Taking D-Bus to Explore the Bluetooth Landscape

Paul A. Wortman, PhD
Mauddib28

May 18, 2024

Table of contents

Whoami

- PhD
- Bluetooth Security Researcher
- Research Scientist

Whatisthis?

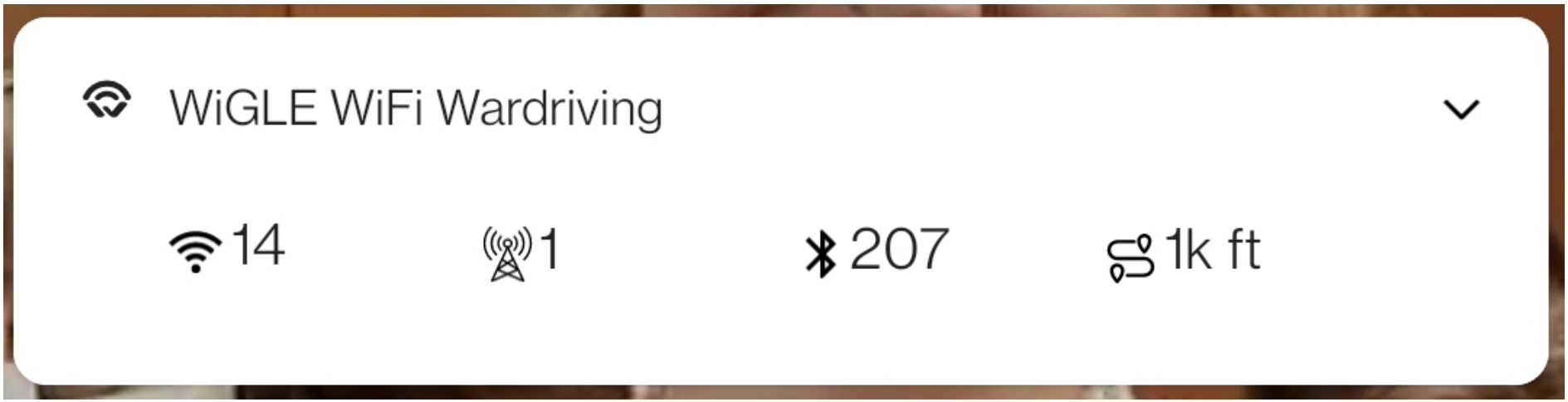


Figure 1: Wiggle War Drive - Landing Strip

What is this about?

- Surveying the Bluetooth Landscape
- Improve Bluetooth Wildlife Observation
- Augment Research Community Access to Bluetooth

