

EXPLIoT IoT Security Learning Kit

Lab Manual

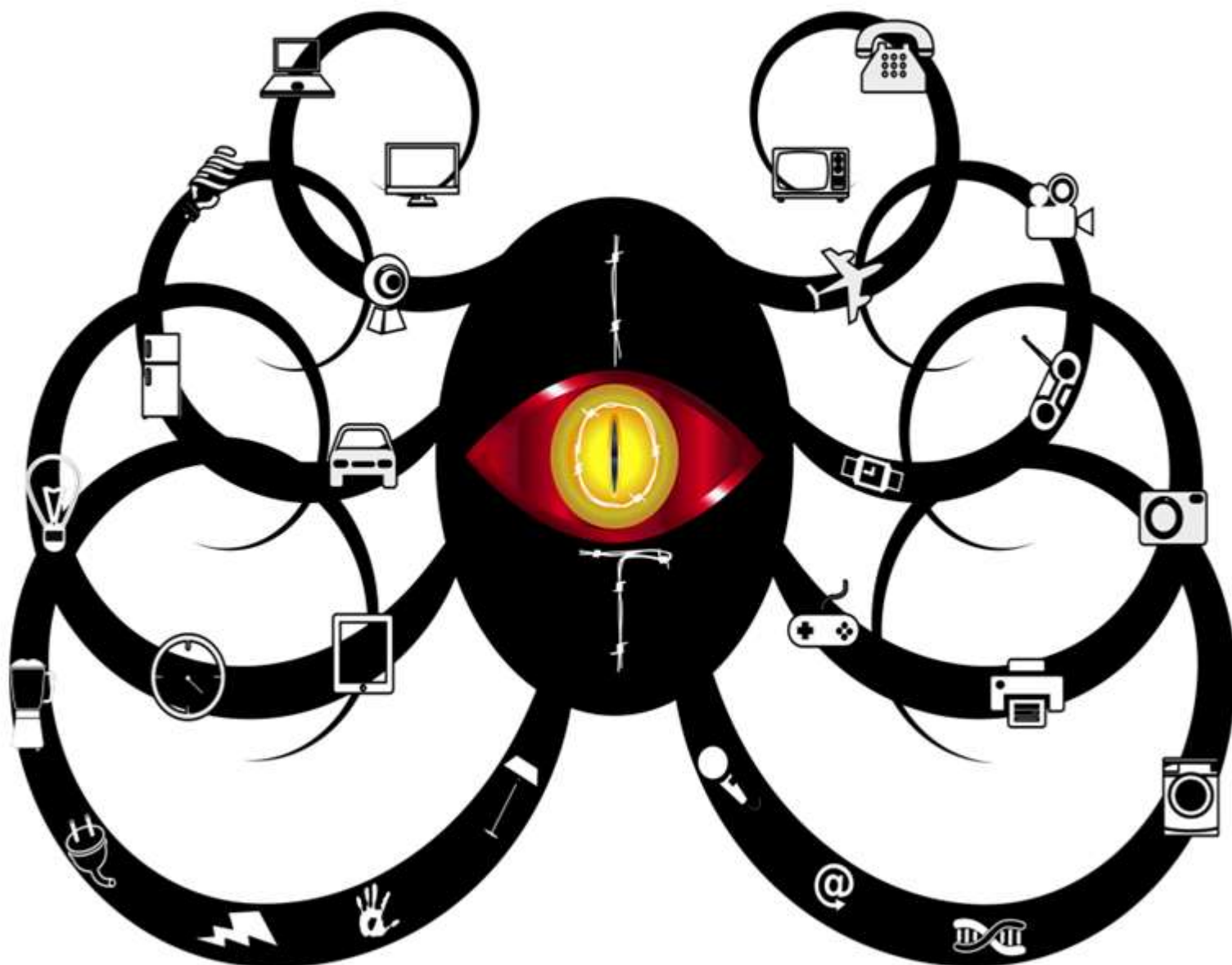




Table of Contents

Introduction

About EXPLIoT

Labs

UART

[Lab 1 – UART Identification Method 1](#)

[Lab 2 – UART Identification Method 2](#)

[Lab 3 – UART Communication with Target](#)

JTAG

[Lab 1 – JTAG Identification Method 1](#)

[Lab 2 – JTAG Identification Method 2](#)

[Lab 3 - Microcontroller firmware extraction over JTAG](#)

[Lab 4 - Microcontroller firmware patching over JTAG](#)

[Lab 5 - Microcontroller peripheral access over JTAG](#)

I2C

[Lab 1 – I2C chip reconnaissance](#)

[Lab 2 – I2C communication sniffing to bypass authentication](#)

[Lab 3 – I2C chip memory dumping](#)

SPI

[Lab 1 – SPI chip recon](#)

[Lab 2 – SPI communication sniffing to bypass authentication](#)

[Lab 3 – SPI chip memory dumping and firmware extraction](#)

BLE

[BLE Labs – DIVA Board](#)

[Lab 1 – Recon](#)

[Lab 2 – GATT Services and Characteristics enumeration](#)

[Lab 3 – Attack on BLE Handle](#)

[BLE Labs – Smart Watch](#)

[Lab 1 – Recon](#)

[Lab 2 – GATT Services and Characteristics enumeration](#)

[Lab 3 – Sniffing GATT protocol communication \(Android\)](#)

[Lab 4 – Analyze GATT Protocol communication](#)

[Lab 5 – Controlling the Device](#)

[Lab 6 – Crack BLE LTK](#)

ZigBee

[ZigBee\(802.15.4\) With One DIVA](#)

[Pre-requisite - Set-up 1 DIVA Module for ZigBee.](#)

[Lab 1 – ZigBee Network Discovery](#)

[Lab 2 – Eavesdropping](#)

[Lab 3 – Replay Attack](#)

[ZigBee\(802.15.4\) With Two DIVA](#)

[Pre-requisite - Set-up 2 DIVA Module for ZigBee.](#)

[Lab 1 – Network Discovery](#)

[Lab 2 – Eavesdropping](#)

[Lab 3 – Replay Attack](#)

[Lab 4 – Decrypt ZigBee Communication](#)

Introduction:

The EXPLIoT IoT Security learning kit is a great way to get started with IoT device security and exploitation using the DIVA (Damn Insecure and Vulnerable Application) board and other targets. This kit includes everything that you need in your arsenal to perform UART, I2C, SPI, JTAG, ZigBee, BLE analysis. This Lab Manual provides guidance and step by step process of performing each lab. It will teach you everything from finding the communication ports, sniffing, manipulating communication to exploiting IoT devices.

What is included:



1. DIVA (Damn Insecure and Vulnerable Application) IoT Board
2. EXPLIoT Nano
3. Zigbee Auditor
4. Arduino Nano
5. Vulnerable Smart watch with charging cable
6. Logic Analyzer
7. SPI Flash (Winbond)
8. Digital Multimeter
9. Bread Board + Jumper Wires (M-M, M-F, F-F)
10. USB-Mini USB, USB-Micro USB Cable

Note:

There are some files provided (in respective Lab directories) along with the lab manual. Use the files wherever required as mentioned in the lab itself.

About EXPLIoT

EXPLIoT is an open-source framework designed for security testing and exploiting IoT. It is an initiative by Payatu, a research-powered cybersecurity services company with expertise in IoT, embedded, mobile security and vulnerability research. More details – <https://payatu.com>.

We also organize one of the largest information security conferences in Asia called nullcon – <https://nullcon.net/> and one of the most active Hardware Security conferences in Europe called hardware.io – <http://hardware.io>.

For more details about EXPLIoT and our products, visit <https://explot.io/>

EXPLIoT Offerings:

1. **IoT Auditor** – Automated Assessments for Hardware, Firmware, and RF Interfaces, all in a single, seamless platform. For more details - <https://explot.io/platform/>
2. **Online Store** – Our online store where you can purchase hardware tools for IoT Security. More details - <https://explot.io/collections/frontpage>
3. **EXPLIoT Academy** – Boost your confidence, master the field, become a certified professional with our expert-led courses
<https://academy.explot.io/home>
4. **EXPLIoT Framework** – An open source IoT Security Testing and Exploitation Framework. More details - https://gitlab.com/explot_framework/explot

Labs

UART

Lab 1 – UART Identification Method 1

Objective:

UART Identification Method 1 using micro-controller pins and DMM (Digital Multimeter) conductivity test. The goal is to identify UART pinouts on the PCB board, provided we know of micro-controller UART pins, which can be found in the datasheet of the micro-controller.

Description:

In this lab, we will understand how to read the datasheet of a micro-controller to find out the pinouts and carry out conductivity tests using DMM. We will learn to read the pin diagram from the datasheet, mapping pinouts from the micro-controller to the board pinouts using DMM conductivity test and identify UART ports.

Download the STM32WB55RG datasheet. (also provided along with the manual)

(datasheet - <http://www.st.com/resource/en/datasheet/stm32wb55re.pdf>)

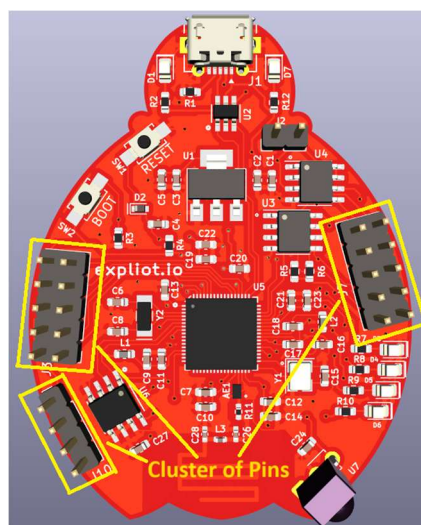
On the actual microcontroller IC, pin 1 starts with a small round index mark. Pins are calculated in an anti-clockwise direction from the mark.

Setup:

1. DMM (Digital Multimeter)
2. DIVA IoT Board (**DIVA Board must be powered off**)

Steps:

1. Try to locate a cluster of pinouts/headers. After shortlisting them, number them for convenience.



- Identify the name printed on the microcontroller and do a quick Google search for its correct datasheet. Identify the correct package by counting the number of pins on it and their shape on the microcontroller while comparing it with the datasheet (page 57 of STM32WB55RGV6 datasheet). The package used in DIVA is VFQFPN68.

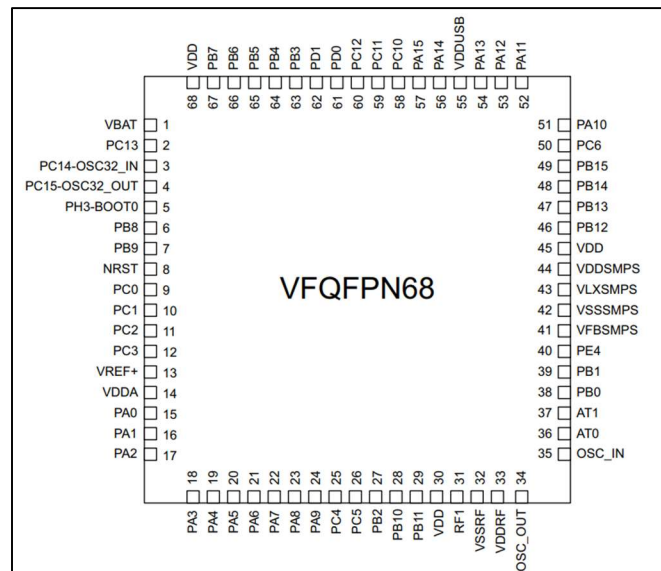


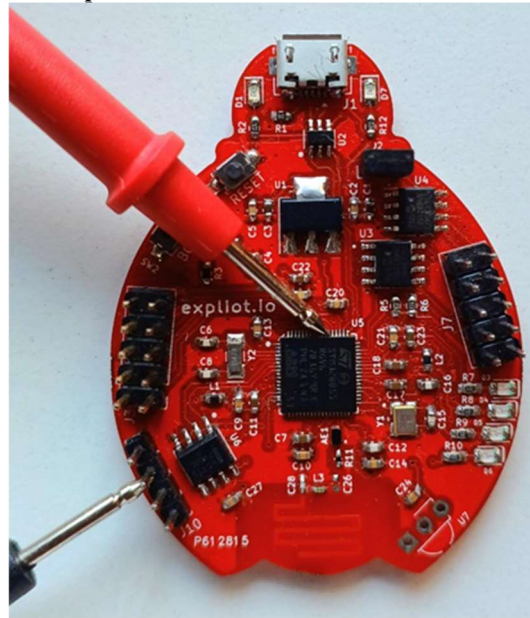
Fig. – STM Micro-controller package used in DIVA

- Go to the pin definition table in the datasheet (starting at page no. 60) and locate the UART pins for the package VFQFPN68 (pin numbers of USART TX, USART RX). Please note that there can be more than one UART port on the micro-controller.
- Power ON the DMM using the Power button (press and hold power button for 2 seconds). You will see the below image (---). This function means your DMM is in continuity mode (Same mode is used for measuring the AC/DC voltage and resistance). **Please refer to the DMM Manual for more information.**
- The DIVA (Target board) must be powered off while performing the continuity test. Otherwise, it will cause damage to the board, so be careful.



6. Touch one DMM (either red or black) probe on the target board pinout/header which is to be identified and another probe on the located UART pin of micro-controller.
If the DMM beeps, it means there is conductivity between those two points. This indicates that the pinout on the board is directly connected to the corresponding micro-controller UART pin. So, if you are testing for TX pin on the microcontroller and one of the pin headers on the board and the DMM beep, voila!

7. You have found the TX pinout.



8. Repeat the same process to identify the RX pin.

Lab 2 – UART Identification Method 2

Objective:

UART Identification Method 2 using DMM Voltage tests on the board pinouts - The objective is to identify UART pinouts on the PCB board when we do not have the knowledge of the micro-controller UART pins and the datasheet of the micro-controller.

Setup: DIVA IoT board

Steps:

1. Power up the DIVA board by connecting it to your laptop (using the micro-USB cable provided).
2. DIVA appears as a virtual-serial (USB-cdc/virtual-com) port by default; the task is to find out the virtual-serial port of DIVA. In Windows, virtual-serial ports are called COM ports. In Linux, they appear as files prepended with "tty" to their names. E.g./dev/ttyACM0. Enter the command **lsusb** in the terminal to list all the connected USB devices. You will get the below output.

```
dinesh@dinesh-VirtualBox:~$ lsusb
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 007: ID 0483:5740 STMicroelectronics Virtual COM Port
Bus 001 Device 002: ID 80ee:0021 VirtualBox USB Tablet
Bus 001 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
```

3. If you see STMicroelectronics Virtual COM Port in the output, you are good to go. If not, check the connections, try USB connect-disconnect, and check if the USB ports are properly forwarded. Enter command **dmesg | tail** in the terminal to figure out the assigned USB-serial name.

We can see the device **"ttyACM0"**

```
[18695.067753] usb 1-2: New full-speed USB device number 7 using ohci-pci
[18695.468296] usb 1-2: New USB device found, idVendor=0483, idProduct=5740, bcdDevice= 2
[18695.468308] usb 1-2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[18695.468312] usb 1-2: Product: STM32 Virtual ComPort
[18695.468366] usb 1-2: Manufacturer: STMicroelectronics
[18695.468370] usb 1-2: SerialNumber: 204C306A4242
[18695.486633] cdc_acm 1-2:1.0: ttyACM0: USB ACM device
```

4. Launch "screen" with the identified USB-serial name in the terminal to talk to the DIVA board. (DIVA comes with a small shell-like interface over USB-serial)

\$ screen /dev/ttyACM0

Note: Baud rate is not important while connecting to DIVA.

Press ENTER and then type any character (followed by Enter) in the terminal and check the response.

```
diva>
diva>
diva>.l
List Of Challenges ..
[1] UART lab
[2] I2C Sniffing Lab
[3] SPI Sniffing Lab
[4] Firmware analysis lab
[5] JTAG lab
[6] 802.15.4 lab
[7] BLE lab
[8] Diva Exploitation lab
[9] Board Test
Choose the challenge number to run?
diva>
```

If you see the DIVA console, the DIVA board is ready for challenges!

Type “.l” (**dot el**), followed by an "Enter" to list the challenges.

Important notes:

- i. To exit screen – press **[CTRL]-A**, release them, then press **k** and then press **y**.
 - ii. Do not open multiple terminal sessions with any device, including DIVA, as it will not connect. This means simply closing the terminal where the screen is running will not help; either exit the screen or kill the process manually (ps aux | grep -l screen to find the screening process and then kill -9 <PID> to kill it by specifying the PID of the process). **ps aux | grep -i screen** to find the screening process and then **kill -9 <PID>** to kill it by specifying the PID of the process).
 - iii. The labs use screen as the hyper terminal. However, you can use any hyper-terminal software of your choice – minicom, putty etc.
5. Enter 1 for the UART lab and again enter 1 to turn ON device UART logs.

```
Choose the challenge number to run?
diva>1
Welcome to UART Port Lab

This Lab will teach you to interface with a UART port and to retrieve the device logs

Enter 1 to ON, 0 to turn OFF device logs
diva>>1
device logs activated
diva>>
```

Description:

In this lab, we will get a basic understanding of how to locate and identify UART pinouts on the target board by reading their voltage levels using DMM.

Steps:

1. Inspect the PCB to locate potential UART pinouts/headers, typically four or more pins in series.
2. Test each pin with DMM to identify its purpose, start with VCC, RX, TX, GND.
3. First Identify GND. (**Do not Powered on the DIVA Board**)



- a. Power ON DMM and put it in (---) mode, using **SEL** button.
 - b. Identify any metallic sheet area (usually metallic sheet areas are connected to ground on the board) or identify the GND pin of the input DC supply of the DIVA board.
 - c. Put one probe on the pin to be tested and the other probe on the metallic sheet or to the GND of input DC supply.
 - d. If the DMM beeps, it means that it is a GND pin.
4. To Identify VCC:
(NOTE: VCC is not used when connecting to serial (UART) interface, but

identifying it helps in narrowing our search for Tx and Rx. So, please **DO NOT** connect the VCC of the target board to your UART connector (EXPLIoT Nano))

- a. Power ON the DMM.
- b. Power on the DIVA board.
- c. Place the red probe on the pinout/header to be tested.
- d. Place the black probe on the identified GND metallic sheet, the GND of the input DC supply, or the GND pin identified in the previous step. If the DMM displays a relatively consistent voltage (e.g., 3.3V), it indicates VCC. The voltage level at which the STM operates on the board is 3.3V. When you read the datasheet, this is one important detail to verify. Typically, most microcontrollers operate at 3.3V.

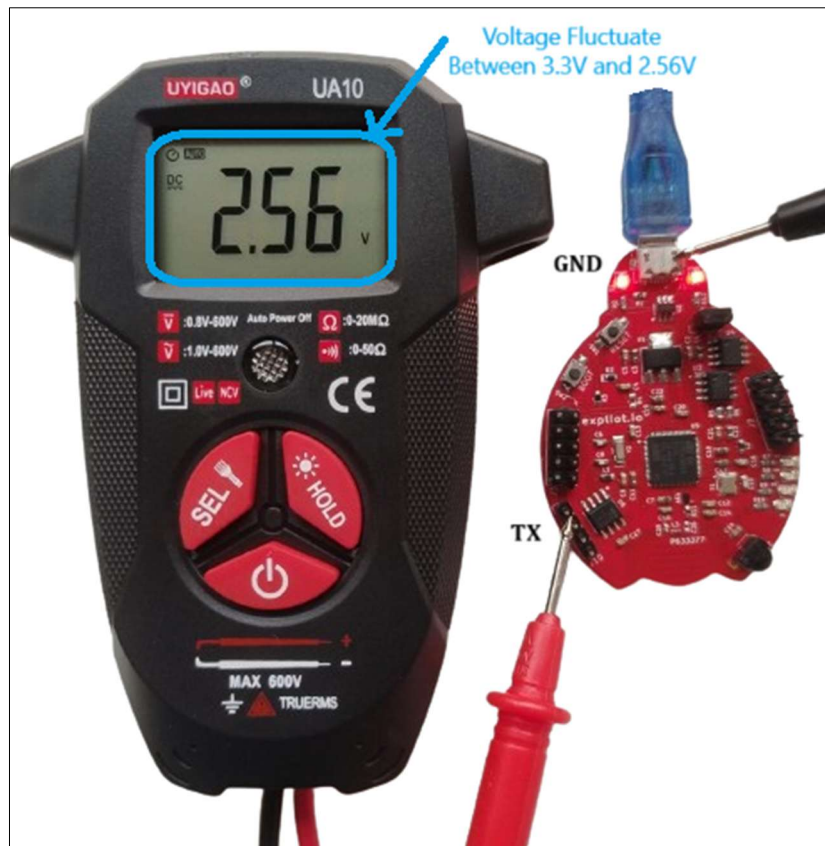
If not, repeat with other pins till you find one.



5. To identify TX

- a. Follow the same process as for VCC.
- b. If the DMM displays varying voltage, it is likely a TX pin. The reason for varying voltage is that, while transmitting data over TX, the voltage fluctuates between low and high depending on the binary value being sent (0 or 1). If no data is being transmitted, it will remain high and may be confused with VCC.

- c. If the TX pin is not found, repeat the process with other pins.
- d. If there are two pins with constant voltage, one of them could be TX. Usually, devices log information over UART when they are booted. One way to identify TX is to test it immediately after rebooting the device. You can also use a logic analyzer or oscilloscope to observe the activity on these pinouts.



6. To identify RX

This is slightly trickier than the others as the voltage may be low or high. However, if you have identified 3 of the other pins (VCC, GND, and TX), you can experiment with each of the unidentified pins in the located pin group by considering them individually as RX and connecting to see if it works. If the located group of pins has only 4 pins, then the 4th and unidentified pin should be the RX pin.

Lab 3 – UART Communication with Target

Objective:

Once the UART port is identified, communicate with the target device to further analyze its read/write access. The objective is to understand the process of interfacing with the target device, identify the correct baud rate it uses for communication, and analyze the read/write access.

Description:

You will learn to interface and communicate with the target device over a UART connection. This involves identifying the correct baud rate that the device uses and determining whether you have read-only access or read/write access over UART. If a UART connection is enabled, the devices usually provide:

- i. A Linux or custom shell over the UART connection – This is typically utilized by support or development staff for troubleshooting. However, if accessed by attackers, it opens many avenues for exploiting the device, such as reverse engineering binaries, backdooring, debugging to find vulnerabilities, and accessing scripting source codes.
- ii. Read only logs – These typically include boot log messages and other runtime messages. They are equally important as they provide crucial information about the hardware, firmware, and system behavior.

