py` script that contains commands for InternalBlue to patch the ROM and impersonate the target device. The generate script uses two other JSON files: attack and impersonation file discussed in section 3. The generation of the `bias.py` script is just copying the template script and echoing correct values inside.

The `bias.py` handles all of the needed impersonation functionality. At first, ROM of the dev-board is patched with custom Bluetooth firmware to override authentication and secure connections establishment procedures [1]. The process of ROM patching is just writing custom assembly code into specific addresses of the board using proprietary HCI commands [1].

Next, the info about impersonated device might be loaded into board's memory. This process runs as a second part of the `bias.py` script.

After finishing the script, the dev-board is visible as the impersonated device. The repository contains some prepared impersonation files, but to be really able to execute this attack own impersonation file has to be created, this process is further described in section 6.

5.5 Connecting to victim device

After finishing the impersonation, board is ready to be connected to the victim device. As a victim device I chose an ASUS X509JB-EJ078 netbook, and as the impersonated device SOUNDPEATS Truengine 2 earbuds. Firstly, I paired the impersonated device with the victim device. Then, I constructed the impersonation file as shown in Figure 4.

After loading the contents of the impersonation file into the dev-board, I was able to connect to the victim device as shown in Figure 5. Connection was also possible from the victim device to the dev-board, however this connection dropped every time just after creation.

```
{
    "if": "SOUNDPEATS_TRUENGINE2_IF.json",
    "lmin": "07",
    "lmax": "07",
    "btadd": "00:14:BE:60:98:23",
    "btname": "SOUNDPEATS Truengine 2 L",
    "lmpver": "9",
    "companyid": "10",
    "subversion": "3676",
    "lmpfp0": "fffe8ffedbff5b87",
    "lmpfp1": "0300000000000000",
    "lmpfp2": "0f03000000000000",
    "iocaps": "01",
    "authreq": "03",
    "oobdata": "0",
    "deviceclass": "040424"
}
```


Figure 4: Impersonation file contents for SOUNDPEATS Truengine 2 earbuds.

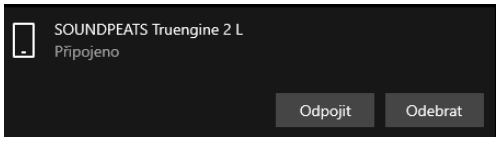

Figure 5: Established connection between victim device and dev-board.

Even though the connection was successful, it did not exactly impersonate chosen device. The dev-board should be visible as audio device, which is indicated by a different state notification as shown in Figure 6.

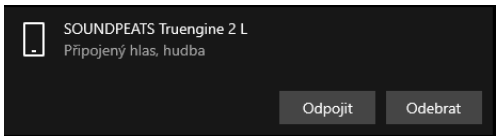

Figure 6: Genuine connection between impersonated device and victim device.

This seems to be caused by incorrect parameters setting in the impersonation file. The parameters were set to match the ones of the impersonated device, however they did not work as expected. Even experimenting with another combinations, different from the retrieved one, did not bring the expected results.

6 Creating impersonation file

The impersonation file contains the information about impersonated device:

- MAC address
- Bluetooth name
- LMP version and subversion
- Company ID
- LMP feature pages 0-2

- `iocaps`
- `authreq`
- `oobdata`
- `deviceclass`

Majority of these parameters can be retrieved from the output of command `sudo hcitool info MAC_ADDR`, as Figure 7 shows [5]. The `iocaps, authreq and oobdata` are not present in this output, but can be retrieved from captured wireshark pairing packet capture into a JSON and searching in it as shown in Figure 8 [5]. The device class might be retrieved from the device itself, or might be calculated using this tool: `https://www.ampedrftech.com/cod.htm`.

```
anton@anton-Lenovo-Ideapad-700-15ISK:~$ sudo hcitool info FC:66:CF:2D:6E:3E
Requesting information ...
        BD Address:  FC:66:CF:2D:6E:3E
        Device Name: iPhone
        LMP Version:  (0xb) LMP Subversion: 0x6308
        Manufacturer: Broadcom Corporation (15)
        Features page 0: 0xbf 0xfe 0xcf 0xfe 0xdb 0xff 0x7b 0x87
                <3-slot packets> <5-slot packets> <encryption> <slot offset>
                <timing accuracy> <role switch> <sniff mode> <RSSI>
                <channel quality> <SCO link> <HV2 packets> <HV3 packets>
                <u-law log> <A-law log> <CVSD> <paging scheme> <power control>
                <transparent SCO> <broadcast encrypt> <EDR ACL 2 Mbps>
                <EDR ACL 3 Mbps> <enhanced iscan> <interlaced iscan>
                <interlaced pscan> <inquiry with RSSI> <extended SCO>
                <EV4 packets> <EV5 packets> <AFH cap. slave>
                <AFH class. slave> <LE support> <3-slot EDR ACL>
                <5-slot EDR ACL> <sniff subrating> <pause encryption>
                <AFH cap. master> <AFH class. master> <EDR eSCO 2 Mbps>
                <EDR eSCO 3 Mbps> <3-slot EDR eSCO> <extended inquiry>
                <LE and BR/EDR> <simple pairing> <encapsulated PDU>
                <err. data report> <non-flush flag> <LSTO> <inquiry TX power>
                <EPC> <extended features>
        Features page 1: 0x0f 0x00 0x00 0x00 0x00 0x00 0x00 0x00
        Features page 2: 0x7f 0x0f 0x00 0x00 0x00 0x00 0x00 0x00
```

Figure 7: Output of the `hcitool info` command.

```
"bthci_cmd.io_capability": "1",
"bthci_cmd.oob_data_present": "0",
"bthci_cmd.auth_requirements": "3",
```

Figure 8: `iocaps, authreq` and `oobdata` values in the wireshark JSON packet capture.

7 Conclusions

During this experiment, I was able to impersonate a Bluetooth earbuds and establish connection with a victim device. However the dev-board did not exactly impersonate the earbuds, as a different type of connection was shown on the victim device after connection establishment.

This experiment shows that impersonating a Bluetooth device is possible even with minimal effort, as the automated BIAS toolkit provides everything necessary. The most demanding part is creating correct impersonation file. As shown in this experiment, the impersonation file is the crucial element deciding whether the dev-board is able to connect to the victim device without the end user noticing.

Finally, this paper might serve as a tutorial for anyone willing to further explore the Bluetooth Impersonation AttackS.

References

[1] D. Antonioli, N. O. Tippenhauer, and K. Rasmussen. Bias: Bluetooth impersonation attacks. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 549–562, 2020.

[2] Daniele Antonioli. Bias. https://github.com/francozappa/bias, 2020.

[3] Vivek Gite. How to compile and install linux kernel 5.6.9 from source code. https://www.cyberciti.biz/tips/compiling-linux-kernel-26.html, 2020.

[4] Secure Mobile Networking Lab. Internalblue. https://github.com/seemoo-lab/internalblue, 2020.

[5] Marcin Kozlowski. Cve-2020-10135-bias. https://github.com/marcinguy/CVE-2020-10135-BIAS, 2020.