BR/ED vs BLE

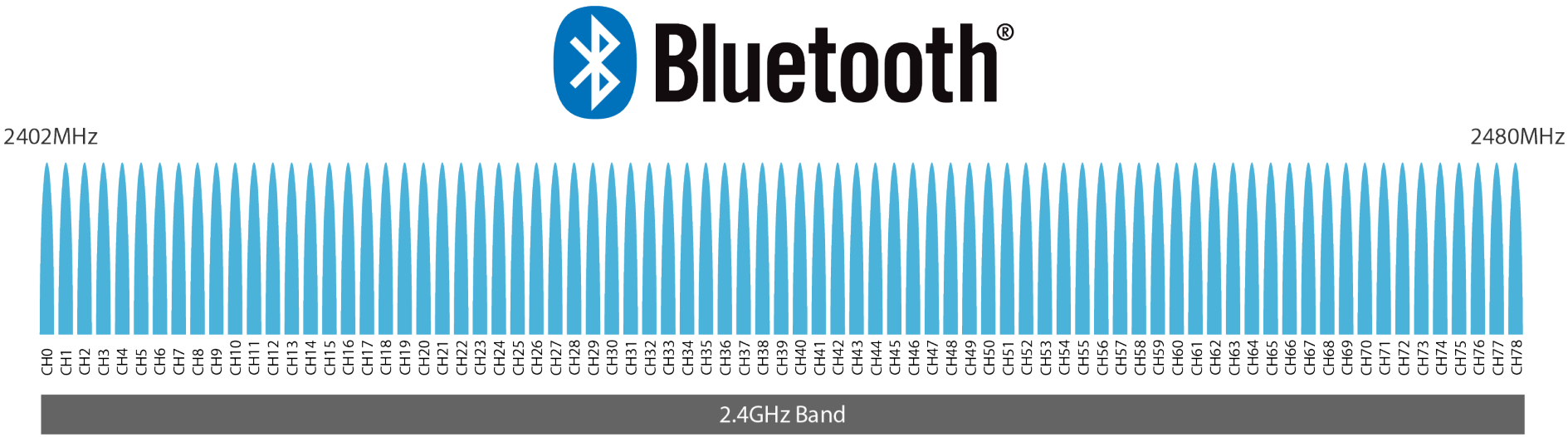


Figure 2: Bluetooth Classic Spectrum

- 79 number of (pairing) channels
- Freq. hoping to minimize interference
  - Defense in depth
- Larger power requirements minimize effectiveness for embedded systems

BR/ED vs BLE

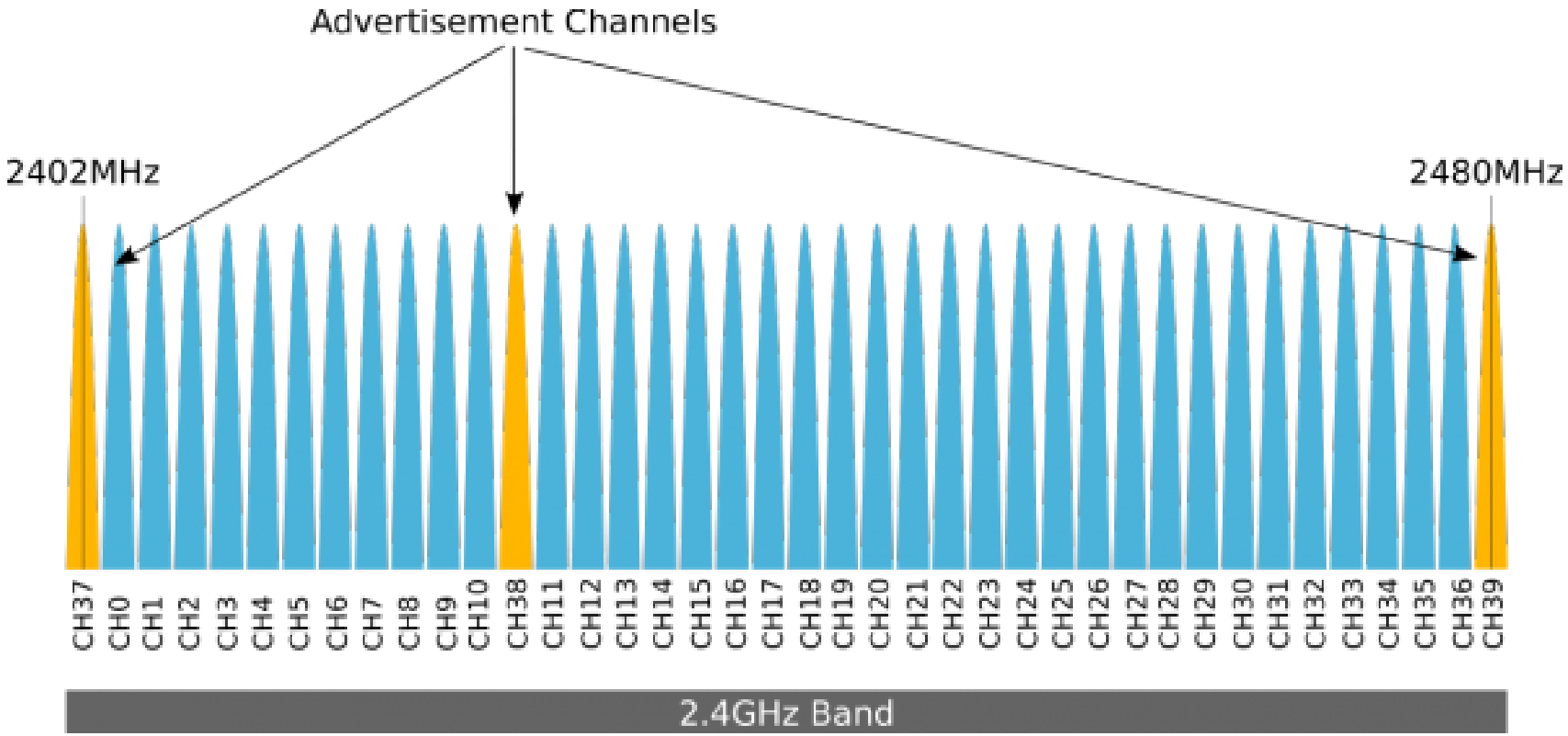


Figure 3: Bluetooth Low Energy Spectrum

- Three pairing channels; simplified sniffing of pairing
- Post connection return to classic frequency hopping
  - Capture of handshake provides easier eavesdropping
- Lower power requirements; ideal for embedded systems