Setup:

1. DIVA IoT board
2. EXPLIoT Nano
3. F-F Jumper wires

Steps:

1. Interfacing identified UART pinouts with EXPLIoT Nano
 - a. Connect EXPLIoT Nano with the UART pins on the DIVA board using wires and breakaway headers.
 - b. Solder wires/headers if required.
 - c. EXPLIoT Nano Pin 3 RX -----→ TX UART Pin on board
 - d. EXPLIoT Nano Pin 5 TX -----→ RX UART Pin on board
 - e. EXPLIoT Nano Pin 1 GND -----→ GND UART Pin on board

Important Note:

- iv. DO NOT connect VCC pins of the target board and EXPLIoT Nano.
- v. The TX of one device connects to the RX of another and vice versa.
- vi. DO NOT connect the TX to TX and/or RX to RX.

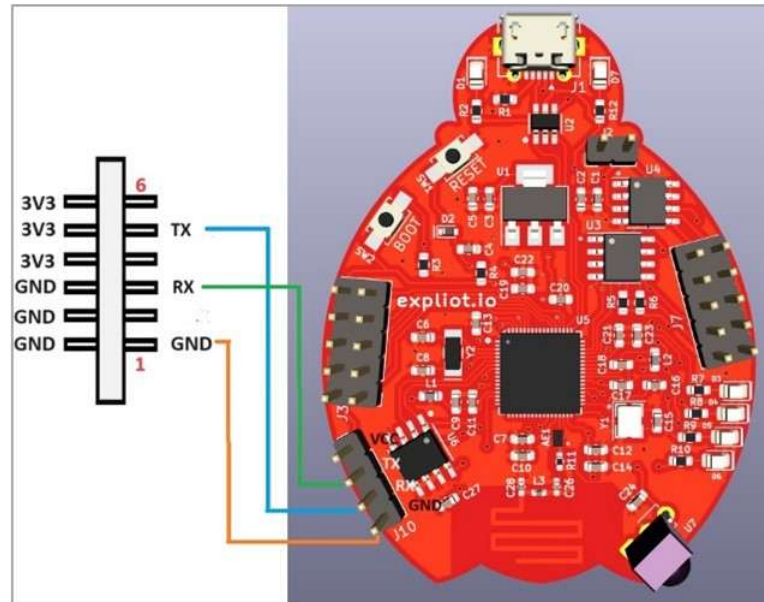
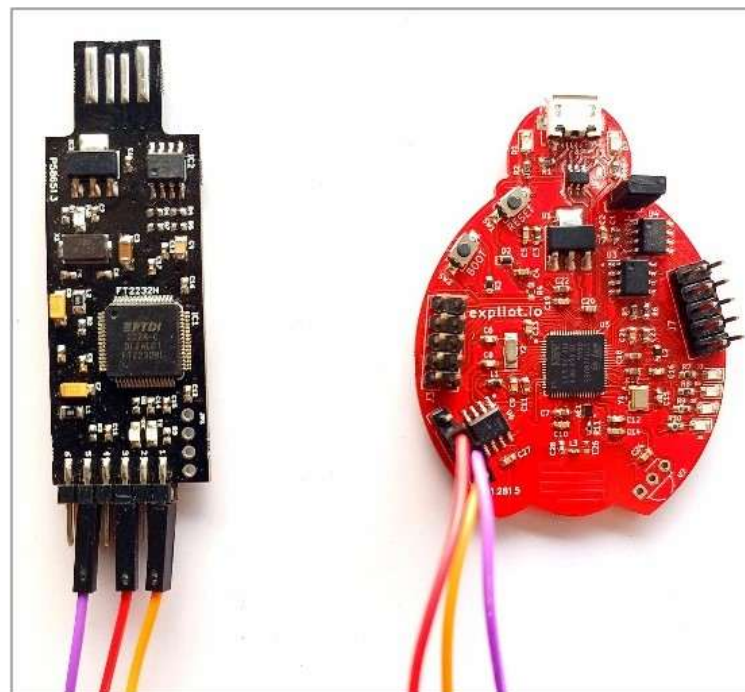


Fig. – DIVA board and EXPLIoT Nano UART connection.



2. Baud rate identification:

As the UART connection is asynchronous there is no clock monitoring the data synchronization. The way UART devices communicate correctly is by using pre-configured baud rate (the rate at which data is transmitted). Unless both the devices use the same baud rate, they will receive garbage data on their RX. Hence, we need to identify the correct baud rate before we can start communicating. For this we will use the baud scan plugin of EXPLIoT framework.

- i. Install EXPLIoT Framework if it is not already installed. The source code and documentation can be found here- <https://expiot.readthedocs.io/en/latest/index.html>
- ii. Most devices will have a default baud rate of 9600 or 115200. However, if you see binary garbage data, it is most likely that the baud rate specified is wrong.
- iii. After making the physical connections between the EXPLIoT Nano and DIVA, connect the EXPLIoT Nano to your PC via USB. Ensure that the DIVA is powered on and that the UART lab is initiated using a separate DIVA command-line screen session.
- iv. Identify the device file – **ttty*** of the Nano on your PC as mentioned in the above labs.
- v. Run the EXPLIoT framework (as root/sudo if your system does not allow user processes to connect to ttty* devices).

\$ sudo exploit

- vi. Run the baud scan plugin, to see the options use -h:

ef> run uart.generic.baudscan -h

For example: **ef> run uart.generic.baudscan -p /dev/ttyUSB0**

- vii. It will start reading data using different baud rates such as 1200, 2400, 4800, 9600, 19200, 38400, 57600, and 115200, and will provide a percentage of ASCII data in the communication for each baud rate. The baud rate with the highest percentage is most likely the correct one.
- viii. Once it is detected, note it down and make sure to exit the EXPLIoT framework. **ef> exit**

```
ef> run uart.generic.baudscan -p /dev/ttyUSB0
[*] Test:          uart.generic.baudscan
[*] Author:        Aseem Jakhar
[*] Author Email:  aseemjakhar@gmail.com
[*] Reference(s):  ['https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter']
[*] Category:      Protocol=uart|Interface=hardware|Action=analysis
[*] Target:        Name=generic|Version=generic|Vendor=generic
[*]
[*] Connecting to the the serial port (/dev/ttyUSB0) timeout (3)
[*] Scanning for baud rates: 1200,2400,4800,9600,19200,38400,57600,115200
[+] (baud=1200)
[+] (ascii ratio=0/6 0.0%)
[+] (baud=2400)
[+] (ascii ratio=4/10 40.0%)
[+] (baud=4800)
[+] (ascii ratio=8/20 40.0%)
[+] (baud=9600)
[+] (ascii ratio=30/30 100.0%)
[+] Found correct baud rate - 9600
[+] Test uart.generic.baudscan Passed
```

Fig. – we found the correct baud rate **9600** as it has 100% ascii data.

3. Access the device over UART:

- i. Now that we have identified the correct baud rate, use *the screen* to gain access to the device. EXPLIoT Nano port will be /dev/ttyUSB0.
- ii. Power on the DIVA, if it's not powered and make sure that you start the UART Lab from the DIVA console. After that run *the screen* or any other hyper terminal/serial console utility of your choice – minicom, putty etc.

\$ sudo screen <device file> <identified baud rate>

For example: \$ sudo screen /dev/ttyUSB0 9600.

- iii. If you see any log messages on screen, you have accessed the UART. Typically, on devices you may find the Linux shell, custom shells created by the vendors or read only logs. These logs are always helpful and if you get a shell through it, then nothing like it. Sometimes vendors may protect the shell with passwords. These passwords may be Linux user passwords, which can be extracted from the firmware if you already have it or custom passwords, which are usually hardcoded in the firmware or stored in external memory.

```
Tem 25.87 degree C
Tem 25.81 degree C
Tem 25.75 degree C
Tem 25.81 degree C
Tem 25.75 degree C
Tem 25.81 degree C
Tem 25.81 degree C
Tem 25.75 degree C
Tem 25.75 degree C
Tem 25.87 degree C
```

JTAG

Lab 1 – JTAG Identification Method 1

Objective

JTAG Debug port identification method 1 using micro-controller pins and DMM (Digital Multimeter) conductivity test. The objective is to identify JTAG pinouts from the PCB board provided. Now we have knowledge of micro-controller JTAG pins, which can be found in the datasheet of the micro-controller.

Description:

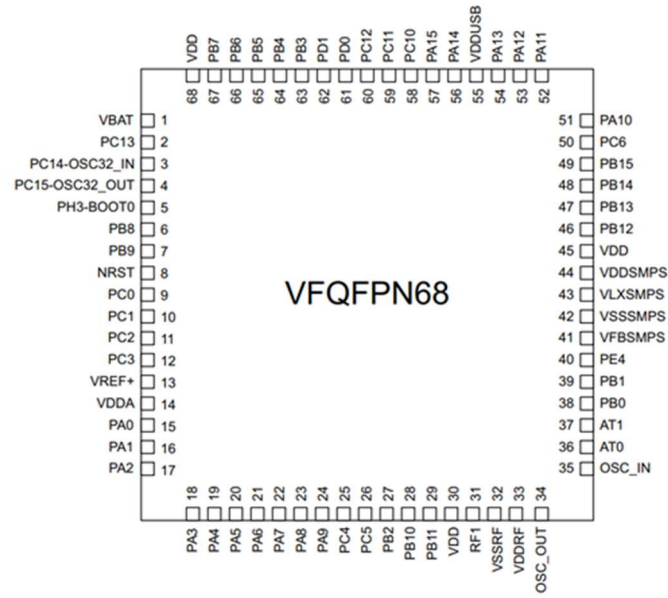
In this lab, we will learn how to use datasheets of a micro-controller to identify JTAG pinouts using DMM conductivity tests. We will learn to read the pin diagram from the datasheet and mapping pinouts from the micro-controller to the target board pinouts to identify JTAG. The same process can be used for other debug ports like SWD, JTAG etc.

Setup:

1. DMM
2. DIVA IoT board (**DIVA Board Must be Power off**)

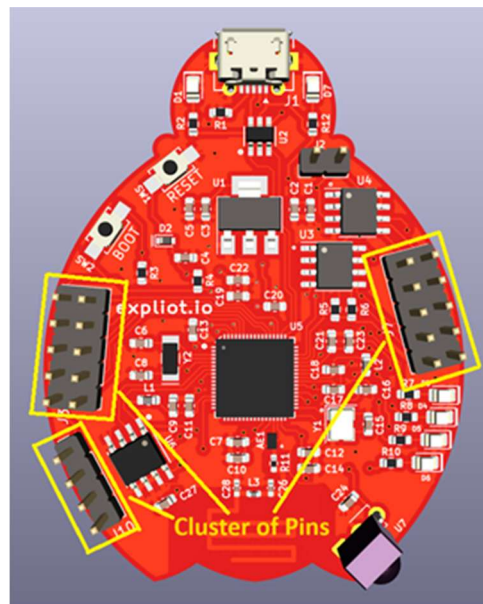
Steps:

1. The steps for this lab are the same as the “UART Identification Method 1” lab except that we are looking for JTAG pins on the micro-controller instead of the UART.
2. Identify the name printed on the microcontroller and do a quick google search for its datasheet. Identify the correct package by matching the name printed on the microcontroller and in the datasheet. Go through the datasheet and find the pinout of the microcontroller (page 57 for STM32WB55RGV6 VFQFPN68).



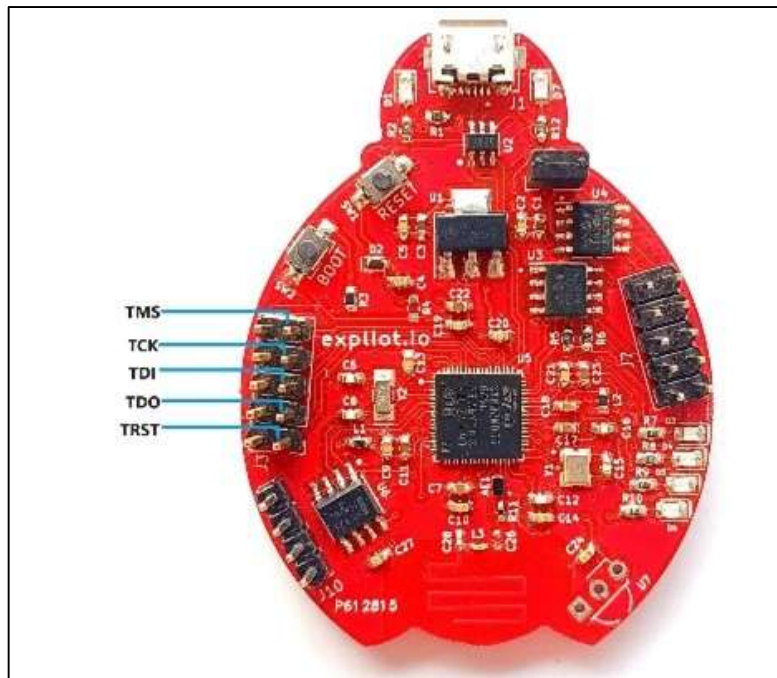
(datasheet - <http://www.st.com/resource/en/datasheet/stm32wb55re.pdf>)

3. Go to the pin definition table in the datasheet (starting at page no. 60) and locate JTAG pins for the package VFQFPN68 (pin numbers of JTCK, JTMS, JTD0, and JTDI).
4. Try to locate a cluster/group of pins/headers on the board that you want to test. After shortlisting them, number them for convenience.



5. Touch one DMM probe (either red or black) on the target board pinout/header which is to be identified and another probe on the located JTAG pin of the micro-controller.
6. If the DMM beeps, it means there is conductivity between those two points. This indicates that the pinout on the board is directly connected to the corresponding micro-controller JTAG pin. So, if you are testing for JTCK pin on the micro-controller with one of the pinouts on the boards and the DMM beeps, voila! You have found the JTCK pinout.
7. Repeat the same process to identify all the JTAG pins (JTCK, JTMS, JTDO, and JTDI)

Disclaimer: Spoiler ahead. Below image shows the JTAG pins of DIVA



Lab 2 – JTAG Identification Method 2

Objective

Automated JTAG Pin scanning and identification using JtagEnum with Arduino Nano. The objective is to understand the process and perform automated JTAG scanning using JtagEnum running on Arduino Nano.

Description:

In this lab, we will learn how to scan and identify JTAG pinouts on the target board with the help of JtagEnum, which is an open-source project available for Arduino. More details about the project and its source code are available here: <https://github.com/cyphunk/JTAGenum/>

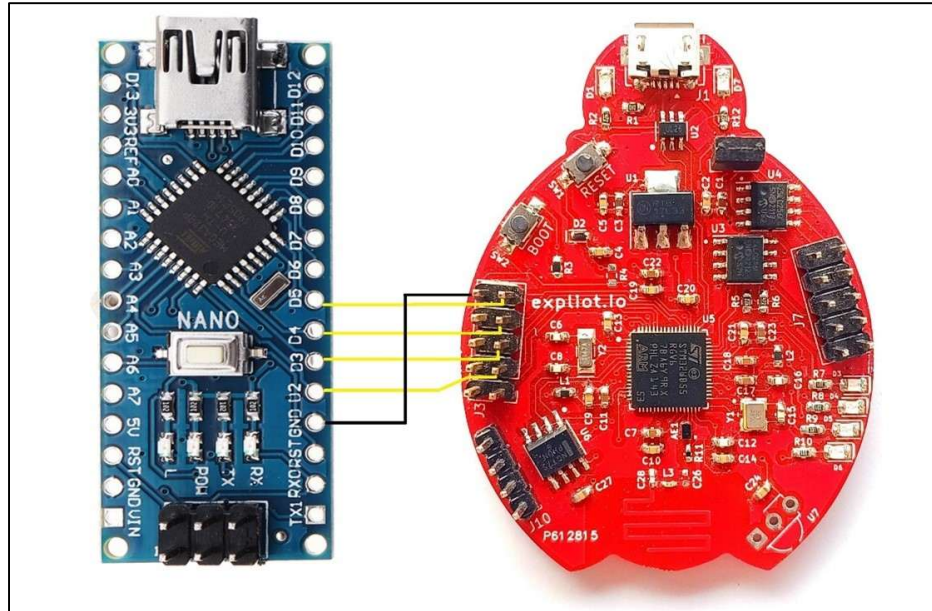
We are going to use DIVA as the target board to identify its JTAG pins.

Setup:

- a. Arduino Nano flashed with JtagEnum code (Arduino Nano is Pre flashed with JtagEnum program).
- b. Diva IoT Board
- c. Female-Female Jumper wires

Steps:

1. Connect the Arduino Nano pins, D2 to D6 to the suspected pins (cluster of pins) on the DIVA board using female to female jumper cables, as shown in the image. Note: Order of the pins doesn't matter.
2. Power ON the DIVA board using a USB to Micro-USB cable.



3. Connect Arduino Nano to your PC. (If you are using a virtual box, select the Arduino in USB options of the virtual box guest to pass through Arduino to the guest from the host machine.) You can use your PC to access Arduino over serial directly. To access Arduino nano over serial, run the following command.

\$ sudo screen /dev/ttyUSB0 115200

4. On the serial terminal, start the JTAG scan by pressing **S**

```

ERATTA
-----
l > loopback check
  ignores tck,tms. if patterns passed to tdo pins are
  connected there is a short or a false-positive
  condition exists that should be taken into account
r > pullups
  internal pullups on inputs, on/off. might increase
  stability when using a bad patch cable.
v > verbose
  on/off. print tdo bits to console during testing. If on, this will slow
  down scan.
d > delay
  on/off. will slow down scan.
- > delay -
  reduce delay by 1000us
+ > delay +
h > help
n > list pin names

OTHER JTAG TESTS
-----
Each of the following will not scan/find JTAG and require
that you manually set the JTAG pins. See their respective
call from the loop() function of code to set.

l > pattern scan single
  runs a full check on one code-defined tdi<->tdo pair.
  look at the main()/loop() code to specify pins.
x > boundary scan
  checks code defined tdo for 4000+ bits.
  look at the main()/loop() code to specify pins.
y > irenum
  sets every possible Instruction Register and then
  checks the output of the Data Register.
  look at the main()/loop() code to specify pins.
  
```

- It keeps on running continuously even after identifying the JTAG pins. When it finds a pin combination, it prints ***“FOUND!”*** in the scan results as shown below.

```
> 5
=====
Starting scan for pattern:0110011101001101101000010111001001
active ntrst:DIG_3 tck:DIG_4 tms:DIG_2 tdo:DIG_5 tdi:DIG_6 bits toggled:4
active ntrst:DIG_3 tck:DIG_4 tms:DIG_2 tdo:DIG_5 tdi:DIG_7 bits toggled:4
active ntrst:DIG_3 tck:DIG_4 tms:DIG_2 tdo:DIG_5 tdi:DIG_8 bits toggled:4
active ntrst:DIG_3 tck:DIG_4 tms:DIG_2 tdo:DIG_5 tdi:DIG_9 bits toggled:4
active ntrst:DIG_3 tck:DIG_4 tms:DIG_2 tdo:DIG_5 tdi:DIG_10 bits toggled:4
active ntrst:DIG_3 tck:DIG_4 tms:DIG_2 tdo:DIG_5 tdi:DIG_11 bits toggled:4
active ntrst:DIG_3 tck:DIG_4 tms:DIG_6 tdo:DIG_5 tdi:DIG_2 bits toggled:12
active ntrst:DIG_3 tck:DIG_4 tms:DIG_7 tdo:DIG_5 tdi:DIG_2 bits toggled:12
active ntrst:DIG_3 tck:DIG_4 tms:DIG_8 tdo:DIG_5 tdi:DIG_2 bits toggled:12
active ntrst:DIG_3 tck:DIG_4 tms:DIG_9 tdo:DIG_5 tdi:DIG_2 bits toggled:12
active ntrst:DIG_3 tck:DIG_4 tms:DIG_10 tdo:DIG_5 tdi:DIG_2 bits toggled:12
active ntrst:DIG_3 tck:DIG_4 tms:DIG_11 tdo:DIG_5 tdi:DIG_2 bits toggled:12
active ntrst:DIG_6 tck:DIG_2 tms:DIG_3 tdo:DIG_5 tdi:DIG_4 bits toggled:3
active ntrst:DIG_6 tck:DIG_2 tms:DIG_7 tdo:DIG_5 tdi:DIG_4 bits toggled:15
active ntrst:DIG_6 tck:DIG_3 tms:DIG_7 tdo:DIG_5 tdi:DIG_2 bits toggled:12
active ntrst:DIG_6 tck:DIG_3 tms:DIG_8 tdo:DIG_5 tdi:DIG_2 bits toggled:12
active ntrst:DIG_6 tck:DIG_3 tms:DIG_9 tdo:DIG_5 tdi:DIG_2 bits toggled:12
active ntrst:DIG_6 tck:DIG_3 tms:DIG_10 tdo:DIG_5 tdi:DIG_2 bits toggled:12
active ntrst:DIG_6 tck:DIG_3 tms:DIG_11 tdo:DIG_5 tdi:DIG_2 bits toggled:12
FOUND! ntrst:DIG_6 tck:DIG_4 tms:DIG_2 tdo:DIG_5 tdi:DIG_3 IR length: 9
active ntrst:DIG_6 tck:DIG_4 tms:DIG_2 tdo:DIG_5 tdi:DIG_7 bits toggled:4
active ntrst:DIG_6 tck:DIG_4 tms:DIG_2 tdo:DIG_5 tdi:DIG_8 bits toggled:4
active ntrst:DIG_6 tck:DIG_4 tms:DIG_2 tdo:DIG_5 tdi:DIG_9 bits toggled:4
active ntrst:DIG_6 tck:DIG_4 tms:DIG_2 tdo:DIG_5 tdi:DIG_10 bits toggled:4
active ntrst:DIG_6 tck:DIG_4 tms:DIG_2 tdo:DIG_5 tdi:DIG_11 bits toggled:4
active ntrst:DIG_6 tck:DIG_4 tms:DIG_3 tdo:DIG_5 tdi:DIG_2 bits toggled:12
active ntrst:DIG_6 tck:DIG_4 tms:DIG_7 tdo:DIG_5 tdi:DIG_2 bits toggled:12
active ntrst:DIG_6 tck:DIG_4 tms:DIG_8 tdo:DIG_5 tdi:DIG_2 bits toggled:12
active ntrst:DIG_6 tck:DIG_4 tms:DIG_9 tdo:DIG_5 tdi:DIG_2 bits toggled:12
active ntrst:DIG_6 tck:DIG_4 tms:DIG_10 tdo:DIG_5 tdi:DIG_2 bits toggled:12
active ntrst:DIG_6 tck:DIG_4 tms:DIG_11 tdo:DIG_5 tdi:DIG_2 bits toggled:12
```

- You may get more than one ***“FOUND!”***. The reason is that it tries all combinations. Multiple ***“FOUND!”*** will all have same pin numbers except for TRST, which we are not scanning and is optional in JTAG.
- Awesome! You have just identified the correct JTAG pins. Note them down for your further reference and go ahead and celebrate your finding.

Lab 3 – Micro-controller Firmware Extraction Over JTAG

Objective:

Use the JTAG Debug port to access the micro-controller and extract the firmware. The objective is to understand how to connect JTAG with the target board and how to extract data from the internal memory of the microcontroller.

Description:

We will use an open-source tool called Open On-Chip Debugger (OpenOCD) to communicate with the target board over the JTAG protocol. This will help us understand its interface and perform various operations on the microcontroller. Ultimately, we will extract the firmware from the microcontroller.

Setup:

- a. EXPLIoT-Nano multi-purpose debug adapter
- b. DIVA IoT board
- c. Jumper cables
- d. Breadboard

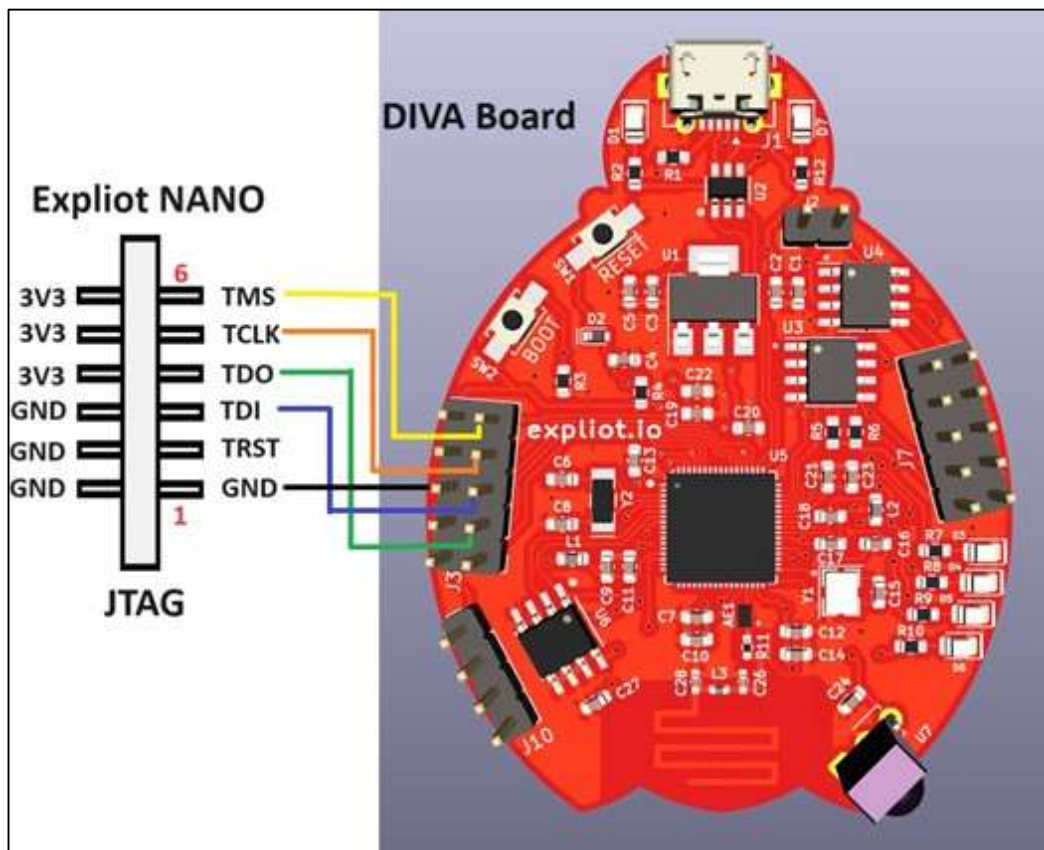
Steps:

1. Install **OpenOCD** if not already done. On Ubuntu, you can install it directly using **apt**, but this may not provide the latest version. For more details, please refer to the documentation available at - <http://openocd.org/>

Note: For DIVA 3.0 you need to install Openocd 0.11.0 or Later versions.

2. Identify the JTAG pins on the target board using any of the methods mentioned above (1 and 2).
3. Connect the JTAG pins on the target board to the JTAG pins of the EXPLIoT Nano. You can use female-to-female jumper cables for these connections.

Note: Power up the DIVA board (using a USB cable) only after all the pins are connected and the physical connection is verified.



4. Power up the DIVA board by connecting it to the laptop USB port using the micro-USB cable provided and connect the EXPLIoT Nano debug adapter to another USB port.
5. Access the DIVA console using a screen. The process has already been discussed in UART labs.

\$ lsusb

\$ sudo screen /dev/ttyACM0

- i. Note: Baud rate is not important while connecting to DIVA
 - ii. Type any character (followed by Enter) in the terminal and check the response.
 - iii. DIVA board is now ready for the lab!
6. Now launch openocd in a new terminal window.

Note: We need to pass configuration files to OpenOCD to make it aware of the target micro-controller (STM32 for this lab) and the debug adapter used (EXPLIoT Nano for this lab). For a different microcontroller or adapter configuration, you may need to select the corresponding configuration files. These configuration files are provided with the lab manual. Additionally, you may need to run OpenOCD as root/sudo if your system does not allow users to access the tty* devices.

\$ sudo openocd -f <path to config file> <EXPLIoT Nano JTAG .cfg file>

```
dinesh@dinesh-VirtualBox:~$ sudo openocd -f /home/dinesh/Downloads/Explot/explot_nano_jtag.cfg
[sudo] password for dinesh:
Open On-Chip Debugger 0.11.0
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
DEPRECATED! use 'adapter driver' not 'interface'
Info : auto-selecting first available session transport "jtag". To override use 'transport select <transport>'.
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : clock speed 500 kHz
Info : JTAG tap: stm32wbx.cpu tap/device found: 0x6ba00477 (mfg: 0x23b (ARM Ltd), part: 0xba00, ver: 0x6)
Info : JTAG tap: stm32wbx.bs tap/device found: 0x06495041 (mfg: 0x020 (STMicroelectronics), part: 0x6495, ver: 0x0)
Info : stm32wbx.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : starting gdb server for stm32wbx.cpu on 3333
Info : Listening on port 3333 for gdb connections
```

7. OpenOCD starts a Telnet session for command line console access. Launch a Telnet session by running the following command in a new terminal window:

\$ telnet localhost 4444

8. Through the Telnet interface, we can perform memory read, write, and debug operations.

1. The following are the OpenOCD commands we would need for the labs.

```
poll      - to check microcontroller status
halt      - to halt the microcontroller operation (Once halted the controller
            won't execute any instructions until restored by a "resume" or "step" command)
resume    - resumes from the halt state
step      - executes the next instruction and goes to the halt again
mdw/mdh/mb - display a word/halfword/byte from memory of the given address.
            (w-4bytes, h-2bytes, b-1byte)
            usage: mdw address <count>          count is optional
mww/mwh/mwb - write a word/halfword/byte to memory of given address
dump_image - to dump a large chunk of memory
            usage: dump_image /path/to/outputfile address length
load_image - to write a large chunk of binary data to memory
            usage: load_image /path/to/inputfile address
```

2. First, we need to determine the address from which the firmware starts and the length of the firmware. The firmware is stored in the internal

flash memory. Refer to the datasheet to identify the flash memory address and the total length.

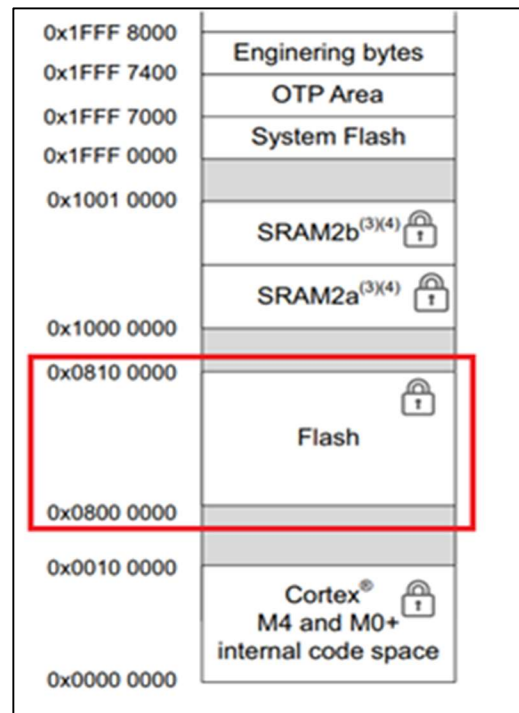


Fig. Memory mapping of the micro-controller specified in the datasheet

As we can see here, the flash memory is located at 0x08000000. Its length is calculated as $0x8100000 - 0x8000000 = 100000$. After determining the address, perform the following operations in the Telnet window -> halt the micro-controller -> dump the flash memory -> resume the controller.

3. Commands (in order)

- i. `> halt`
- ii. `> dump_image /tmp/flashdump1.bin 0x8000000 0x7FFFF`
- iii. `> resume`

```
dinesh@dinesh-VirtualBox:~$ telnet localhost 4444
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Open On-Chip Debugger
> halt
target halted due to debug-request, current mode: Thread
xPSR: 0x61000000 pc: 0x080044d6 msp: 0x2002ffa8
> dump_image /tmp/flashdump1.bin 0x8000000 0x7FFFF
dumped 524287 bytes in 15.013407s (34.103 KiB/s)

> resume
```

4. Check the file `/tmp/flashdump1.bin` to make sure that data was extracted into the file.
5. That's it! We have successfully dumped the microcontroller firmware.

(Note: Please keep the setup ready for the next exercise)

Lab 4 - Microcontroller Firmware Patching Over JTAG

Objective:

Find out the hardcoded password from the extracted firmware, modify it, write it back and login. The objective is to familiarize yourself with firmware extraction, analyzing hardcoded data, and patching the firmware on-the-fly.

Description:

We will understand why it is critical to not hardcode sensitive data in firmware and how to analyze hardcoded data from the firmware binaries.

Setup:

1. EXPLIoT Nano multi-purpose debug adapter
2. DIVA IoT board
3. Jumper cables
4. Breadboard

Steps:

1. You need to complete the previous lab before starting this one. If you haven't completed it yet, please do so now.
2. Now that we have a firmware binary. List out all the strings present inside the binary file. The easiest way to do that is to use the *strings* utility.

1. \$ strings <file_name>

3. Launch Challenge 4 (Firmware Analysis Lab) from the DIVA shell. How to get to DIVA shell is documented in the above labs.

```
diva>4
Welcome to Firmware analysis Lab

In this Lab, you will do basic firmware analysis

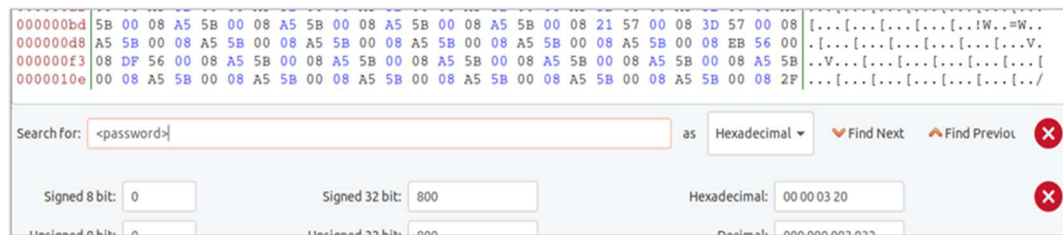
Please login
Enter the password
diva>>
```

4. When it asks for a password, try out any random password and access will be denied.
5. If the password is hardcoded in the firmware binary, you need to analyze the strings and identify those that could potentially be passwords. You can automate this process by writing a Python script to extract interesting strings

and then send them over the serial connection to the DIVA board. Analyze the responses to determine if the password was correct or not.

6. Congrats! You figured out the password. If not, keep looking.....
7. Now open the binary file in **bless** (bless is a UI based hex editor tool to view and edit binary files). Install bless, if not already installed, or you can use any other hex editor of your choice.

1. **\$ bless <file_name>**



Search for identified password inside bless (in the bottom search bar, search as Text). Modify the password, save it to a different file (for example */tmp/firmware.bin*) but make sure the size of the file does not change.

Important Note: The new password must have the same length as the old one. (If password length is 6 bytes, the modified password should have exact 6 bytes). If you change more bytes than the password length, it will brick the device and it won't boot after that, so be very careful with this lab. Just delete a byte from the password and add a new byte in its place.

8. Write the modified binary file back to DIVA using the OpenOCD-Telnet interface. If you haven't connected your DIVA board with OpenOCD, please refer to the previous lab. Once Telnet is up, perform the following operations in the Telnet console: **reset init -> write the modified binary to the flash memory -> reset the controller.**

1. Commands (in order)

- i. `> reset init`
- ii. `> flash write_image erase /tmp/firmware.bin 0x8000000`
- iii. `> reset`

```
> reset init
JTAG tap: stm32wbx.cpu tap/device found: 0x6ba00477 (mfg: 0x23b (ARM Ltd), part: 0xba00, ver: 0x6)
JTAG tap: stm32wbx.bs tap/device found: 0x06495041 (mfg: 0x020 (STMicroelectronics), part: 0x6495, ver: 0x0)
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x08005b4c msp: 0x20030000
> flash write_image erase firmware.bin 0x8000000
DAP transaction stalled (WAIT) - slowing down
device idcode = 0x20036495 (STM32WB5x - Rev 'unknown' : 0x2003)
DAP transaction stalled (WAIT) - slowing down
flash size = 1024kbytes
DAP transaction stalled (WAIT) - slowing down
flash mode : single-bank
Padding image section 0 at 0x0801c751 with 7 bytes (bank write end alignment)
Adding extra erase range, 0x0801c758 .. 0x0801cfff
DAP transaction stalled (WAIT) - slowing down
```

9. Once done with flashing the updated firmware, close openocd.
10. Reset the DIVA board (perform a power on-off reset). Connect to it over serial (*using screen*) and choose “Firmware Analysis Lab” again. Enter the modified password.

```
diva>4
Welcome to Firmware Analysis Lab

In this Lab you will do basic firmware analysis

Please login
Enter the password
diva>>*****
Access Granted :)
diva>
```

Access granted; patched firmware is working! 😊

You have successfully backdoor-ed the device.....!

Lab 5 - Microcontroller Peripheral Access Over JTAG

Objective:

Use the JTAG debug port to manipulate the GPIOs of the microcontroller and generate an LED pattern with the onboard LEDs. The objective is to understand how to control the GPIOs of the microcontroller over JTAG to change the behavior of the target according to your needs.

Description:

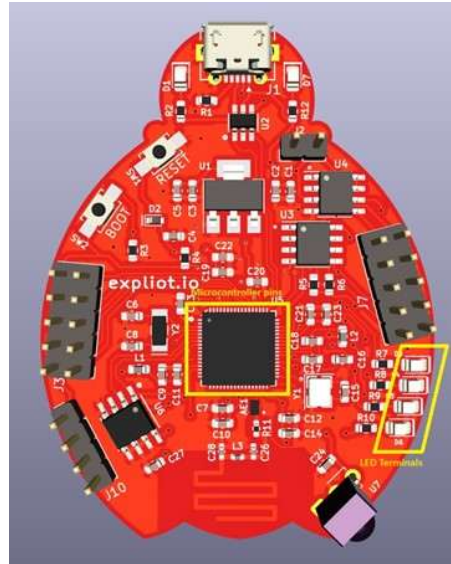
We will learn ways to control or manipulate GPIOs and other components on the board via the debug interface. We will also gain an understanding of the memory map, memory-mapped IO, GPIO registers, and related concepts of an ARM Cortex-M4 microcontroller.

Setup:

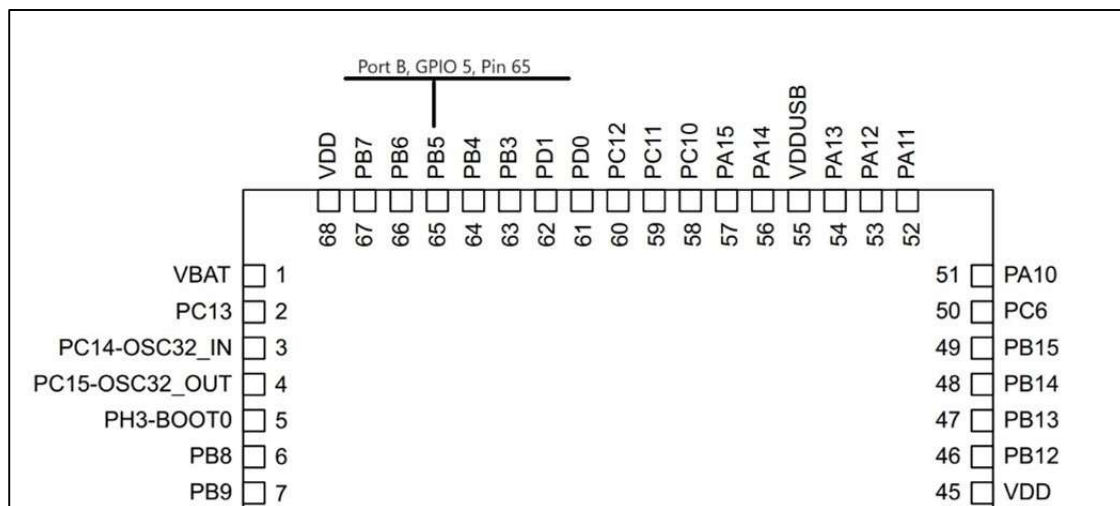
- a. EXPLIoT-Nano multi-purpose debug adapter
- b. DIVA IoT board
- c. Jumper cables
- d. Breadboard

Steps

1. Open the DIVA STM32 datasheet yet again!
2. Let us look at the LED terminals (pinouts) and how they are connected to the microcontroller.
3. Using a DMM, perform a conductivity test to identify which microcontroller pins are connected to which LED pinout on the board. You don't need the datasheet for this, as it involves directly brute-forcing the pin connections.



4. Once the pins are identified, note them down and find out which PORT (Like PORT A, PORT B, PORT C etc.) the connected LED pinouts belong to. The picture below shows how to figure out pin-port relation.



Note: Always perform conductivity test with the device in powered off state. Once you identify the pins, look at the datasheet to find out the associated port. E.g. pin 65 => port B.

5. The ARM Cortex-M4 uses memory-mapped I/O, meaning all the peripherals, including GPIOs, are mapped to the same address space. The advantage is that normal memory access operations will work on GPIO registers too. Find the memory map of the **identified port**, for example, **GPIOC** (refer to the datasheet).

0x4800 1000 - 0x4800 13FF	1 K	GPIOE
0x4800 0C00 - 0x4800 0FFF	1 K	GIPIOD
0x4800 0800 - 0x4800 0BFF	1 K	GPIOC
0x4800 0400 - 0x4800 07FF	1 K	GPIOB
0x4800 0000 - 0x4800 03FF	1 K	GPIOA

6. Dump the memory corresponding to the identified port from DIVA over the OpenOCD telnet interface. If you haven't connected your DIVA board with OpenOCD, please refer to the previous labs. Once the telnet connection is established, perform the following operations in the telnet shell to extract the GPIO memory: **halt the micro-controller -> dump GPIO registers -> resume the controller.**

Commands (in order):

- i. `> halt`
- ii. `> dump_image /tmp/sample1.bin 0x48000400 0x3FF`
- iii. `> resume`

```
> halt
target halted due to debug-request, current mode: Thread
xPSR: 0x61000000 pc: 0x080044dc msp: 0x2002ffa8
> dump_image /tmp/sample1.bin 0x48000400 0x3FF
dumped 1023 bytes in 0.050373s (19.833 KiB/s)
> resume
```

7. Launch Challenge 5 (JTAG Lab) from the DIVA shell. The DIVA board should be connected to the PC over USB, and we use the **screen** to access the shell.

```
diva>5
Welcome to JTAG Lab

In This Lab you will control onboard LED via JTAG
Enter 1 to turn ON, 0 to OFF LED0
Enter 2 for blink pattern

diva>>1
LEDs activated
diva>
```

8. Type 1 followed by Enter to turn on the first LED.
9. Perform the following operations in the telnet shell to extract GPIO memory.
halt the micro-controller -> dump GPIO registers -> resume the controller

Commands (in order)

- i. `> halt`
- ii. `> dump_image /tmp/sample2.bin 0x48000400 0x3FF`
- iii. `> resume`

```
> resume
> halt
target halted due to debug-request, current mode: Thread
xPSR: 0x61000000 pc: 0x080044d4 msp: 0x2002ffa8
> dump_image /tmp/sample2.bin 0x48000400 0x3FF
dumped 1023 bytes in 0.045210s (22.097 KiB/s)

> resume
```

10. Compare both dumps (sample1.bin and sample2.bin) using **vbindiff** to identify the bytes responsible for the LED state change. Some trial and error may be involved here. vbindiff is a utility for comparing and diffing binary files.

sample1.bin															
0000	0000:	B5	A6	DA	A9	00	03	00	00	CA	F8	00	00	00	01
0000	0010:	50	13	00	00	00	10	00	00	00	00	00	00	00	00
0000	0020:	00	00	00	77	44	00	50	55	00	00	00	00	00	00
0000	0030:	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000	0040:	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000	0050:	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000	0060:	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000	0070:	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000	0080:	00	00	00	00	00	00	00	00	00	00	00	00	00	00
sample2.bin															
0000	0000:	B5	A6	DA	A9	00	03	00	00	CA	F8	00	00	00	01
0000	0010:	70	13	00	00	00	10	00	00	00	00	00	00	00	00
0000	0020:	00	00	00	77	44	00	50	55	00	00	00	00	00	00
0000	0030:	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000	0040:	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000	0050:	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000	0060:	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000	0070:	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000	0080:	00	00	00	00	00	00	00	00	00	00	00	00	00	00

11. Calculate the offset of the changed bytes. **Target address = identified GPIO port start address (in micro-controller memory map) + offset of the**

identified byte (in GPIO memory dump). You can also write different values to the target address and verify the changes in LEDs glowing pattern.

12. For writing bytes to the micro-controller memory, use openocd-telnet command mwb or mwh

> mwh 0x48000414 0xFFFF

13. For more fun, write an OpenOCD script to display an LED running pattern from top to bottom. OpenOCD comes with a **TCL**-like interpreter built-in, which helps automate commands. For example, the command:

> loop i 0 16 {sleep 100;mwh 0x48000414 [expr 0b1 << \$i]}

I2C

Lab 1 – I2C Chip Reconnaissance

Objective:

To perform reconnaissance on the Microchip 24LC256 EEPROM, refer to the datasheet. The objective is to understand the information necessary to obtain from the memory chip datasheet, which will aid in further analysis.

Description:

In this lab, we will go through the datasheet of the 24LC256 EEPROM IC to understand the package, pin specifications, and various other parameters such as chip size, maximum clock frequency, pinouts, etc., which are important for troubleshooting in case we encounter issues with reading or writing the chip.

Download the datasheet

(source: <https://www.microchip.com/wwwproducts/en/24LC256>)

Steps:

Go through the datasheet and find the following answers.

1. Identify the size:

Description

The Microchip Technology Inc. 25AA256/25LC256 (25XX256⁽¹⁾) are 256-Kbit Serial Electrically Erasable PROMs. The memory is accessed via a simple Serial Peripheral Interface (SPI) compatible serial bus. The bus signals required are a clock input (SCK) plus separate data in (SI) and data out (SO) lines. Access to the device is controlled through a Chip Select ($\overline{\text{CS}}$) input.

2. Identify the clock frequency:

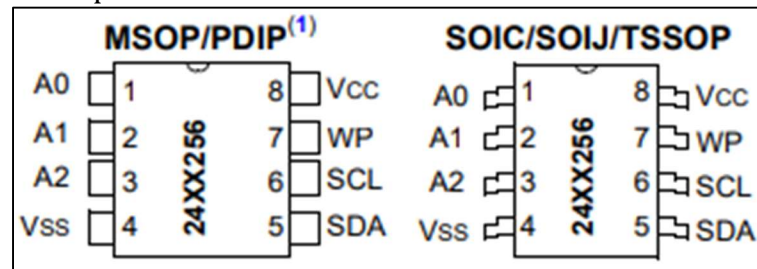
Device Selection Table

Part Number	Vcc Range	Maximum Clock Frequency
24AA256	1.7V-5.5V	400 kHz ⁽¹⁾
24LC256	2.5V-5.5V	400 kHz
24FC256	1.7V-5.5V	1 MHz ⁽²⁾

Note 1: 100 kHz for Vcc < 2.5V.

2: 400 kHz for Vcc < 2.5V.

3. Identify the Pin specifications:



4. Identify the slave address:

2.1 A0, A1, A2 Chip Address Inputs

The A0, A1 and A2 inputs are used by the 24XX256 for multiple device operations. The levels on these inputs are compared with the corresponding bits in the client address. The chip is selected if the compare is true.

Note: For the MSOP package only, pins A0 and A1 are not connected.

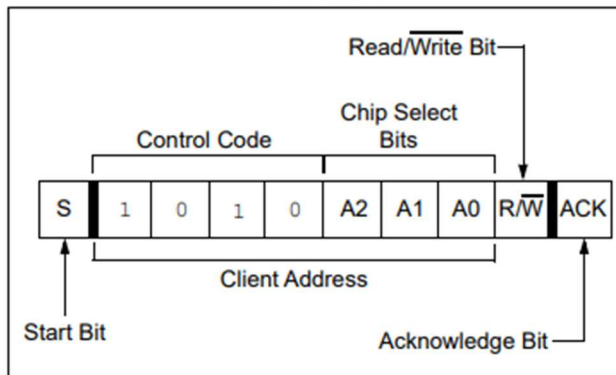
Up to eight devices (two for the MSOP package) may be connected to the same bus by using different Chip Select bit combinations. These inputs must be connected to either VCC or VSS.

In most applications, the chip address inputs A0, A1 and A2 are hard-wired to logic '0' or logic '1'. For applications in which these pins are controlled by a microcontroller or other programmable device, the chip address pins must be driven to logic '0' or logic '1' before normal device operation can proceed.

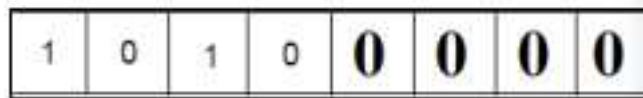
5. Understand the commands:

The control byte consists of a 4-bit control code. For the 24LC256, this is set to '1010' in binary for read and write operations. The next three bits of the control byte are the Chip Select bits (A2, A1, A0). These Chip Select bits allow the use of up to eight 24LC256 devices on the same bus and determine which device is accessed. The Chip Select bits in the control byte must correspond to the logic levels on the corresponding A2, A1, and A0 pins for the device to respond. Effectively, these bits serve as the three most significant bits of the word address.

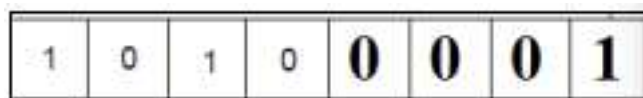
FIGURE 5-1: CONTROL BYTE FORMAT



So, if we transmit 1010 **000[0]** to the slave chip, it means it's a write command, followed by address and data.



And if we transmit 1010 000[1] to the slave chip, it means it's a read command. 24LC256 contains an address counter that maintains



the address of the last word accessed, internally incremented by '1'. Therefore, if the previous read access were to address n (n is any legal address), the next current address read operation would access data from address n + 1.

Identify SPI Read/Write/Write enable commands.

TABLE 3-1: INSTRUCTION SET

Instruction Name	Instruction Format	Description
READ	0000 0011	Read data from memory array beginning at selected address
WRITE	0000 0010	Write data to memory array beginning at selected address
WRDI	0000 0100	Reset the write enable latch (disable write operations)
WREN	0000 0110	Set the write enable latch (enable write operations)
RDSR	0000 0101	Read STATUS register
WRSR	0000 0001	Write STATUS register

Lab 2 – I2C Communication Sniffing to Bypass Authentication

Objective:

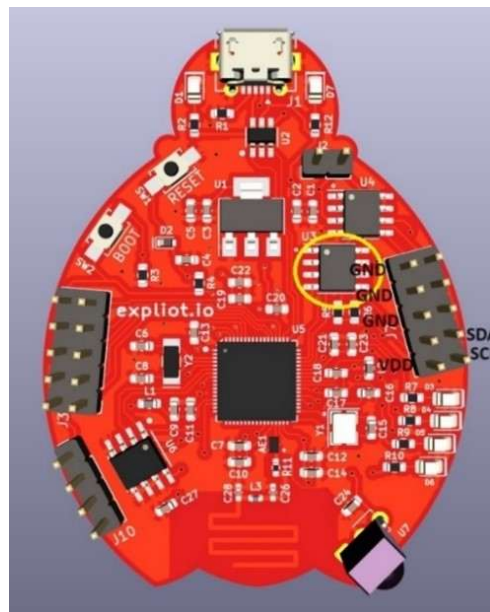
Sniffing I2C communication using a logic analyzer and bypassing authentication. The objective is to gain experience in interfacing with a logic analyzer and reverse-engineer the communication between the micro-controller and the I2C chip to identify any issues or vulnerabilities.

Description:

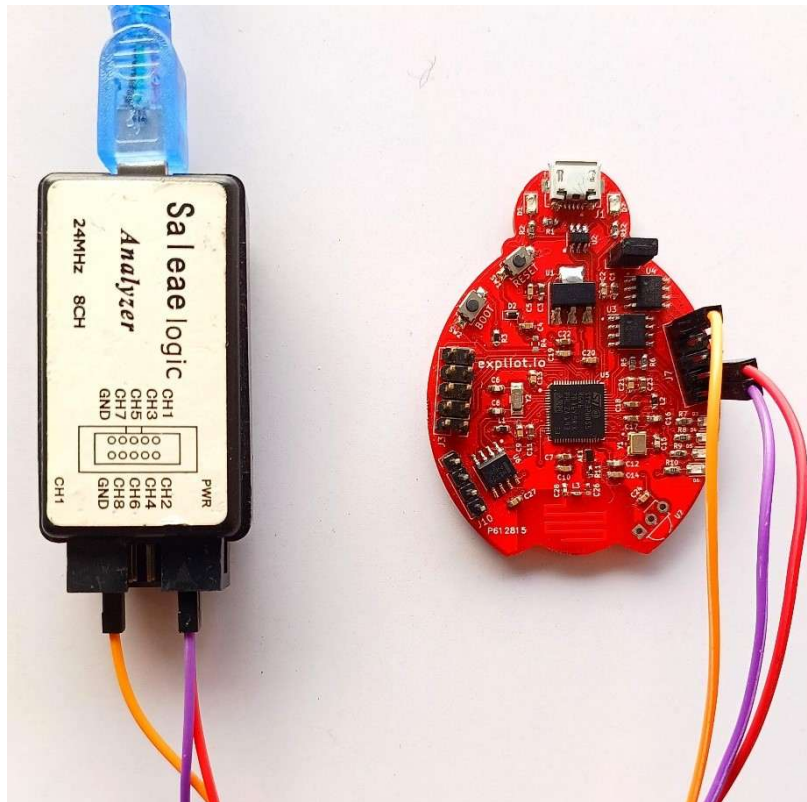
In this lab, we will learn how to use a Logic Analyzer on the DIVA board to sniff the communication between the micro-controller and the I2C EEPROM memory and analyze it to find any sensitive information. After sniffing the data, we will analyze the data transferred during the authentication process on DIVA. The DIVA board has two memory chips; the one marked as **U3** is the I2C memory chip, which is in a Small Outline Package (SOIC).

Steps:

1. Find the pinout of the I2C EEPROM chip on the DIVA board. Use the conductivity test to identify if they are exposed on any of the pin headers on the DIVA board. The process is the same as for UART or JTAG identification Method 1 (with datasheet). You need to look for SCL and SDA (and GND). If you are feeling lazy ;), the figure below shows the I2C pin headers on the DIVA board.



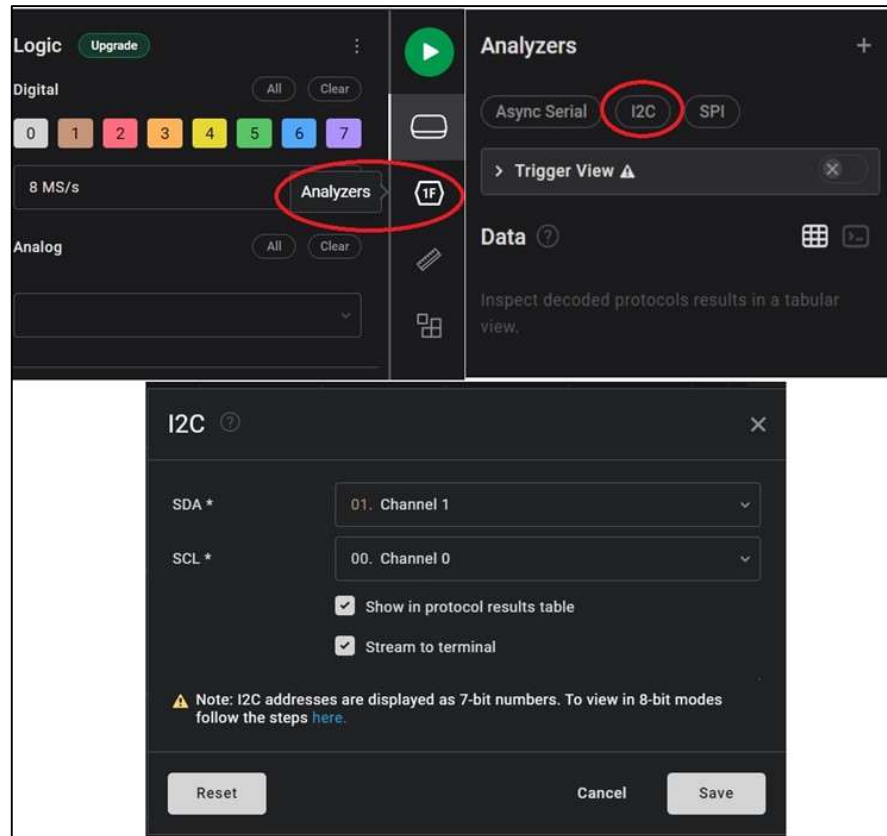
2. Connect the Logic analyzer to the DIVA board. CH0 and CH1 should be connected to SCL and SDA respectively. The GND of the DIVA board should be connected to the GND of the Logic analyzer.



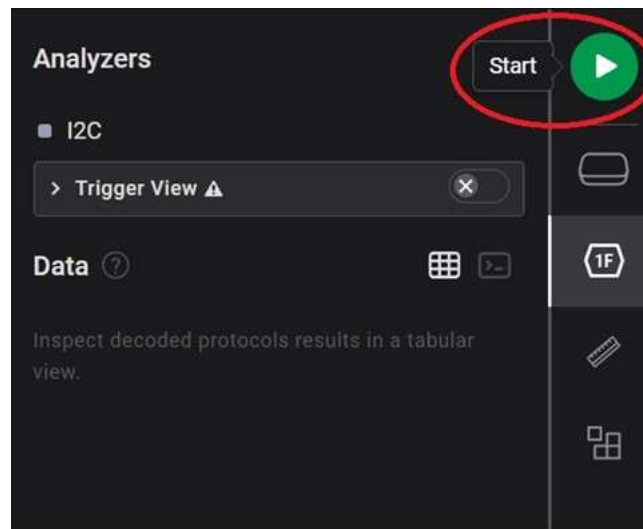
Logic Analyzer Pins	DIVA Board Pins
CH0	SCL
CH1	SDA
GND	GND

3. Connect both the Logic analyzer and the DIVA board to the PC using the USB cable.
4. Open the Logic analyzer software with root privileges on the PC.
5. Start the *screen* for the DIVA board. Set the correct serial device name and baud rate as 9600.
6. Click on the Analyzer on the right of logic software, select I2C, and set the correct channel number for appropriate wires.

NOTE: If your logic analyzer device channels start from CH1 instead of CH0, then assume CH1 on the Logic Analyzer as CH0 on the Logic software and so on. There might be a manufacturing issue with some devices.



7. Click start on the Logic analyzer software and return to the DIVA board console.



8. On the terminal enter **.h** for help and **.l** to list the challenges

```
diva>
diva>
diva>.h
diva>.l
List Of Challenges ..
[1] UART lab
[2] I2C Sniffing Lab
[3] SPI Sniffing Lab
[4] Firmware analysis lab
[5] JTAG lab
[6] 802.15.4 lab
[7] BLE lab
[8] Diva Exploitation lab
[9] Board Test
Choose the challenge number to run?
diva>
```

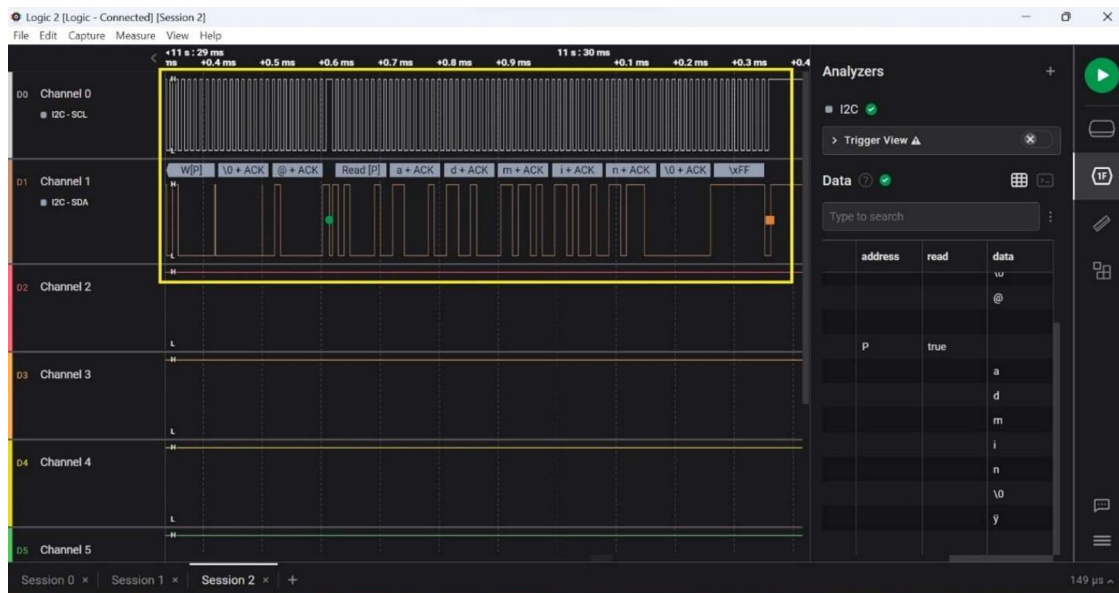
9. Enter **2** (for I2C sniffing Lab) and type the password when asked. Since we do not know the password, type any incorrect password. Once done, click on **stop** on the Logic analyzer software.

```
Choose the challenge number to run?
diva>2
Welcome to I2C Sniffing Lab

In this Lab you will sniff an I2C transaction

Please login
Enter the password
diva>>*****
Access Denied :(
diva>
```

- On the bottom right-side of Logic software, we can see the decoded protocol communication including the data that was sniffed. Analyze the data and try to figure out what was sniffed.



Hint: we can see that the data (string) “*admin*” is read from the chip. Does it ring a bell? 😊

Lab 3 – I2C Chip Memory Dumping

Objective:

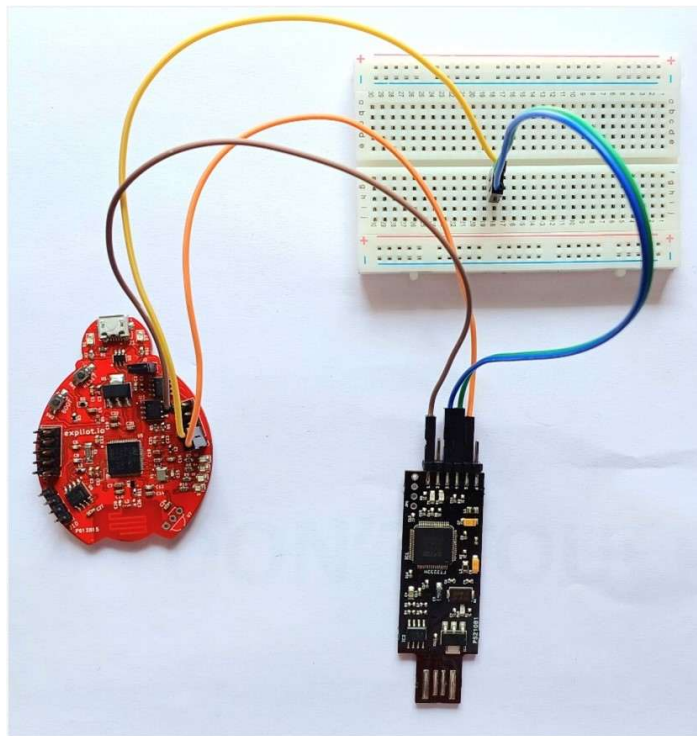
I2C EEPROM memory dumping using EXPLIoT Nano. The objective is to get familiar with the process of dumping I2C memory chip data.

Description:

In this lab, we will learn how to use EXPLIoT nano and EXPLIoT framework to dump data from I2C EEPROM memory.

Steps:

- 1) Find the pinout of the I2C EEPROM present on the DIVA board as done in the previous lab.
- 2) Connect the SDA/SCL/GND of the DIVA to the EXPLIoT NANO pinouts.
- 3) Use Breadboard for shorting pin 3 and pin 4 of the EXPLIoT NANO which is SDA.



- 4) To extract data from I2C EEPROM type the following command in the EXPLIoT framework.

a. *ef> run i2c.generic.readeprom -c 24AA256 -w <filename.bin>*

```
ef> run i2c.generic.readeprom -c 24AA256 -w xyz.bin
[*] Test: i2c.generic.readeprom
[*] Author: Aseem Jakhar
[*] Author Email: aseemjakhar@gmail.com
[*] Reference(s): ['https://github.com/eblot/pyi2cflash']
[*] Category: technology=i2c|Interface=hardware|Action=analysis
[*] Target: Name=generic|Version=generic|Vendor=generic
[*]
[*] Reading data from i2c eeprom at address(0) using device(ftdi:///1)
[+] (chip size=32768 bytes)
[?] Reading 32768 bytes from start address 0
Read @ 0x0000
[?] Writing data to the file (xyz.bin)
[+] Done. Total bytes read (32768) Time taken to read = 13.18 secs
[+] Test i2c.generic.readeprom passed
```

5) Use *strings* or *bleess* to see all the strings within the file.

SPI

Lab 1 – SPI Chip Reconnaissance

Objective:

To perform reconnaissance on the Microchip 25LC256 EEPROM using the datasheet. The objective is to understand the information that can be obtained from the memory chip datasheet, which will aid in further analysis.

Description

In this lab, we will review the datasheet of the 25LC256 EEPROM IC to understand its package, pin specifications, and various other parameters such as chip size, maximum clock frequency, pinouts, etc. These details are typically useful for troubleshooting in case we encounter issues with reading or writing to the chip.

Download the datasheet (source:

<https://www.microchip.com/wwwproducts/en/25LC256>)

Steps:

Go through the datasheet and find the following answers.

1. Identify the memory size

Description

The Microchip Technology Inc. 25AA256/25LC256 (25XX256⁽¹⁾) are 256-Kbit Serial Electrically Erasable PROMs. The memory is accessed via a simple Serial Peripheral Interface (SPI) compatible serial bus. The bus signals required are a clock input (SCK) plus separate data in (SI) and data out (SO) lines. Access to the device is controlled through a Chip Select ($\overline{CS}$) input.

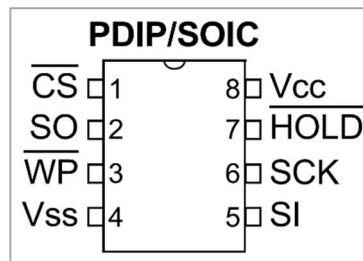
Communication to the device can be paused via the hold pin ($\overline{HOLD}$). While the device is paused, transitions on its inputs will be ignored, with the exception of Chip Select, allowing the host to service higher priority interrupts.

2. Identify the clock frequency:

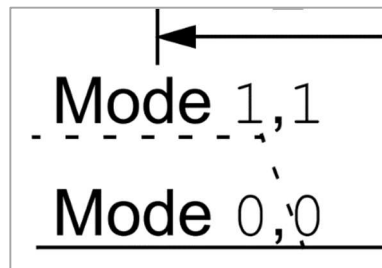
Features

- Maximum Clock 10 MHz

3. Identify the Pin specifications



4. Identify Clock polarity and phase (CPOL and CPHA)



5. Identify the SPI Read/Write/Write enable commands

TABLE 3-1: INSTRUCTION SET

Instruction Name	Instruction Format	Description
READ	0000 0011	Read data from memory array beginning at selected address
WRITE	0000 0010	Write data to memory array beginning at selected address
WRDI	0000 0100	Reset the write enable latch (disable write operations)
WREN	0000 0110	Set the write enable latch (enable write operations)
RDSR	0000 0101	Read STATUS register
WRSR	0000 0001	Write STATUS register

Lab 2 – SPI Communication Sniffing to Bypass Authentication

Objective:

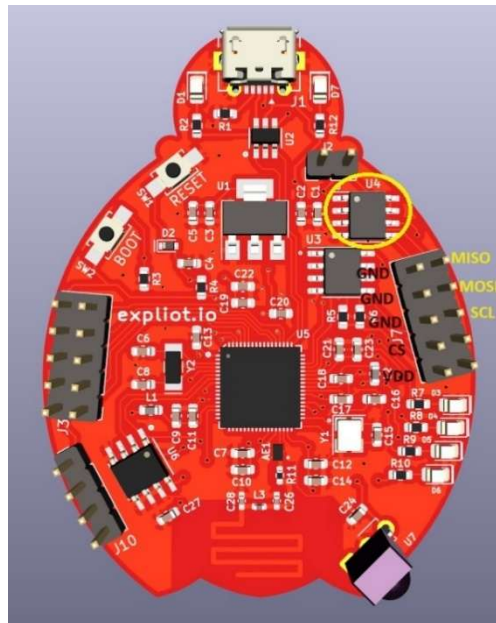
SPI communication sniffing using a logic analyzer and bypassing authentication. The objective is to gain experience in interfacing with a logic analyzer and reverse engineering the communication between the microcontroller and the SPI chip to identify any issues or vulnerabilities.

Description:

In this lab, we will learn how to use the Logic Analyzer on the DIVA board to sniff the communication between the microcontroller and the SPI EEPROM memory and analyze it to find any sensitive information. After sniffing the data, we will analyze the information transferred during the authentication process on the DIVA. The DIVA board contains two memory chips; the one marked as **U4** is the SPI memory, housed in a Small Outline Integrated Circuit (SOIC) package.

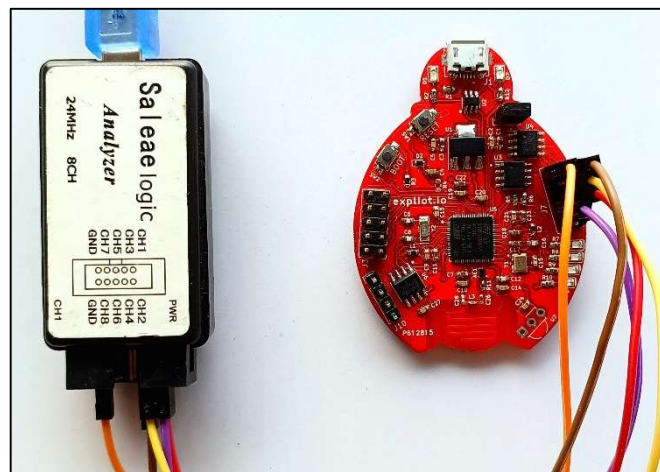
Steps:

1. Find the pinouts of the SPI EEPROM chip on the DIVA board. Use the conductivity test to identify if they are exposed on any of the pin headers on the DIVA board. The process is the same as for UART or JTAG identification Method 1 (with datasheet). If you prefer a shortcut, the figure below shows the I2C pin headers on the DIVA board.



2. Connect the logic analyzer to the DIVA board, with CH0 to CH3 connected to CS, SCLK, MOSI, and MISO in the same order. Connect the GND of the DIVA to the GND of the logic analyzer.

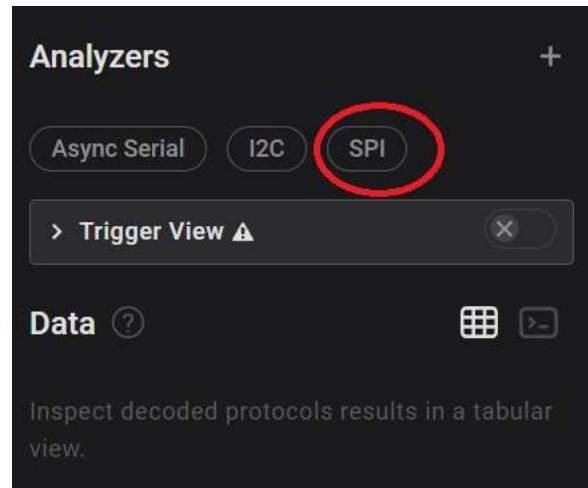
Logic Analyzer Pins	DIVA Board Pins
CH0	CS
CH1	SCLK
CH2	MOSI
CH3	MISO
GND	GND



1. Connect both the Logic analyzer and the DIVA board to the PC using the USB cables.
2. Open the Logic analyzer software with root privileges on the host machine. You need to install the Logic software, which is available for download on <https://www.saleae.com/downloads/> for free.
3. Start the *screen* for the DIVA board. Set the correct serial device name and the baud rate to 9600.
4. Click on Analyzer on the right of Logic software, select SPI, and set the correct channel number for appropriate wires.

NOTE: If your logic analyzer device channels start from CH 1 instead of CH0, then assume CH1 on the device as CH0 on the Logic software and so on. This is a manufacturing issue with some devices.

5. Press Start on the Logic software and return to the DIVA board's terminal.



SPI ?

×

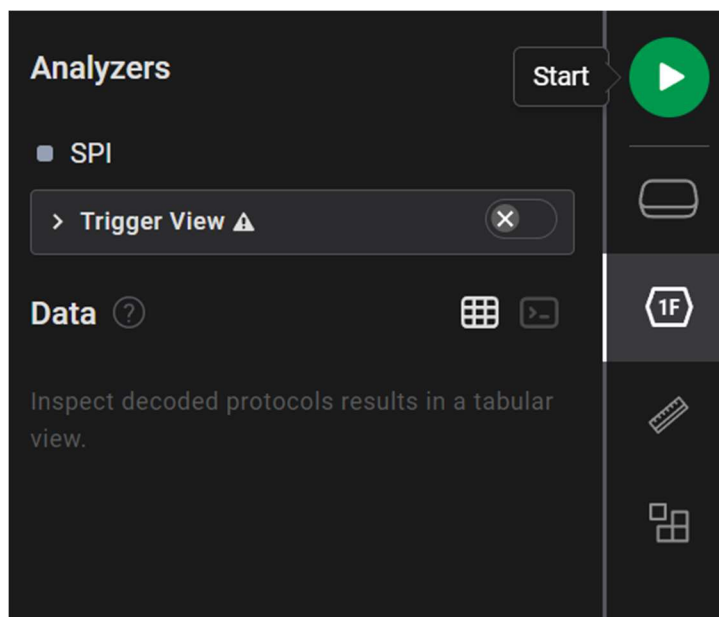
MOSI	02. Channel 2	▼
MISO	03. Channel 3	▼
Clock *	01. Channel 1	▼
Enable	00. Channel 0	▼
Significant Bit	Most Significant Bit First (Standard) ▼	
Bits per Transfer	8 Bits per Transfer (Standard) ▼	
Clock State	Clock is Low when inactive (CPOL = 0) ▼	
Clock Phase	Data is Valid on Clock Leading Edge (CPHA = 0) ▼	
Enable Line	Enable line is Active Low (Standard) ▼	

☒ Show in protocol results table

Reset

Cancel

Save



6. On the terminal enter `.h` for help and `.l` to list the challenges

```
diva>.l
List of challenges
[1] UART port Lab
[2] I2C sniffing Lab
[3] SPI sniffing Lab
[4] Firmware analysis Lab
[5] 802.15.4 Lab
[6] JTAG Lab
[7] Diva exploitation Lab
[t] Board Test
Choose the challenge number to run?
```

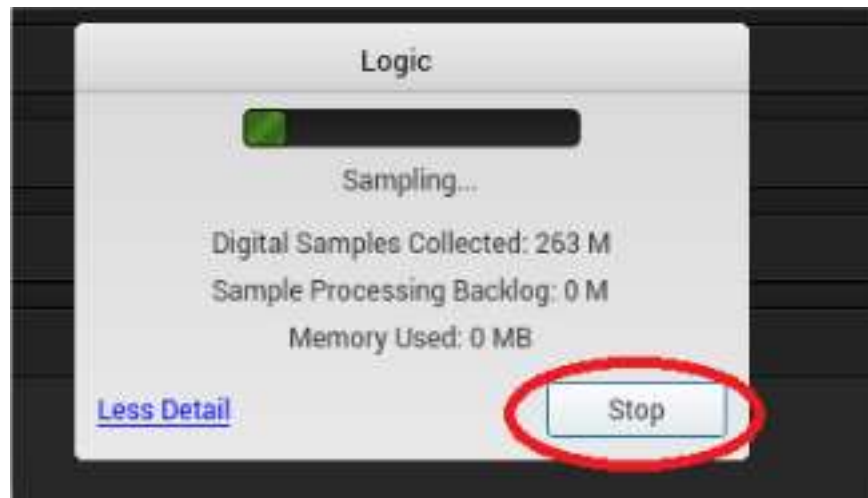
7. Enter 3 (SPI sniffing Lab) and type the password when asked.

```
diva>3
Welcome to SPI sniffing Lab

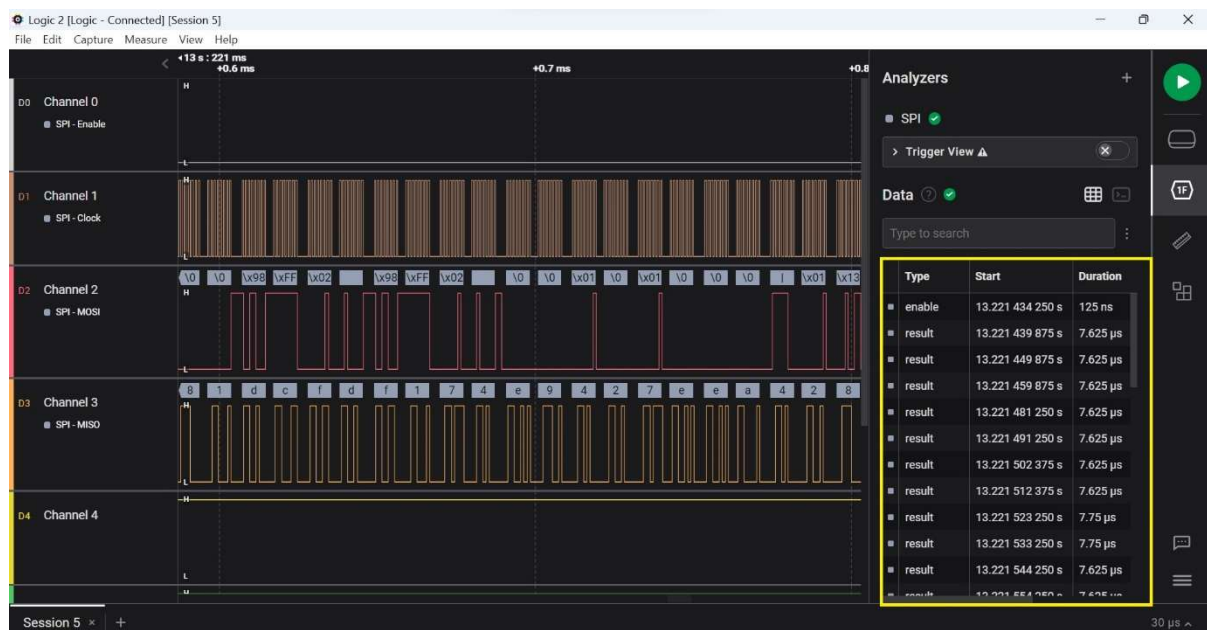
In this Lab you will sniff a SPI transaction

Please login
Enter the password
diva>>*****
Access Denied :(
diva>
```

Since we do not know the password, type any incorrect password. Once done, click on **stop** in the Logic software.



8. On the bottom right-side of the Logic software, we can see the decoded protocol communication including the data that was sniffed. Analyze the data and try to figure out what was sniffed.



Hint: The data that we sniffed is an md5 hash, if you note down all the 32 bytes and try online MD5 crackers to see if you can crack the MD5, you may get the correct password.

Lab 3 – SPI Chip Memory Dumping and Firmware Extraction

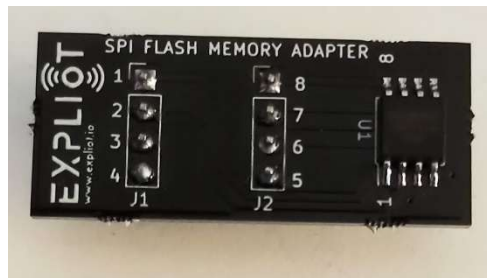
Objective:

SPI Flash memory dumping using EXPLIoT Nano.

The objective is to get familiar with the process of dumping SPI memory chip data and extracting Linux based firmware.

Description:

In this lab, we will learn how to use EXPLIoT Nano and dump data from the SPI Flash memory. We will also try to analyze the data to see if it is a valid firmware and extract it using *binwalk* utility.



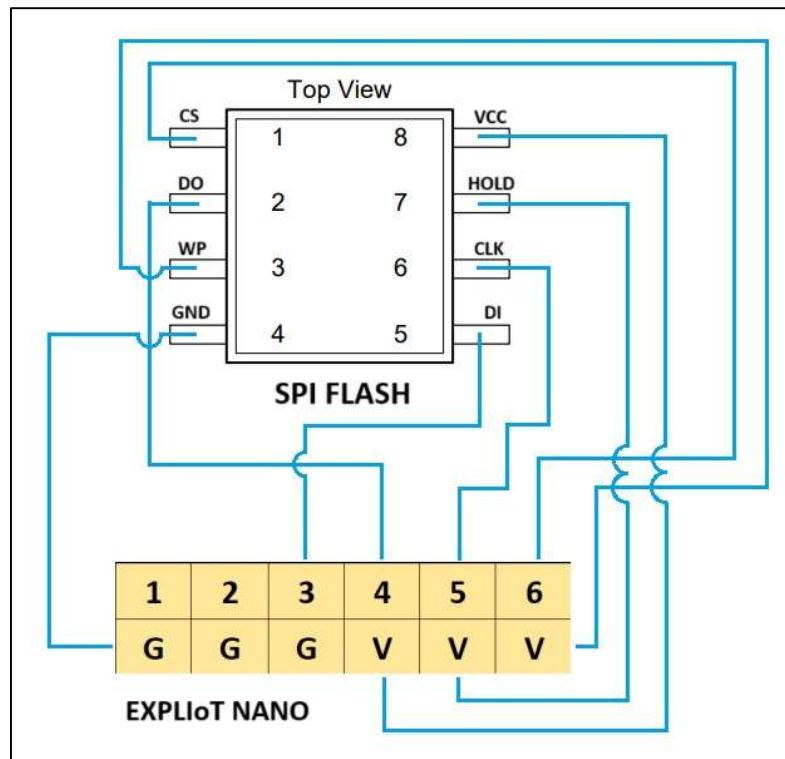
Note:

1. This lab will be performed on an SPI flash chip provided in the kit. This lab will not work for the SPI EEPROM on the DIVA board due to compatibility issues with FTDI (the chip in EXPLIoT Nano).

Download the datasheet (Source: <https://www.winbond.com/resource-files/w25q64fwrevd032513.pdf>)

Steps:

1. Connect SPI Flash memory pins to the EXPLIoT Nano as shown in the figure. Note that DO means MISO and DI means MOSI in the below diagram. Pins names may differ across vendors.



2. Now connect the EXPLIoT Nano to the host machine and check to make sure it is recognized.
3. Now start the EXPLIoT framework and run the SPI read flash plugin:
 1. ***ef> run spi.generic.readflash -w /tmp/camera_dump.bin***

```
ef> run spi.generic.readflash -w /tmp/camera_dump.bin
[*] Test: spi.generic.readflash
[*] Author: Aseem Jakhar
[*] Author Email: aseemjakhar@gmail.com
[*] Reference(s): ['https://github.com/eblot/pyspiflash']
[*] Category: technology=spi|Interface=hardware|Action=analysis
[*] Target: Name=generic|Version=generic|Vendor=generic
[*]
[*] Reading data from SPI flash at address(0) using device (ftdi:///1)
[+] (chip found=Winbond W25Q64 8 MiB)(chip size=8388608 bytes)(using frequency=30000000)
[?] Reading 8388608 bytes from start address 0
[?] Writing data to the file (/tmp/camera_dump.bin)
[+] Done. Total bytes read (8388608) Time taken to read = 2.87 s
[+] Test spi.generic.readflash passed
```

- Now you can use *binwalk* utility to extract the firmware. You need to install binwalk and its dependencies if it's not already installed. Documentation and code are here – <https://github.com/ReFirmLabs/binwalk>

1. \$ binwalk -e /tmp/camera_dump.bin

```
rupesh@rupesh-Lenovo-Ideapad-320-15IKB:~/camera_dump$ binwalk -e camera.bin
```

DECIMAL	HEXADECIMAL	DESCRIPTION
4960	0x1360	LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncompressed size: 60720 bytes
33280	0x8200	Zlib compressed data, default compression
75800	0x12818	LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncompressed size: 3806120 bytes

```
rupesh@rupesh-Lenovo-Ideapad-320-15IKB:~/_camera_dump.bin.extracted$ cd squashfs-root/
rupesh@rupesh-Lenovo-Ideapad-320-15IKB:~/_camera_dump.bin.extracted/squashfs-root$ ls
bin dev etc home init lib mnt proc sys test tmp usr var web www
rupesh@rupesh-Lenovo-Ideapad-320-15IKB:~/_camera_dump.bin.extracted/squashfs-root$ ls -la
total 72
drwxr-xr-x 14 shakir shakir 4096 Feb 21 2014 .
drwxr-xr-x 40 shakir shakir 20480 Dec 31 16:04 ..
drwxr-xr-x 2 shakir shakir 4096 Feb 21 2014 bin
drwxr-xr-x 6 shakir shakir 4096 Feb 21 2014 dev
drwxr-xr-x 7 shakir shakir 4096 Feb 21 2014 etc
drwxr-xr-x 2 shakir shakir 4096 Feb 21 2014 home
lrwxrwxrwx 1 shakir shakir 8 Feb 21 2014 init -> bin/init
drwxr-xr-x 3 shakir shakir 4096 Feb 21 2014 lib
drwxr-xr-x 3 shakir shakir 4096 Feb 21 2014 mnt
drwxr-xr-x 2 shakir shakir 4096 Feb 21 2014 proc
drwxr-xr-x 2 shakir shakir 4096 Feb 21 2014 sys
drwxr-xr-x 2 shakir shakir 4096 Feb 21 2014 test
lrwxrwxrwx 1 shakir shakir 8 Feb 21 2014 tmp -> /var/tmp
drwxr-xr-x 2 shakir shakir 4096 Feb 21 2014 usr
drwxr-xr-x 2 shakir shakir 4096 Feb 21 2014 var
lrwxrwxrwx 1 shakir shakir 8 Feb 21 2014 web -> /var/web
drwxr-xr-x 6 shakir shakir 4096 Feb 21 2014 www
```

We just extracted the camera firmware. If you are curious, look around for the extracted firmware.

BLE:

1. BLE Labs - DIVA Board:

Lab 1 – Reconnaissance

Objective:

Reconnaissance with EXPLIoT framework. The objective is to perform scanning for BLE devices in proximity.

Description:

You will learn how to conduct reconnaissance on BLE-enabled devices and utilize the EXPLIoT framework for BLE operations.

Setup:

1. DIVA Board
2. EXPLIoT Framework

Steps:

1. Here are the corrected sentences:

1. Plug in the DIVA Board to your PC.
2. Open the DIVA board console using `screen` or any other serial terminal utility.
3. In the DIVA console, enter 7 (for the BLE Lab).
4. It will display the prompt to turn ON and turn off BLE. Press 1 to turn ON BLE on the DIVA board.

```
diva>7
Welcome to BLE Lab
Enter 1 to Turn ON, 0 to turn OFF BLE
diva>>1
BLE activated
diva>>
```

2. Open the terminal and run the `hciconfig` command. You will see the available BLE interfaces such as hci0, hci1, and others.
 1. hci0 is the default interface for the first Bluetooth adapter connected to your Linux system, typically assigned to the internal adapter.

2. hci1, hci2, and so forth, represent the interfaces for any additional adapters connected to your Linux machine.

```
rupesh@rupesh-Lenovo-Ideapad-320-15IK8: ~$ sudo hciconfig
hci0:  Type: Primary  Bus: USB
       BD Address: 00:1A:7D:DA:71:13  ACL MTU: 310:10  SCO MTU: 64:8
       UP RUNNING
       RX bytes:682 acl:0 sco:0 events:48 errors:0
       TX bytes:3163 acl:0 sco:0 commands:48 errors:0
```

3. Start the EXPLIoT framework with root privileges (sudo) because scanning operations on Linux require elevated permissions. Once you are in the EXPLIoT console, execute the BLE scanner plugin to receive the Bluetooth addresses (BLEADDR) of devices within proximity.

1. *\$ sudo explot*
2. *ef> run ble.generic.scan -i (-i 0 is hci0)*

```
ef> run ble.generic.scan -i 0
[*] Test: ble.generic.scan
[*] Author: Aseem Jakhar
[*] Author Email: aseemjakhar@gmail.com
[*] Reference(s): ['https://en.wikipedia.org/wiki/Bluetooth_Low_Energy']
[*] Category: Technology=ble|Interface=radio|Action=discovery
[*] Target: Name=generic|Version=generic|Vendor=generic
[*]
[*] Scanning BLE devices for 10 second(s)
[+] name: Unknown
[+] addr: 01:1e:f9:e7:72:05
[+] addrtype: random
[+] rssi: -81 dBm
[+] connectable: False
[+] adtype_data:
[+]   adtype: 255
[+]   description: Manufacturer
[+]   value: 06000109200264080e2805e792a7bc4874affac0e370a2752fb728a3f4
[+]
[+] name: Unknown
[+] addr: 58:1c:f8:66:53:09
[+] addrtype: random
[+] rssi: -83 dBm
[+] connectable: False
[+] adtype_data:
[+]   adtype: 255
[+]   description: Manufacturer
[+]   value: 060001092002b5e12caa8f81ea1cf72130e65bfb0f3726d3230fbac4e8
[+]
[+] name: Unknown
[+] addr: 54:38:59:32:3c:ac
[+] addrtype: random
[+] rssi: -82 dBm
```

4. Try to find out the address of the DIVA board (Device Name: EXPLIoT)

```
[+]
[+] name: EXPLIoT
[+] addr: 00:80:e1:27:a1:c7
[+] addrtype: public
[+] rssi: -62 dBm
[+] connectable: True
[+] adtype_data:
[+]   adtype: 9
[+]   description: Complete Local Name
[+]   value: EXPLIoT
[+]   adtype: 1
[+]   description: Flags
[+]   value: 06
[+]   adtype: 10
[+]   description: Tx Power
[+]   value: 00
[+]
```

Lab 2 – GATT Services and Characteristics Enumeration

Objective:

To enumerate the GATT services and characteristics of the BLE device, the objective is to understand the nature of these characteristics and services, as well as how to retrieve this information from a BLE device.

Description:

You will learn to perform the enumeration of GATT services and characteristics of BLE devices. GATT services and, more importantly, characteristics, serve as the primary communication points between the device and the client, facilitating the exchange of data through characteristic values.

Setup:

1. DIVA Board
2. EXPLIoT Framework

Steps:

1. After obtaining the DIVA Board address, discover the services available on the BLE device by rerunning the EXPLIoT BLE scan plugin with the service scan option. This action will enumerate all the services operating on the device and display their UUIDs (Universally Unique IDs) along with the starting and ending handles, which are also used to denote characteristics.

ef> run ble.generic.enum -a <BLEADDR> -s

```
ef> run ble.generic.enum -a 00:80:e1:27:a1:d6 -s
[*] Test: ble.generic.enum
[*] Author: Aseem Jakhar
[*] Author Email: aseemjakhar@gmail.com
[*] Reference(s): ['https://en.wikipedia.org/wiki/Bluetooth_Low_Energy']
[*] Category: Technology=ble|Interface=radio|Action=recon
[*] Target: Name=generic|Version=generic|Vendor=generic
[*]
[*] Enumerating services/characteristics of the device 00:80:e1:27:a1:d6
[+] service_uuid: 00001801-0000-1000-8000-00805f9b34fb
[+] service_uuid_name: Generic Attribute
[+] handle_start: 0x1
[+] handle_end: 0x4
[+]
[+] service_uuid: 00001800-0000-1000-8000-00805f9b34fb
[+] service_uuid_name: Generic Access
[+] handle_start: 0x5
[+] handle_end: 0xb
[+]
[+] service_uuid: 00000000-cc7a-482a-984a-7f2ed5b3e58f
[+] service_uuid_name: 00000000-cc7a-482a-984a-7f2ed5b3e58f
[+] handle_start: 0xc
[+] handle_end: 0xe
[+]
[+] Test ble.generic.enum passed
```

2. To scan for characteristics available on the DIVA board, rerun the EXPLIoT BLE scan plugin with the characteristic scan option. This will enumerate all the characteristics on the device and display their UUIDs (Universally Unique ID) along with their handles. Take note of the handles. You can utilize the -v option of the plugin for verbose output.

ef> run ble.generic.enum -a <device address> -c

```
ef> run ble.generic.enum -a 00:80:e1:27:a1:d6 -c
[*] Test: ble.generic.enum
[*] Author: Aseem Jakhar
[*] Author Email: aseemjakhar@gmail.com
[*] Reference(s): ['https://en.wikipedia.org/wiki/Bluetooth_Low_Energy']
[*] Category: Technology=ble|Interface=radio|Action=recon
[*] Target: Name=generic|Version=generic|Vendor=generic
[*]
[*] Enumerating services/characteristics of the device 00:80:e1:27:a1:d6
[+] char_uuid: 00002a05-0000-1000-8000-00805f9b34fb
[+] char_uuid_name: Service Changed
[+] handle: 0x3
[+] supported_properties: INDICATE
[+]
[+] char_uuid: 00002a00-0000-1000-8000-00805f9b34fb
[+] char_uuid_name: Device Name
[+] handle: 0x7
[+] supported_properties: READ WRITE
[+] readvalue: TEMPLATE
[+]
[+] char_uuid: 00002a01-0000-1000-8000-00805f9b34fb
[+] char_uuid_name: Appearance
[+] handle: 0x9
[+] supported_properties: READ
[+] readvalue:
[+]
[+] char_uuid: 00002a04-0000-1000-8000-00805f9b34fb
[+] char_uuid_name: Peripheral Preferred Connection Parameters
[+] handle: 0xb
[+] supported_properties: READ
[+] readvalue: yyyyyy
[+]
[+] char_uuid: 00000000-8e22-4541-9d4c-21edae82ed19
[+] char_uuid_name: 00000000-8e22-4541-9d4c-21edae82ed19
[+] handle: 0xe
[+] supported_properties: WRITE
[+]
[+] Test ble.generic.enum passed
```

The Standard service UUIDs are defined by the Bluetooth SIG. Manufacturers can also create custom service UUIDs. For more information on enumerated UUIDs of services, please check:

<https://www.bluetooth.com/specifications/gatt/services>

Device Information	org.bluetooth.service.device_information	0x180A	Adopted
Environmental Sensing	org.bluetooth.service.environmental_sensing	0x181A	Adopted
Fitness Machine	org.bluetooth.service.fitness_machine	0x1826	Adopted
Generic Access	org.bluetooth.service.generic_access	0x1800	Adopted
Generic Attribute	org.bluetooth.service.generic_attribute	0x1801	Adopted

3. Try out different commands and analyse the results on the DIVA board.

- a. **Writechar:** Open the EXPLIoT framework and run the `ble.generic.writechar` plugin. To discover the available options, use `-h`.

```
ef> run ble.generic.writechar -h
```

To control a particular LED on the DIVA board, you can write to Handle: 0x000e with Value: 0x01010101.

```
ef> run ble.generic.writechar -a <BDADDR> -n <handle> -w <value>
```

```
run ble.generic.writechar -a 00:80:e1:27:a1:c7 -n 0x00e -w 01010101
```

Now we can turn on and turn off led by changing the above value, there is a specific sequence of bit to turn on and turn off 4 led on DIVA board, try to find out that sequence.

```
ef> run ble.generic.writechar -a 00:80:e1:27:a1:c7 -n 0x00e -w 01010101
[*] Test: ble.generic.writechar
[*] Author: Aseem Jakhar
[*] Author Email: aseemjakhar@gmail.com
[*] Reference(s): ['https://en.wikipedia.org/wiki/Bluetooth_Low_Energy']
[*] Category: Technology=ble|Interface=radio|Action=analysis
[*] Target: Name=generic|Version=generic|Vendor=generic
[*]
[*] Writing the value (01010101) to handle (0xe) on BLE device (00:80:e1:27:a1:c7)
[+] Test ble.generic.writechar passed
```

(Hint: LSB of each byte is used to control led on DIVA board.)

- b. **Fuzzchar:** One of the interesting plugins in the EXPLIoT framework allows you to fuzz the characteristic value and write to the BLE device. The plugin options are the same as `ble.generic.writechar`, except that you can mark the byte(s) in the value that you want to fuzz as "xx" (for each byte that you want to fuzz).

```
run ble.generic.fuzzchar -a <BDADDR> -n <handle> -w <value_with_XX_markers>
```

```
run ble.generic.fuzzchar -a 00:80:e1:27:a1:c7 -n 0x00e -w 0101xxxx
```

```
ef> run ble.generic.fuzzchar -a 00:80:e1:27:a1:c7 -n 0x000e -w 0101xxxx
[*] Test: ble.generic.fuzzchar
[*] Author: Arun Magesh
[*] Author Email: arun.m@payatu.com
[*] Reference(s): ['https://en.wikipedia.org/wiki/Bluetooth_Low_Energy']
[*] Category: Technology=ble|Interface=radio|Action=fuzz
[*] Target: Name=generic|Version=generic|Vendor=generic
[*]
[*] Fuzzing the value (0101xxxx), iterations (255) for handle (0xe) on BLE device (00:80:e1:27:a1:c7)
[?] Writing the fuzzed value (0101620f)
[?]
[?] Writing the fuzzed value (0101a18d)
[?]
[?] Writing the fuzzed value (01013344)
[?]
[?] Writing the fuzzed value (0101af84)
[?]
```

You can fuzz the DIVA Board by changing the last bytes. You can see the random flashing of LEDs.

Lab 3 – Attack on BLE Handle

Objective:

The objective is to attack a specific handle of a BLE device remotely using EXPLIoT tools.

Description:

In this lab, you will perform an attack on the BLE handle, attempting to crash it using the EXPLIoT framework by writing certain values to the BLE handle.

Setup:

1. Diva Board
2. EXPLIoT Framework

Setup:

1. Open the DIVA board console using screen or any other serial terminal utility. The procedure for using the screen and checking if DIVA is connected is discussed in the above hardware labs.
2. In the DIVA console, enter 7 (BLE Lab).
3. It will then display the prompt to turn ON and turn off BLE. Press 1 to turn ON BLE of the DIVA board.

```
diva>7
Welcome to BLE Lab
Enter 1 to Turn ON, 0 to turn OFF BLE
diva>>1
BLE activated
diva>>
```

4. Open EXPLIoT and run the ble.generic.writechar plugin. To find out the options available use -h

ef> run ble.generic.writechar -h

5. As you have seen in the previous lab, there is one handle available on the DIVA board BLE for controlling the LED on the DIVA board; you can write any value to that handle.

Handle: 0x000e and Value: 0x01010101

ef> run ble.generic.writechar -a <BDADDR> -n <handle> -w <value>

6. When you write any value to a handle through an app or using any tools, it will be stored in some buffer. You can then attempt a buffer overflow attack, which can crash the BLE handle. When more data is placed by a program or system process, the extra data overflows, causing some of that data to leak out into other buffers, which can corrupt or overwrite whatever data they were holding.
7. You can check the BLE handle by writing the value **0x01010101** (which will turn on all 4 LEDs). Up to a certain length, the BLE handle will work fine. However, when you successfully perform a buffer overflow attack, the BLE handle will crash, and there will be no effect of writing any value to the handle.

run ble.generic.writechar -a 00:80:e1:27:a1:c7 -n 0x00e -w 01020304050607080901020304050601

```
ef> run ble.generic.writechar -a 00:80:e1:27:a1:c7 -n 0x00e -w 01020304050607080901020304050601
[*] Test: ble.generic.writechar
[*] Author: Aseem Jakhar
[*] Author Email: aseemjakhar@gmail.com
[*] Reference(s): ['https://en.wikipedia.org/wiki/Bluetooth_Low_Energy']
[*] Category: Technology=ble|Interface=radio|Action=analysis
[*] Target: Name=generic|Version=generic|Vendor=generic
[*]
[*] Writing the value (01020304050607080901020304050601) to handle (0xe) on BLE device (00:80:e1:27:a1:c7)
[+] Test ble.generic.writechar passed
ef>
```

8. In this case, only the BLE of the DIVA board will not work, but other peripherals will function properly.
9. To restore the proper functioning of the DIVA board BLE, press the restart button or power cycle the board, and it will work properly.

2. BLE Labs - Smart Watch

Lab 1 – Reconnaissance

Objective:

Reconnaissance with the EXPLIoT framework. The objective is to perform scanning for BLE devices in proximity.

Description:

You will learn to conduct reconnaissance on BLE-enabled devices and utilize the EXPLIoT framework for BLE operations.

Setup:

1. Smart watch
2. EXPLIoT Framework

Steps:

1. Turn on your laptop's Bluetooth (as you performed the previous BLE labs, so do not need it to turn it ON intentionally)
2. Open your terminal and run the *hciconfig* command. You will see the available BLE interfaces, such as hci0, hci1, and so on.
 - i) hci0 is the default interface of the first connected Bluetooth adapter to your Linux system, typically the internal adapter.
 - ii) hci1, hci2, and so forth are the interfaces of any additional adapters you connect to your Linux machine.

```
rupesh@rupesh-Lenovo-Ideapad-320-15IK8: ~$ sudo hciconfig
hci0:  Type: Primary  Bus: USB
       BD Address: 00:1A:7D:DA:71:13  ACL MTU: 310:10  SCO MTU: 64:8
       UP RUNNING
       RX bytes:682 acl:0 sco:0 events:48 errors:0
       TX bytes:3163 acl:0 sco:0 commands:48 errors:0
```

3. Start the EXPLIoT framework with root privileges (sudo). The reason for this is that scanning operations on Linux require root privileges. Once you are in the EXPLIoT console, run the BLE scanner plugin, and you will receive all BLEADDR (Bluetooth addresses) of devices within proximity.

\$ sudo explot

ef> run ble.generic.scan -i (-i 0 is hci0)

```
ef> run ble.generic.scan -i 0
[*] Test: ble.generic.scan
[*] Author: Aseem Jakhar
[*] Author Email: aseemjakhar@gmail.com
[*] Reference(s): ['https://en.wikipedia.org/wiki/Bluetooth_Low_Energy']
[*] Category: Technology=ble|Interface=radio|Action=discovery
[*] Target: Name=generic|Version=generic|Vendor=generic
[*]
[*] Scanning BLE devices for 10 second(s)
[+] name: Unknown
[+] addr: 01:1e:f9:e7:72:05
[+] addrtype: random
[+] rssi: -81 dBm
[+] connectable: False
[+] adtype_data:
[+]   adtype: 255
[+]   description: Manufacturer
[+]   value: 06000109200264080e2805e792a7bc4874affac0e370a2752fb728a3f4
[+]
[+] name: Unknown
[+] addr: 58:1c:f8:66:53:09
[+] addrtype: random
[+] rssi: -83 dBm
[+] connectable: False
[+] adtype_data:
[+]   adtype: 255
[+]   description: Manufacturer
[+]   value: 060001092002b5e12caa8f81ea1cf72130e65bfb0f3726d3230fbac4e8
[+]
[+] name: Unknown
[+] addr: 54:38:59:32:3c:ac
[+] addrtype: random
[+] rssi: -82 dBm
```

Lab 2 – GATT Services and Characteristics Enumeration

Objective:

To enumerate the GATT services and characteristics of the BLE device, the objective is to understand what characteristics and services are and how you can obtain that information from a BLE device.

Description:

You will learn to perform the enumeration of GATT services and characteristics of BLE devices. GATT services, and more importantly, characteristics, are the main communication points between the device and the client, where data is sent and received in characteristic values.

Setup:

1. Smart Watch
2. EXPLIoT Framework

Steps:

1. Keep the Smart Watch near you. You can either scan for BLE devices to identify which one is the smart watch, or alternatively, you can check the BLE address on the watch itself by choosing the specific option.
2. After obtaining the smart watch address, find out the services available on the watch by running the EXPLIoT BLE scan plugin again with the service scan option. This will enumerate all the services running on the device and display their UUIDs (Universally Unique ID) along with the starting and ending handles. Handles are also used to denote characteristics.

```
ef> run ble.generic.enum -a <BLEADDR> -s
```

```
ef> run ble.generic.enum -a ba:03:64:1b:c5:0e -s
[*] Test: ble.generic.enum
[*] Author: Aseem Jakhar
[*] Author Email: aseemjakhar@gmail.com
[*] Reference(s): ['https://en.wikipedia.org/wiki/Bluetooth_Low_Energy']
[*] Category: Technology=ble|Interface=radio|Action=recon
[*] Target: Name=generic|Version=generic|Vendor=generic
[*]
[*] Enumerating services/characteristics of the device ba:03:64:1b:c5:0e
[+] service_uuid: 00001800-0000-1000-8000-00805f9b34fb
[+] service_uuid_name: Generic Access
[+] handle_start: 0x1
[+] handle_end: 0x5
[+]
[+] service_uuid: 00001801-0000-1000-8000-00805f9b34fb
[+] service_uuid_name: Generic Attribute
[+] handle_start: 0xc
[+] handle_end: 0xf
[+]
[+] service_uuid: 0000180a-0000-1000-8000-00805f9b34fb
[+] service_uuid_name: Device Information
[+] handle_start: 0x10
[+] handle_end: 0x18
[+]
[+] service_uuid: 0000180f-0000-1000-8000-00805f9b34fb
[+] service_uuid_name: Battery Service
[+] handle_start: 0x19
[+] handle_end: 0x1d
[+]
[+] service_uuid: 6e400001-b5a3-f393-e0a9-e50e24dcca9d
[+] service_uuid_name: 6e400001-b5a3-f393-e0a9-e50e24dcca9d
[+] handle_start: 0x1e
[+] handle_end: 0x23
[+]
[+] service_uuid: 0000fd00-0000-1000-8000-00805f9b34fb
[+] service_uuid_name: fd00
[+] handle_start: 0x24
[+] handle_end: 0x29
[+]
[+] Test ble.generic.enum passed
```

3. The Standard service UUIDs are defined by the Bluetooth SIG. Manufacturers can also create custom service UUIDs. For more information on enumerated UUIDs of services, please check: <https://www.bluetooth.com/specifications/gatt/services>

Device Information	org.bluetooth.service.device_information	0x180A	Adopted
Environmental Sensing	org.bluetooth.service.environmental_sensing	0x181A	Adopted
Fitness Machine	org.bluetooth.service.fitness_machine	0x1826	Adopted
Generic Access	org.bluetooth.service.generic_access	0x1800	Adopted
Generic Attribute	org.bluetooth.service.generic_attribute	0x1801	Adopted

4. To scan for characteristics available on the watch, run the EXPLIoT BLE scan plugin again with the characteristic scan option. It will enumerate

all the characteristics on the device and display their UUIDs (Universally Unique IDs) along with their handles. Make a note of the handles. You can use the ``-v`` option of the plugin for verbose output.

ef> run ble.generic.enum -a <BLEADDR> -c

```
ef> run ble.generic.enum -a ba:03:64:1b:c5:0e -c
[*] Test: ble.generic.enum
[*] Author: Aseem Jakhar
[*] Author Email: aseemjakhar@gmail.com
[*] Reference(s): ['https://en.wikipedia.org/wiki/Bluetooth_Low_Energy']
[*] Category: Technology=ble|Interface=radio|Action=recon
[*] Target: Name=generic|Version=generic|Vendor=generic
[*]
[*] Enumerating services/characteristics of the device ba:03:64:1b:c5:0e
[+] char_uuid: 00002a00-0000-1000-8000-00805f9b34fb
[+] char_uuid_name: Device Name
[+] handle: 0x3
[+] supported_properties: READ
[+] readvalue: LM716
[+]
[+] char_uuid: 00002a01-0000-1000-8000-00805f9b34fb
[+] char_uuid_name: Appearance
[+] handle: 0x5
[+] supported_properties: READ
[+] readvalue:
[+]
[+] char_uuid: 00002a05-0000-1000-8000-00805f9b34fb
[+] char_uuid_name: Service Changed
[+] handle: 0xe
[+] supported_properties: INDICATE
[+]
[+] char_uuid: 00002a26-0000-1000-8000-00805f9b34fb
[+] char_uuid_name: Firmware Revision String
[+] handle: 0x12
[+] supported_properties: READ
[+] readvalue: 0.3.1
[+]
[+] char_uuid: 00002a27-0000-1000-8000-00805f9b34fb
[+] char_uuid_name: Hardware Revision String
[+] handle: 0x14
[+] supported_properties: READ
[+] readvalue: 01
[+]
```

5. The Standard characteristics UUIDs are defined by the Bluetooth SIG. For more information on enumerated UUIDs of characteristics, please check: - <https://www.bluetooth.com/specifications/gatt/characteristic>

OTS Feature	org.bluetooth.characteristic.ots_feature	0x2ABD	Adopted
Peripheral Preferred Connection Parameters	org.bluetooth.characteristic.gap.peripheral_preferred_connection_parameters	0x2A04	Adopted
Peripheral Privacy Flag	org.bluetooth.characteristic.gap.peripheral_privacy_flag	0x2A02	Adopted
PLX Continuous Measurement	org.bluetooth.characteristic.plx_continuous_measurement	0x2A5F	Adopted
PLX Features	org.bluetooth.characteristic.plx_features	0x2A60	Adopted

Lab 3 – Sniffing GATT Protocol Communication (Android)

Objective:

Sniff the GATT protocol communication on Android. The objective is to understand how to sniff the GATT protocol communication, on an Android phone, with a BLE peripheral.

Description:

Students will learn to sniff GATT protocol communication between the device and mobile application, enabling them to analyze and identify triggers/actions related to the corresponding characteristics/handles.

Setup:

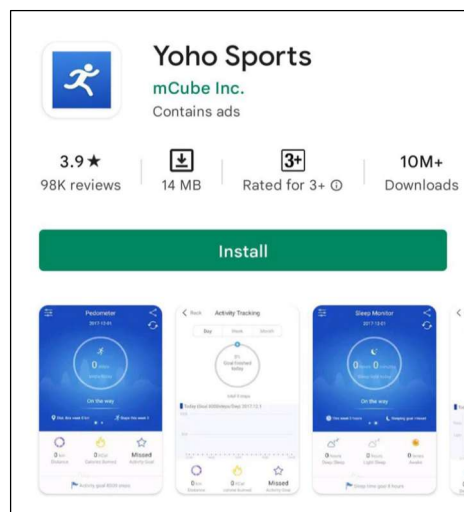
1. Smart Watch
2. Android Device

Steps:

1. To sniff the communication, you need to activate the btsnoop utility on your Android phone from the developer options. Follow the steps below to enable it.
 1. Go to Settings.
 2. In Settings, go to Developer options.
 3. If you do not see Developer Options on your Android phone, you need to enable it. Go to Settings -> About Phone -> Build Number and tap the “Build number” option seven times. If the process is different for your phone, search for it online.
 4. In the Developer options screen, select “Enable Bluetooth HCI snoop log”. The log file is now enabled.
 5. Turn Bluetooth OFF and then turn it back ON.
 6. Reboot the Android device.
 7. The HCI log file is created on the SD Card. The default location is “/sdcard/btsnoop_hci.log”. For some manufacturers, the file path may be different, so please verify that.



2. On your Android phone, download and install the “**Yoho Sports**” application from the Play Store.



3. Pair the smart band with the Android application and perform operations such as setting the time, sending notifications, and measuring heart rate.
4. This will generate a **btsnoop_hci.log** file on the phone's SD card.
5. Extract the **btsnoop_hci.log** from the mobile device and open Wireshark to analyze the packets. The log file is a PCAP file that can be viewed in Wireshark.
6. You can download the file by connecting your phone to your PC and mounting it as a drive, or by using the ADB utility provided by Android to retrieve the file. There is plenty of information available on the Internet about both processes.

Lab 4 – Analyze GATT Protocol Communication

Objective:

To identify write characteristic commands, their handles, and the values written when triggering actions on the BLE device. The objective is to understand how to identify and analyze which characteristics correspond to the commands or operations you perform on the device via the Android app.

Description:

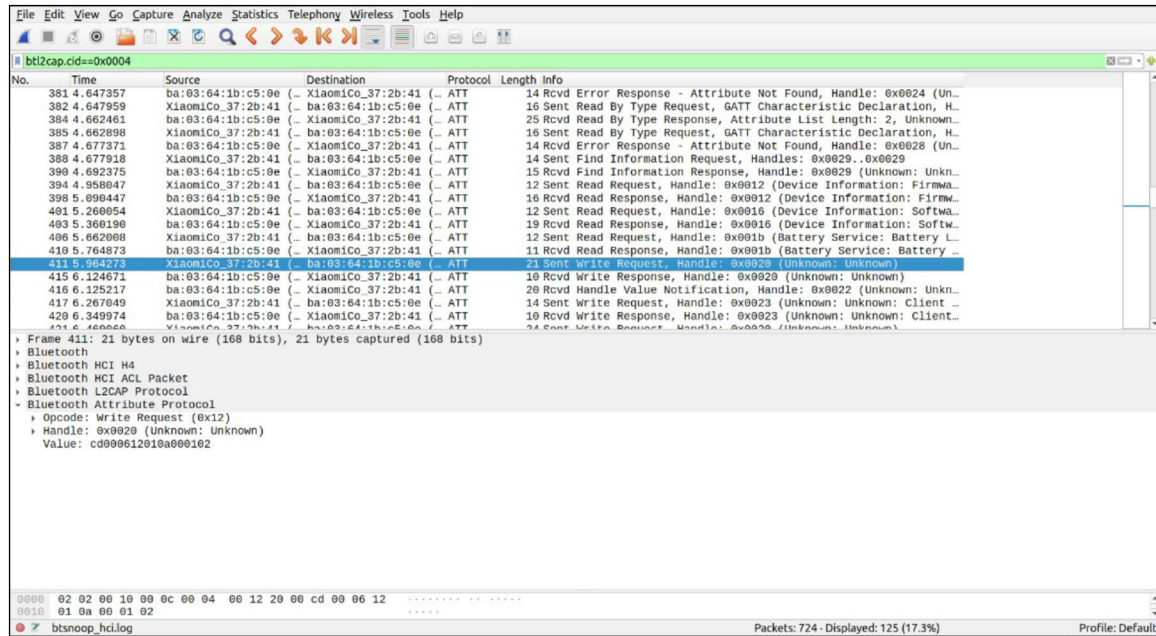
You will gain an understanding of the GATT protocol and learn how to identify read and write commands. This knowledge will help you determine the handles to write to and the values to be written, enabling remote control of the BLE device.

Setup:

1. Wireshark

Steps:

1. You can use the PCAP file from the previous lab for analysis. Once you have gained understanding, you can create different PCAP logs for individual operations performed on the device to capture the exact write commands, characteristics, and their values for each operation.
2. For this lab, we will use the PCAP file **smartwatch.log** provided with this lab manual.
3. To find read and write commands, apply filters (**bt12cap.cid==0x0004**) in Wireshark. CID stands for Channel Identifier in the L2CAP protocol and identifies a channel endpoint on the device. The Channel ID **0x0004** is used for data exchanged over the GATT protocol. Take note of your findings, such as characteristic handles and their corresponding values.

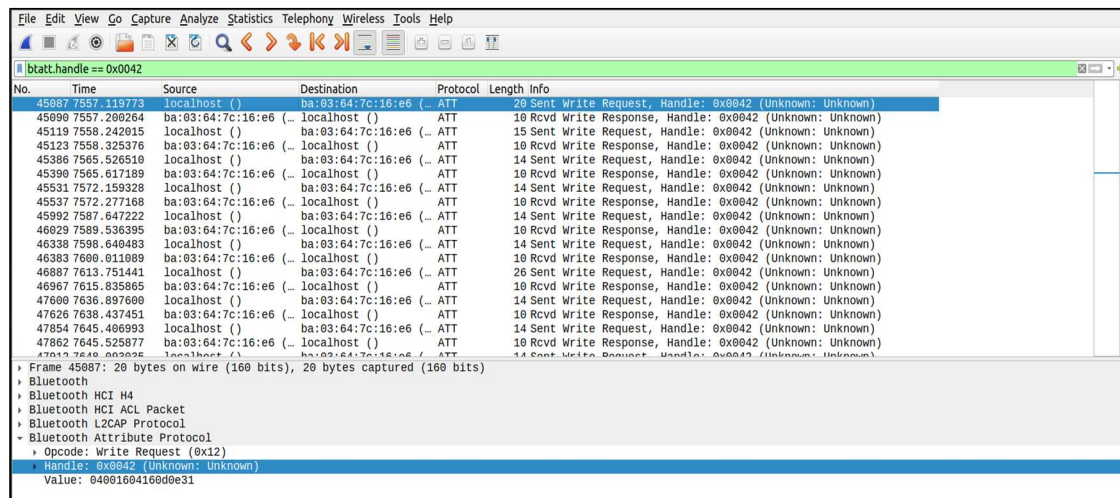


4. For example, we can see below the **Sent Write Command**,

Handle: 0x0042, Value: 04001604160d0e31

04001604 16 0d 0e 31

16 0d 0e 31 this is the hex value, if we convert it to decimal, we get to know the actual time and date which is displayed on the smart watch.



If you write and modify the value **16 0d 0e 31**, you will learn the date, month, hours, and minutes changing in the watch. In the next lab, we will write the values.

Lab 5 – Controlling The BLE Peripheral Device

Objective:

Control the behavior of the device using the write characteristic command in the GATT protocol. The objective is to understand how to communicate with the device using tools instead of the actual app, sending specific values, and analyzing or remotely controlling its behavior.

Description:

You will be able to read and write the GATT commands over BLE communication with the BLE peripheral device (smart watch in this case). If you can successfully write to a handle and change the behavior of the device, you have found a potential issue. The behavior and the handle would be device specific.

Setup:

1. Smart Watch
2. EXPLIoT Framwework

Steps:

1. In the previous lab, we identified various write commands using the handles.
2. Experiment with different commands and analyze the results on the watch.
 1. Open EXPLIoT and run the *ble.generic.writechar* plugin. To find out the options available use `-h`

ef> run ble.generic.writechar -h

To change the time and date we will write Handle: 0x0042 and Value: 04001604160d0e31.

ef> run ble.generic.writechar -a <BDADDR> -n <handle> -w <value>

```
ef> run ble.generic.writechar -a ba:03:64:7c:16:e6 -n 0x0042 -w 04001604160d0e31
[*] Test: ble.generic.writechar
[*] Author: Aseem Jakhar
[*] Author Email: aseemjakhar@gmail.com
[*] Reference(s): ['https://en.wikipedia.org/wiki/Bluetooth_Low_Energy']
[*] Category: Technology=ble|Interface=radio|Action=analysis
[*] Target: Name=generic|Version=generic|Vendor=generic
[*]
[*] Writing the value (04001604160d0e31) to handle (0x42) on BLE device (ba:03:64:7c:16:e6)
[+] Test ble.generic.writechar passed
```

2. Now we can change the Hour and Minute by changing the value.

From **0400160416 0d 0e 31** to **0400160416 0a 1e 31**, we have updated the values. Now, write the new value and observe the updated time and date.

```
ef> run ble.generic.writechar -a ba:03:64:7c:16:e6 -n 0x0042 -w 04001604160a1e31
[*] Test: ble.generic.writechar
[*] Author: Aseem Jakhar
[*] Author Email: aseemjakhar@gmail.com
[*] Reference(s): ['https://en.wikipedia.org/wiki/Bluetooth_Low_Energy']
[*] Category: Technology=ble|Interface=radio|Action=analysis
[*] Target: Name=generic|Version=generic|Vendor=generic
[*]
[*] Writing the value (04001604160a1e31) to handle (0x42) on BLE device (ba:03:64:7c:16:e6)
[+] Test ble.generic.writechar passed
```

Controlling Heart Rate:

Handle: 0x0042 and Value: 0a01

```
ef> run ble.generic.writechar -a ba:03:64:7c:16:e6 -n 0x0042 -w 0a01
[*] Test: ble.generic.writechar
[*] Author: Aseem Jakhar
[*] Author Email: aseemjakhar@gmail.com
[*] Reference(s): ['https://en.wikipedia.org/wiki/Bluetooth_Low_Energy']
[*] Category: Technology=ble|Interface=radio|Action=analysis
[*] Target: Name=generic|Version=generic|Vendor=generic
[*]
[*] Writing the value (0a01) to handle (0x42) on BLE device (ba:03:64:7c:16:e6)
[+] Test ble.generic.writechar passed
```

You can observe on the BLE watch that the Heart Rate monitor turns on after running the above command.

One of the interesting plugins in the EXPLIoT framework allows you to fuzz the characteristic values and write to the BLE device. The plugin options are similar to `ble.generic.writechar`, with the addition that you can mark the byte(s) in the value that you want to fuzz as "xx" (for each byte that you want to fuzz).

1. *ef> run ble.generic.fuzzchar -a <BDADDR> -n <handle> -w <value_with_XX_markers>*
2. We can fuzz the watch by changing the last bytes. You should observe the watch restart with the time and date reset.
3. For homework, try experimenting with different characteristic values and observe if you can identify any abnormal behavior such as device crashing, rebooting, or entering DFU (Device Firmware Update) mode.

```
ef> run ble.generic.fuzzchar -a ba:03:64:7c:16:e6 -n 0x0042 -w 04001604160axxxx
[*] Test: ble.generic.fuzzchar
[*] Author: Arun Magesh
[*] Author Email: arun.m@payatu.com
[*] Reference(s): ['https://en.wikipedia.org/wiki/Bluetooth_Low_Energy']
[*] Category: Technology=ble|Interface=radio|Action=fuzz
[*] Target: Name=generic|Version=generic|Vendor=generic
[*]
[*] Fuzzing the value (04001604160axxxx), iterations (255) for handle (0x42) on BLE device (ba:03:64:7c:16:e6)
[?] Writing the fuzzed value (04001604160a60d0)
[?]
[?] Writing the fuzzed value (04001604160a71a4)
[?]
[?] Writing the fuzzed value (04001604160a6dbc)
[?]
[?] Writing the fuzzed value (04001604160ac3aa)
[?]
[?] Writing the fuzzed value (04001604160ae8a9)
[?]
[?] Writing the fuzzed value (04001604160acd11)
[?]
[?] Writing the fuzzed value (04001604160a8c3a)
[?]
[?] Writing the fuzzed value (04001604160a9107)
[?]
[?] Writing the fuzzed value (04001604160a6a57)
[?]
```

Lab 6 – Crack BLE LTK

Objective:

Crack the LTK (Long Term Key) and decrypt the communication. The objective is to understand the process of cracking the LTK and the data required for cracking it.

Description:

You will be able to crack the LTK and use the found LTK to decrypt communication packets. We will use the crackle utility for this lab. If it's not already installed on your machine, you will need to install it.

Source: (<https://github.com/mikeryan/crackle>)

Setup:

1. For this lab, we will use sample PCAP files provided along with crackle.
2. In a real-world scenario, you will need a Bluetooth sniffer such as Ubertooth and a target BLE device.

Steps:

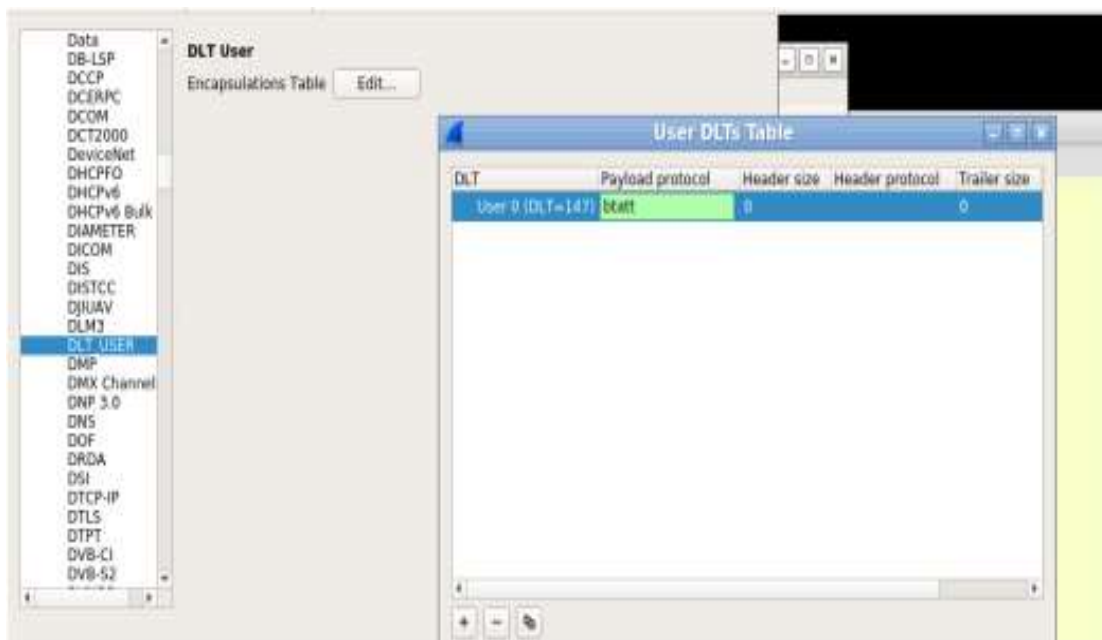
1. We have a sample packet capture of LTK exchange (**ltk_exchange.pcap**) between two devices. In a real-world scenario, you would need to sniff the communication.
2. Run the crackle utility on the **ltk_exchange.pcap** file to calculate/find the LTK from the capture.
 1. ***\$ crackle -i ltk_exchange.pcap -o bar.pcap***

3. Once we obtain the LTK, use it to decrypt the encrypted communication in the pcap file ``encrypted_known_ltk.pcap``.

1. `$ crackle -l 7f62c053f104a5bbe68b1d896a2ed49c -i encrypted_known_ltk.pcap -o decrypted.pcap`

```
exos@eXplIoT:~/Desktop/iot_course_bhusa/labs/protocols-ble-lab-6$ crackle -l 7f62c053f104a5bbe68b1d896a2ed49c -i encrypted_known_ltk.pcap -o abc.pcap
Warning: packet is too short to be encrypted (1), skipping
Warning: packet is too short to be encrypted (2), skipping
Warning: could not decrypt packet! Copying as is..
Warning: could not decrypt packet! Copying as is..
Warning: could not decrypt packet! Copying as is..
Warning: invalid packet (length too long), skipping
Done, processed 297 total packets, decrypted 7
```

4. Open the `decrypted.pcap` file in Wireshark as sudo and go to **Edit -> Preference -> Protocols -> DLT_USER -> Edit -> “+” symbol**. Add a parameter with `“btatt”` and press **OK**. Don’t press enter.



5. You should now be able to see all the decrypted ATT packed.

Note: Not all packets may be decrypted.

ZigBee (802.15.4) With One DIVA

The DIVA board contains a built-in Zigbee interface used for 802.15.4 communication. The Zigbee protocol operates on top of the 802.15.4 standard. Although DIVA does not implement pure Zigbee protocol communication, it provides a good opportunity to become familiar and comfortable with Zigbee tools and techniques such as scanning and replay.

Pre-requisite - Setup 1 DIVA Module for ZigBee

If you have one DIVA Board, follow the process below to set it up as a sender for the first few labs, and then configure it as a receiver for the subsequent labs.

Objective:

To set up the DIVA board for ZigBee (802.15.4) communication

Description: You will set up the PAN ID and device address on the DIVA board for Zigbee (802.15.4) communication.

1. Open the DIVA board console using **screen** or any other serial terminal utility.
2. In the DIVA console, enter option 6 (802.15.4 Lab).

```
diva>6
Welcome to 802.15.4 Lab

This Lab requires 2 boards, one needs to be configured as sender, second as receiver

Follow the wizard to configure the boards
Device address: ACDE480000000001
Enter 0 to configure as sender, 1 for receiver
```

3. SENDER: To make DIVA a sender for transmitting packets

1. Type 0 and hit enter.
2. When prompted for the PAN ID, enter a 4-digit hexadecimal PAN ID (2 bytes). Share this PAN ID with the receiver DIVA or your partner using the other DIVA (it should be unique for each pair of DIVA boards and within the range 0000 to FFFF).
3. When prompted for the "receiver address", enter the device address of the receiver DIVA board. Do not specify the sender device address here.

To find the device address on the DIVA Board:

- Enter the "802.15.4 Lab" on the DIVA console.

- It will display information including the 16-digit hexadecimal (8 bytes) device address of the Zigbee module on the DIVA board.
4. When prompted for "Data to send", enter a small string. For example: EXPLIoT.
 5. The console will then display the configuration and prompt to start or stop. Press 0 to start sending the data.

Note: We are using the short address, and by default, the short address of every device is set to 0x0000.

```

Enter 0 to configure as sender, 1 for receiver
diva>>0
Enter the PAN ID [3332]
diva>>3332
Enter the receiver address [0000]
diva>>0000
Enter the data to send
diva>>Exploit!!
configuration
=====
type           :0
pan id         :3332
Receiver address :0000
Data           :Exploit!!
Device configuration saved
Enter 1 to start, 0 to stop
diva>>1
Started
diva>>

```

4. RECEIVER: To make DIVA a receiver for receiving packets

1. On the DIVA console, press 6 (802.15.4 Lab).
2. Type 1 and hit Enter for receiver.
3. When prompted for the PAN ID, enter the same 4-digit hexadecimal PAN ID (2 bytes) as specified for the sender. PAN IDs must be unique for each pair of DIVA boards and should be within the range 0000 to FFFF.
4. It will then display the configuration and prompt for start/stop. Press 1 to start receiving the data.

```

Enter 0 to configure as sender, 1 for receiver
diva>>1
Enter the PAN ID[3332]
diva>>3332
configuration
=====
type           :1
pan id         :3332
Device configuration saved
Enter 1 to start, 0 to stop
diva>>1
Started
diva>>

```

5. Once both the sender and receiver are configured to send and receive data continuously, you will notice that the receiver starts printing the received data in a loop, indicating continuous reception from the sender. Now, you can proceed with the labs.

```
Enter 1 to start, 0 to stop
diva>>1
Started
diva>>Explot!!
diva>>
Explot!!
diva>>
Explot!!
diva>>
Explot!!
diva>>
Explot!!
```

Lab 1 – ZigBee Network Discovery

Objective:

Reconnaissance with Zigbee Auditor. The objective is to learn how to scan, identify the PAN network, and determine the channel number it uses.

Description:

The attendees will learn to perform reconnaissance on the Zigbee network to discover the PAN ID and channel.

Setup:

1. DIVA setup for sending Purpose
2. Zigbee Auditor
3. EXPLIoT Framework

Steps:

1. Power up the DIVA board and set it up as a sender as discussed above.
2. Plug in Zigbee Auditor to another USB interface on your machine.
3. Open the EXPLIoT Framework and run the Zigbee Auditor Device Information Plugin to determine the firmware and hardware version of Zigbee Auditor.
 1. *ef> run zbauditor.generic.deviceinfo*

```
ef> run zbauditor.generic.devinfo
[*] Test:          zbauditor.generic.devinfo
[*] Author:        Dattatray Hinge
[*] Author Email:  dattatray@exploit.io
[*] Reference(s):  ['Reference: Zigbee Auditor user manual']
[*] Category:      Technology=zbauditor|Interface=radio|Action=recon
[*] Target:        Name=generic|Version=generic|Vendor=generic
[*]
[+] device_name: ZigBee Auditor
[+] fw_revision: 1.0.1
[+] serial_number: DF39C7719876
[+] services:
[+]   read_revision: True
[+]   read_services: True
[+]   channel_selection: True
[+]   radio_on_off: True
[+]   802.15.4_sniffer: True
[+]   802.15.4_injection: True
[+]   802.15.4_network_scan: True
[+]   2400_mhz: True
[+]
[+] Test zbauditor.generic.devinfo passed
```

4. Now scan the Zigbee network for available networks. Run the plugin:

1. ***ef> run zbauditor.generic.nwscan***

```
ef> run zbauditor.generic.nwscan
[*] Test:          zbauditor.generic.nwscan
[*] Author:        Dattatray Hinge
[*] Author Email:  dattatray@exploit.io
[*] Reference(s):  ['https://www.zigbee.org/wp-content/uploads/2014/11/docs-05-3474-20-0csg-zigbee-specification.pdf']
[*] Category:      Technology=zbauditor|Interface=radio|Action=discovery
[*] Target:        Name=generic|Version=generic|Vendor=generic
[*]
[*] Start channel: (11)
[*] End channel : (26)
[*]
[+]
[+] Devices found      1
[+] Device Number      : 1
[+] Channel            : 12
[+] Source Address     : 0x0
[+] Source PAN ID      : 0x1212
[+] Pan Coordinator    : False
[+] Permit Joining     : False
[+] LQI                : 224
[+] RSSI               : -36
[*]
[+] Scan duration      6.537467956542969
[*]
[+] Test zbauditor.generic.nwscan passed
```

5. You should be able to see the device found, device number, channel, PAN ID, and other information in the output.

6. Identify the channel number used by your PAN ID network. This information is required for further labs.
7. Zigbee Auditor enumerates all Zigbee channels and displays the devices found on those channels.

Lab 2 – Eavesdropping

Objective:

Eavesdropping with Zigbee Auditor. The objective is to understand the tools and process of sniffing Zigbee communication.

Description:

You will learn how to perform eavesdropping with Zigbee Auditor and Wireshark tools to capture and analyze data for further labs. Zigbee communication is encrypted using AES encryption. We will discuss decryption in the final lab.

Setup:

1. DIVA setup as a sender
2. Zigbee Auditor
3. EXPLIoT Framework

Steps:

1. Power up the DIVA board and set it up as a sender, as discussed above.
2. Plug Zigbee Auditor into another USB interface on your machine.
3. Open the EXPLIoT Framework and run the `zbauditor.generic.sniffer` plugin. Specify the channel number to sniff and the file name to store the capture (it saves it in .pcap format).

Note: Use the channel number you discovered in the previous lab using Zigbee Auditor for your PAN ID.

1. ***ef> run zbauditor.generic.sniffer***
2. Example command for sniffing on channel 12 and storing the capture in `test.dump` file:

ef>run zbauditor.generic.sniffer -c 12 -f /home/test.dump -t 10

Note: “-c 12” is channel number 12.

```
ef> run zbauditor.generic.sniffer -c 12 -f /home/test.dump -t 10
[*] Test: zbauditor.generic.sniffer
[*] Author: Dattatray Hinge
[*] Author Email: dattatray@exploit.io
[*] Reference(s): ['https://www.zigbee.org/wp-content/uploads/2014/11/docs-05-3474-20-0csg-zigbee-specification.pdf']
[*] Category: Technology=zbauditor|Interface=radio|Action=recon
[*] Target: Name=generic|Version=generic|Vendor=generic
[*]
[*] Channel : (12)
[*] File : (/home/test.dump)
[*] Count : (65535)
[*] Time-Out : (10)
[+] packets_received: 28
[+]
[+] Test zbauditor.generic.sniffer passed
```

4. After the specified time interval, the plugin will stop automatically, as shown in the image above.

Lab 3 – Replay Attack

Objective:

Replay packet captures with Zigbee Auditor. The objective is to understand the plugin and the process of replaying Zigbee packets.

Description:

You will learn to replay packets over the Zigbee network. Replay the same .pcap file captured in the lab above. You will need to set up DIVA as a receiver to receive the replayed packets from Zigbee Auditor. For details on how to set up DIVA as a receiver, refer to the "RECEIVER: To make DIVA a receiver for receiving packets" section above.

Setup:

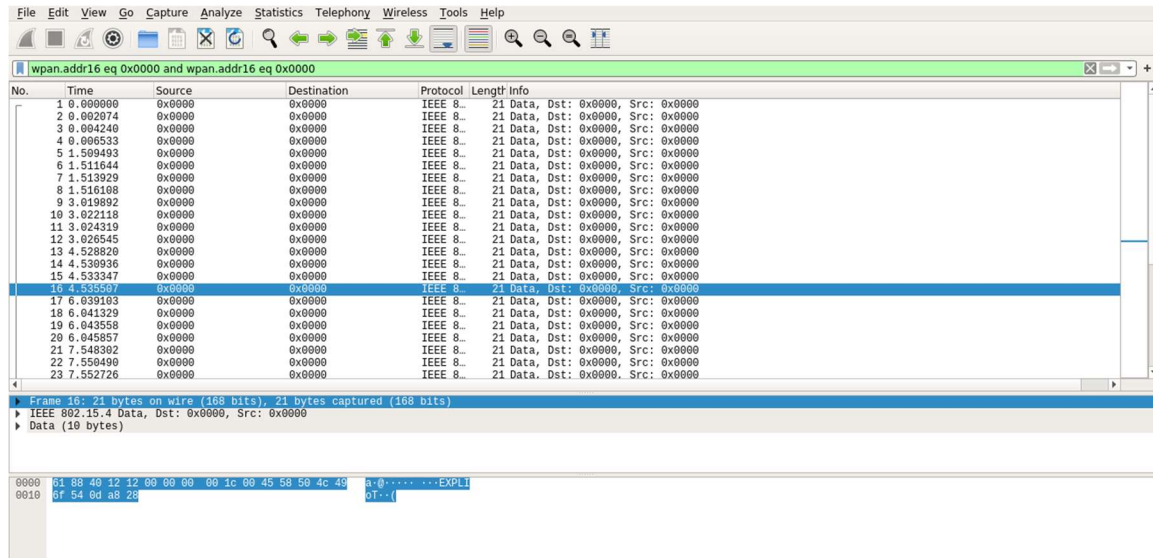
1. DIVA setup a receiver
2. Zigbee Auditor
3. EXPLIoT Framework

Steps:

1. Make sure:
 - a. Zigbee Auditor is plugged into your machine.
 - b. DIVA is still in receiving mode, waiting for the replayed packets to be received.

- Open the dump file from Lab 2 (Eavesdropping Lab) in Wireshark and filter the communication to see only your setup packets using the below filter:

- \$ sudo wireshark filename.dump**
- Filter: `wpan.dst16 == XX:XX`**
- Replace the XX... above with the receiver address you specified when you set up DIVA as the sender (make sure to add a colon ":" after each hex byte in the address).

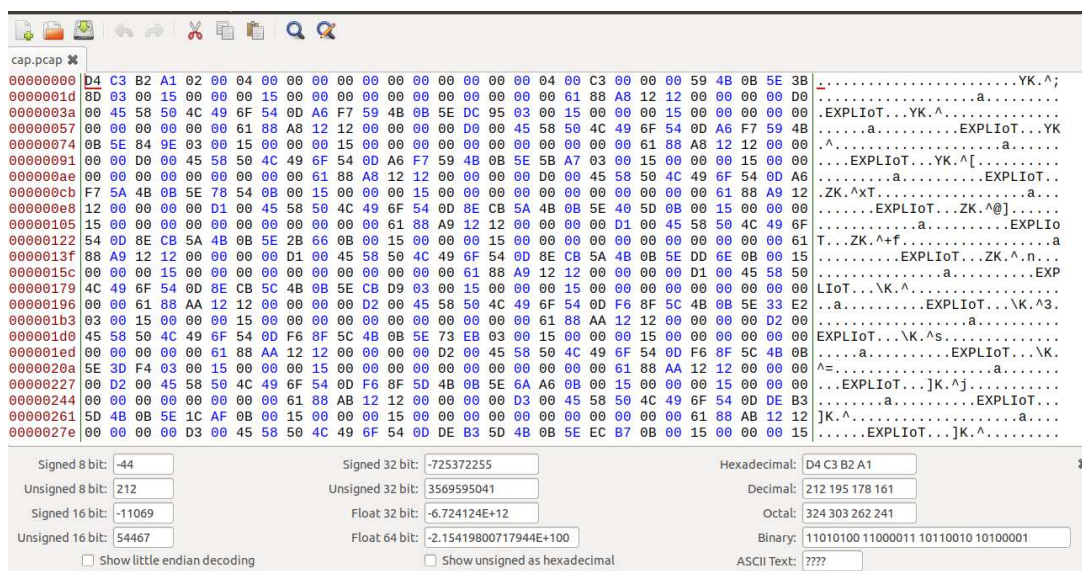


- To save only the filtered packets, go to File and select Export Specified Packets. Save them to a capture file such as *cap.pcap*.
- If you want to forge or manipulate the data you're sending, you can also modify it in the PCAP file. Note that manually editing data for actual ZigBee communication won't work due to encryption. Open the PCAP file in Bless to view the transmitted data. You can then edit these strings and save the PCAP file. In our example (image), we're replacing 'EXPLIoT' with 'HACKERR'. Ensure you change only the specified text.

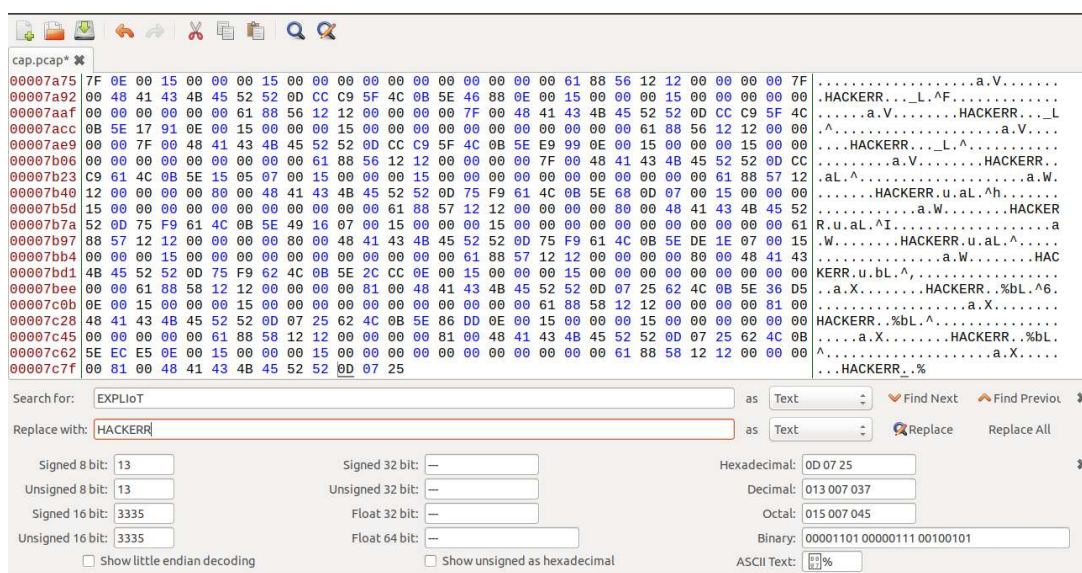
5. Note:

- Change only the parts of the string that your sender was originally sending. If you alter anything else, the packet may become corrupted.
- If the PCAP file requires root permissions, run Bless with `sudo` or change the file permissions to allow access by a non-root user.

Example PCAP file is shown below:



The updated PCAP file is shown below (changed “EXPLIoT” to “HACKERR”):



6. Plug Zigbee Auditor into another USB interface on your machine.
7. Run the EXPLIoT Framework and the *zbauditor.generic.reply* plugin. Specify the channel number along with the PCAP file (that you saved earlier) to be replayed.

```
ef> run zbauditor.generic.replay -c 12 -f /home/dattatray/test_reply.dump
[*] Test: zbauditor.generic.replay
[*] Author: Dattatray Hinge
[*] Author Email: dattatray@exploit.io
[*] Reference(s): ['https://www.zigbee.org/wp-content/uploads/2014/11/docs-05-3474-20-0csg-zigbee-specification.pdf']
[*] Category: Technology=zbauditor|Interface=radio|Action=exploit
[*] Target: Name=generic|Version=generic|Vendor=generic
[*]
[*] Channel : (12)
[*] File : (/home/dattatray/test_reply.dump)
[*] Delay (seconds): (0.2)
[*]
[+] packets_received: 0
[+] packets_transmitted: 28
[+]
[+] Test zbauditor.generic.replay passed
```

ef> run zbauditor.generic.reply -c 12 -f <file path>

8. It will re-transmit each frame from the PCAP file, excluding acknowledgment frames.
9. Confirm that the packets are being received by the receiver. The receiver should be in receiving mode, and if the replay was successful, you should see the data on the receiver's DIVA console.

```
diva>>
HACKERR
diva>>
HACKERR
diva>>
HACKERR
diva>>
HACKERR
```

ZigBee (802.15.4) With Two DIVA:

Pre-requisite - Setup 2 DIVA Modules for ZigBee

If you have 2 DIVA modules or if there are two people with one DIVA each, use the below process to set up one as sender and the other as receiver.

Objective:

To set up the DIVA board for ZigBee (802.15.4) communication

Description: The attendees will learn how to set up the PAN ID and device address on the DIVA board for ZigBee (802.15.4) communication.

1. Open the DIVA board console using **screen** or any other serial terminal utility. Instructions on how to use **screen** and check if DIVA is connected are discussed in the hardware labs above.
2. In the DIVA console, enter 6 (802.15.4 Lab).

```
diva>6
Welcome to 802.15.4 Lab

This Lab requires 2 boards, one needs to be configured as sender,second as receiver

Follow the wizard to configure the boards
Device address: ACDE480000000001
Enter 0 to configure as sender, 1 for receiver
```

3. Select one DIVA board to act as sender and the other as receiver.
4. **SENDER: To make one DIVA a sender for transmitting packets**
 1. Type 0 and hit enter.
 2. When prompted for the PAN ID, choose and enter a 4-digit hexadecimal (2-byte) PAN ID. Share this PAN ID with the receiver DIVA or your partner's DIVA (it should be unique for each pair of DIVA boards and within the range 0000 to FFFF).
 3. When prompted for the 'receiver address', enter the device address of the receiver DIVA board. Please DO NOT specify the sender device address here.
 4. To find the device address on the DIVA Board, access the '802.15.4 Lab' on the DIVA console. It will print information including the 4-digit hexadecimal (2-byte) device address of the ZigBee module on the DIVA board.

5. When prompted for 'Data to send', enter any small string. For example: EXPLIoT.
6. The configuration will then be displayed, and you will be prompted to start or stop. Press 0 to start sending the data.

```

Enter 0 to configure as sender, 1 for receiver
diva>>0
Enter the PAN ID [3332]
diva>>1122
Enter the receiver address [0000]
diva>>0000
Enter the data to send
diva>>EXPLIoT
configuration
=====
type                :0
pan id              :1122
Receiver address    :0000
Data                :EXPLIoT
Device configuration saved
Enter 1 to start, 0 to stop
diva>>

```

5. RECEIVER: To make another DIVA a receiver for receiving packets

1. On the DIVA console, press 6 (802.15.4 Lab).
2. When prompted for the PAN ID, enter the same 4-digit hexadecimal (2-byte) PAN ID as specified on the sender. PAN IDs must be unique for each pair of DIVA boards and within the range 0000 to FFFF.
3. The console will then display the configuration and prompt for start/stop. Press 1 to begin receiving the data.

```

Follow the wizard to configure the boards
Enter 0 to configure as sender, 1 for receiver
diva>>1
Enter the PAN ID[1122]
diva>>1122
configuration
=====
type                :1
Device configuration saved
Enter 1 to start, 0 to stop
diva>>1
Started
diva>>

```

6. Once both the sender and receiver are configured to send and receive data continuously, you will notice that the receiver starts printing the received data in a loop, indicating continuous reception from the sender. Now, you can proceed with the labs.

```
diva>>1
Started
diva>>EXPLIOT
diva>>EXPLIOT
diva>>
EXPLIOT
diva>>
EXPLIOT
diva>>
EXPLIOT
diva>>
EXPLIOT
```

Lab 1 – ZigBee Network Discovery

Note:

This lab is the same as the one in the section “ZigBee (802.15.4) With One DIVA”.

Lab 2 – Eavesdropping

Note:

This lab is identical to the one in the section 'ZigBee (802.15.4) With One DIVA'. The only requirement is to ensure that the sender is transmitting, and the receiver is receiving while you perform this lab.

Lab 3 – Replay Attack

Note:

This lab is identical to the one in the section 'ZigBee (802.15.4) With One DIVA'. The only difference in this lab is that you will need to stop the Sender DIVA so that no packets are being transmitted while you replay the packets using Zigbee Auditor.

Lab 4 – Decrypt ZigBee Communication

Objective:

Decrypt the packet capture from a home security system. The objective is to understand the default encryption in ZigBee and learn how to decrypt ZigBee packets using Wireshark.

Description:

You will learn to use the default Zigbee Link key to first decrypt the Network key and then use the Network key to decrypt any further communication. You will also perform a replay attack. The ZigBee standard specifies a Global Link Key, '**ZigBeeAlliance09**,' for compatibility, and many vendors use this key in their products, which makes the network vulnerable to eavesdropping and replay attacks.

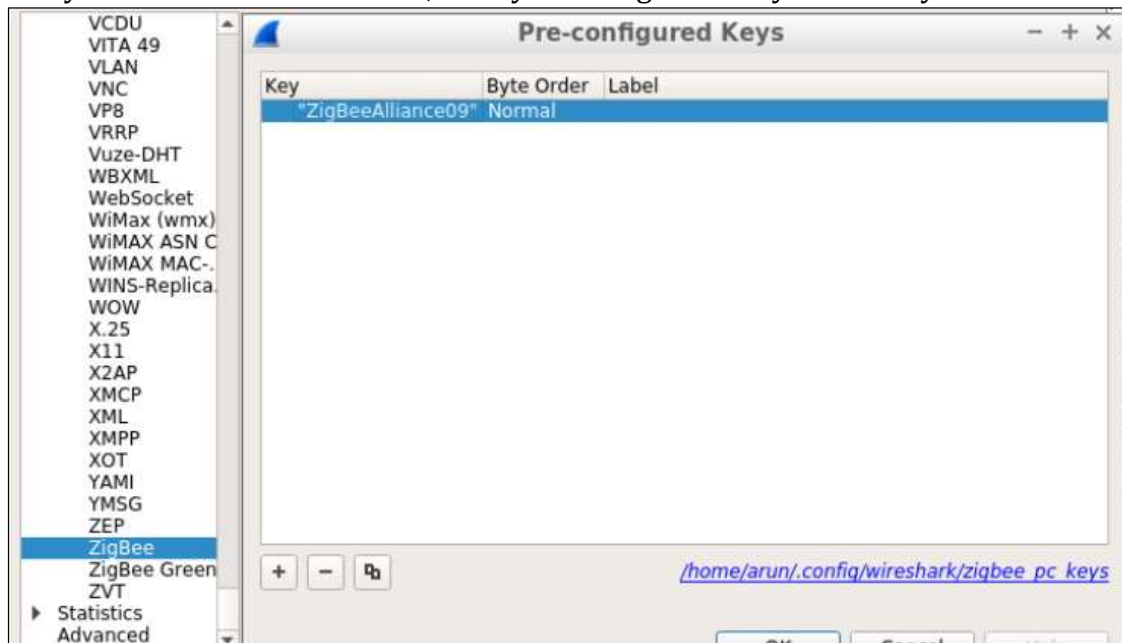
Setup:

1. MI Home Kit communication PCAP files
2. Zigbee Auditor
3. Wireshark

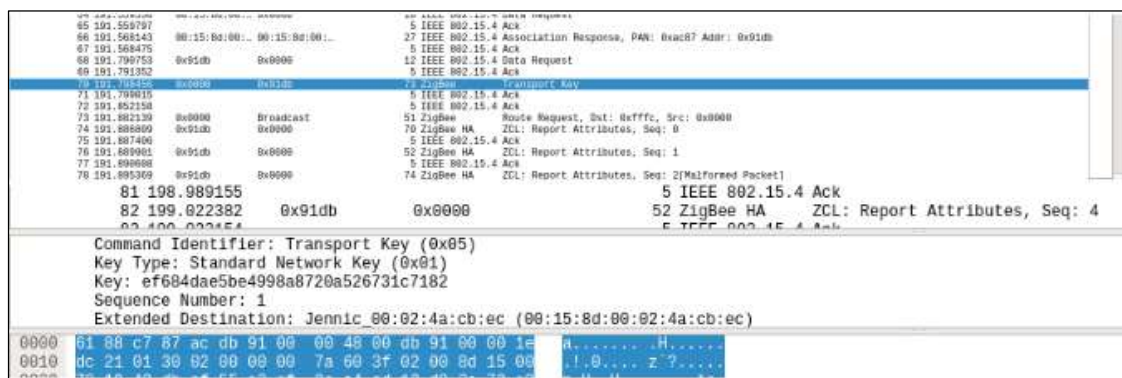
Steps:

1. Two PCAP files for this lab are provided in the labs directory.
 1. ***mi_pairing.pcap*** – This PCAP file contains the communication of a device joining the network.
 2. ***midencrypt.pcap*** – This PCAP file contains general communication between the ZigBee Controller and the ZigBee device including the commands sent etc.
2. Start Wireshark, open the file "*mi_pairing.pcap*" in it and you will see that the data payloads are encrypted.
3. Now go to **Edit -> Preference -> Protocol -> ZigBee -> Edit (Pre-Configured Keys)** and add "**ZigBeeAlliance09**" (Including the double quotes; for ASCII string, Wireshark requires us to enclose it in double quotes) and save

it. Byte Order will be Normal, and you can give it any label of your choice.



- Now you will see that some packets are decrypted, while others are not. In the decrypted section, you will notice packets related to the device joining the network, such as 'Association Request', 'Association Response', 'Data Request', and finally the **'Transport Key'** packet. The Transport Key packet is used to transfer the Network Key to the device. Click on the Transport Key packet, locate the network key in the middle tab, and copy it – **Right click (on the key in the middle tab) -> Copy -> As Hex Stream**.



- Now put the copied **Network key** in the pre-configured key list as discussed above for the **Global Link Key**. Again, byte order will be Normal, and you can Label it with any name of your choice.

VLAN	
VNC	
VPB	
VRRP	
Vuze-DHT	
WBXML	
WebSocket	
WiMax (wmx)	
WiMAX ASN C	

Key	Byte Order	Label
"ZigBeeAlliance09"	Normal	
ef684dae5be4998a8720a526731c7182	Normal	

- You will now be able to decrypt further communication using the same network key.
- Now open *"midecrypt.pcap"* and analyze all the communications. You should now be able to see commands and responses.

30 20.825837	0xb223	0x0000	52 ZigBee HA	ZCL: Report Attributes, Seq: 97
31 20.826549			5 IEEE 802.15.4 Ack	
32 21.200050	0xb223	0x0000	52 ZigBee HA	ZCL: Report Attributes, Seq: 98

Cluster: On/Off (0x0006)				
Profile: Home Automation (0x0194)				
Source Endpoint: 1				
Counter: 189				
ZigBee Cluster Library Frame, Command: Report Attributes, Seq: 92				
Frame Control Field: Profile-wide (0x18)				
Sequence Number: 92				
Command: Report Attributes (0x0a)				
Attribute Field				
Attribute: On/Off (0x0006)				
Data Type: Boolean (0x10)				
Data Value: On (0x01)				

0000	61 88 5d 87 ac 00 00 23 b2 48 02 00 00 23 b2 1e	a . . . H . .
0010	25 28 5d 01 00 00 ec cb 4a 02 00 8d 15 00 01 f3	N (. . . j . . .
0020	86 88 19 8c 53 93 3e 91 1b 6a b9 46 50 92 23 bd	r . S . > j . F P . #
0030	00 9a 1f 01	a . . .

- You can save the specific packets that have **'off'** or **'on'** attributes and perform the replay exercise as in the Replay Lab, using the MI Home Kit (only for physical training attendees). Hint: Use PAN ID or destination address in the filter. Note: Make sure to save in .pcap format and not .pcapng.

Glossary

1. DMM – Digital Multimeter
 1. [https://en.wikipedia.org/wiki/Multimeter#Digital_multimeters_\(DMM_or_DVOM\)](https://en.wikipedia.org/wiki/Multimeter#Digital_multimeters_(DMM_or_DVOM))
2. UART - Universal Asynchronous Receiver Transmitter
 1. https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter
3. BLE – Bluetooth Low Energy
 1. https://en.wikipedia.org/wiki/Bluetooth_Low_Energy
4. JTAG – Joint Test Action Group
 1. <https://en.wikipedia.org/wiki/JTAG>
5. I2C – Inter Integrated Circuit
 1. <https://en.wikipedia.org/wiki/I²C>
6. SPI – Serial Peripheral Interface
 1. https://en.wikipedia.org/wiki/Serial_Peripheral_Interface
7. CPOL - Clock Polarity
 1. https://en.wikipedia.org/wiki/Serial_Peripheral_Interface#Clock_polarity_and_phase

8. CPHA - Clock Phase

1. https://en.wikipedia.org/wiki/Serial_Peripheral_Interface#Clock_polarity_and_phase

9. LCD – Liquid Crystal Display

1. https://en.wikipedia.org/wiki/Liquid-crystal_display

10. VCC – Voltage Common Collector

1. https://en.wikipedia.org/wiki/IC_power-supply_pin

11. GND – Ground

1. [https://en.wikipedia.org/wiki/Ground_\(electricity\)](https://en.wikipedia.org/wiki/Ground_(electricity))

12. TX – Transmission

1. [https://en.wikipedia.org/wiki/Transmission_\(telecommunications\)](https://en.wikipedia.org/wiki/Transmission_(telecommunications))

13. RX – Receive

1. <https://en.wikipedia.org/wiki/Rx>

14. DIP – Dual In-line Package

1. https://en.wikipedia.org/wiki/Dual_in-line_package

15. PDIP – Plastic Dual In-line Package

1. https://en.wikipedia.org/wiki/Dual_in-line_package

16. SOIC - Small Outline Integrated Circuit

1. https://en.wikipedia.org/wiki/Small_outline_integrated_circuit

17. EEPROM - Electrically Erasable Programmable Read-Only Memory

1. <https://en.wikipedia.org/wiki/EEPROM>

18. PCB - Printed Circuit Board

1. https://en.wikipedia.org/wiki/Printed_circuit_board

19. FTDI - Future Technology Devices International

1. <https://en.wikipedia.org/wiki/FTDI>

20. HCI - Host Controller Interface

1. [https://en.wikipedia.org/wiki/List_of_Bluetooth_protocols#Host_Controller_Interface_\(HCI\)](https://en.wikipedia.org/wiki/List_of_Bluetooth_protocols#Host_Controller_Interface_(HCI))

21. PCAP - Packet Capture

1. <https://en.wikipedia.org/wiki/Pcap>

22. GATT - Generic Attribute Profile

1. [https://en.wikipedia.org/wiki/List_of_Bluetooth_profiles#Generic_Attribute_Profile_\(GATT\)](https://en.wikipedia.org/wiki/List_of_Bluetooth_profiles#Generic_Attribute_Profile_(GATT))

23. UUID - Universally unique identifier

1. [https://en.wikipedia.org/wiki/Universally unique identifier](https://en.wikipedia.org/wiki/Universally_unique_identifier)

24. 802.15.4 - IEEE 802.15.4 Standard

1. [https://en.wikipedia.org/wiki/IEEE 802.15.4](https://en.wikipedia.org/wiki/IEEE_802.15.4)

Warranty and Returns:

1. Returns are only allowed for Dead on Arrival (DOA) products or if the wrong item was shipped. In both cases, we will provide a replacement.
2. A product is considered DOA if it shows symptoms of hardware failure preventing basic operability upon its initial use outside of the box.
3. Repaired or modified products are not eligible for refund or replacement.
4. Returns should be initiated within 5 calendar days of the item arriving at the customer's address.
5. Due to the nature and usage of our products, we do not provide any warranty unless it is explicitly mentioned on the product page.