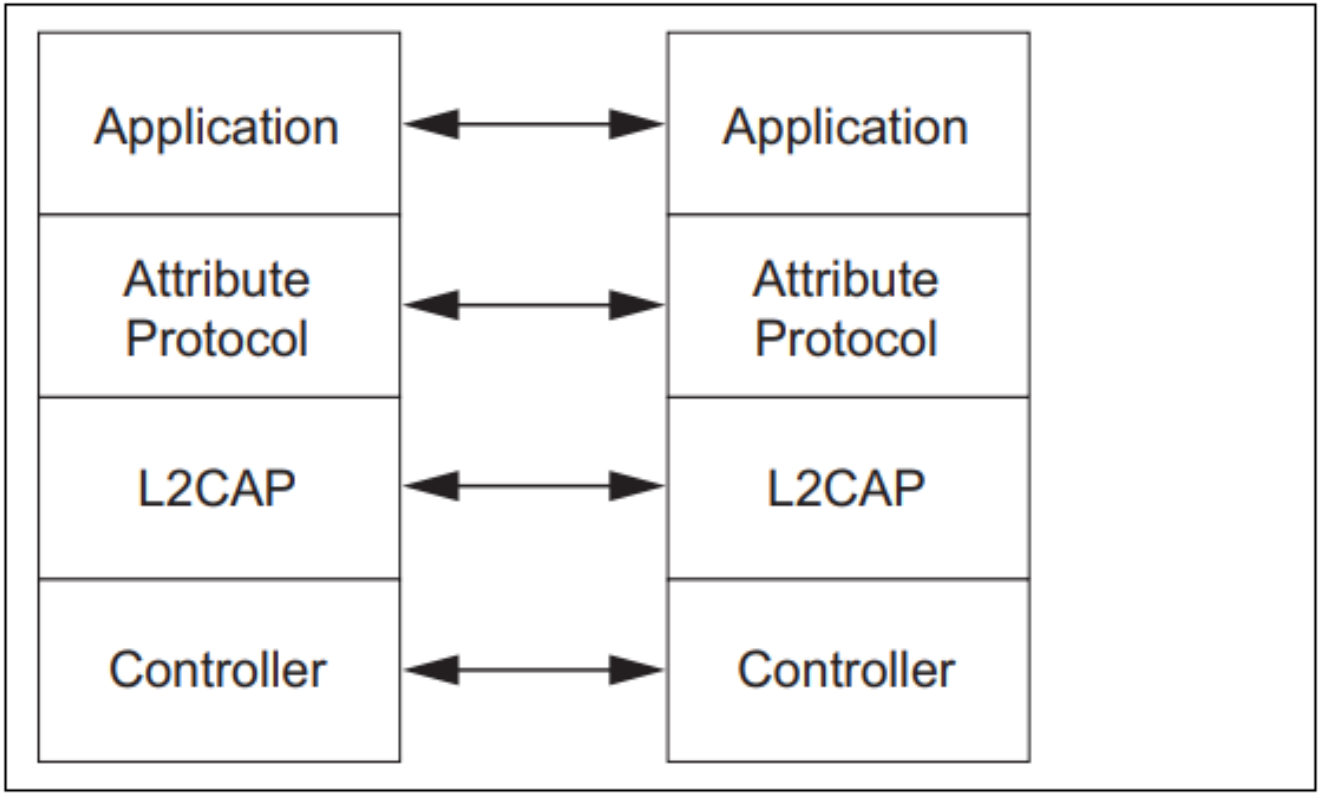


Figure 4: Protocol model [14]

- GATT Server
  - Inability to “change” a GATT server once initialized
  - Flavor differences; Arduino will not allow writing to Descriptors

BLE - Must Know Basics - High-level BLE Structure

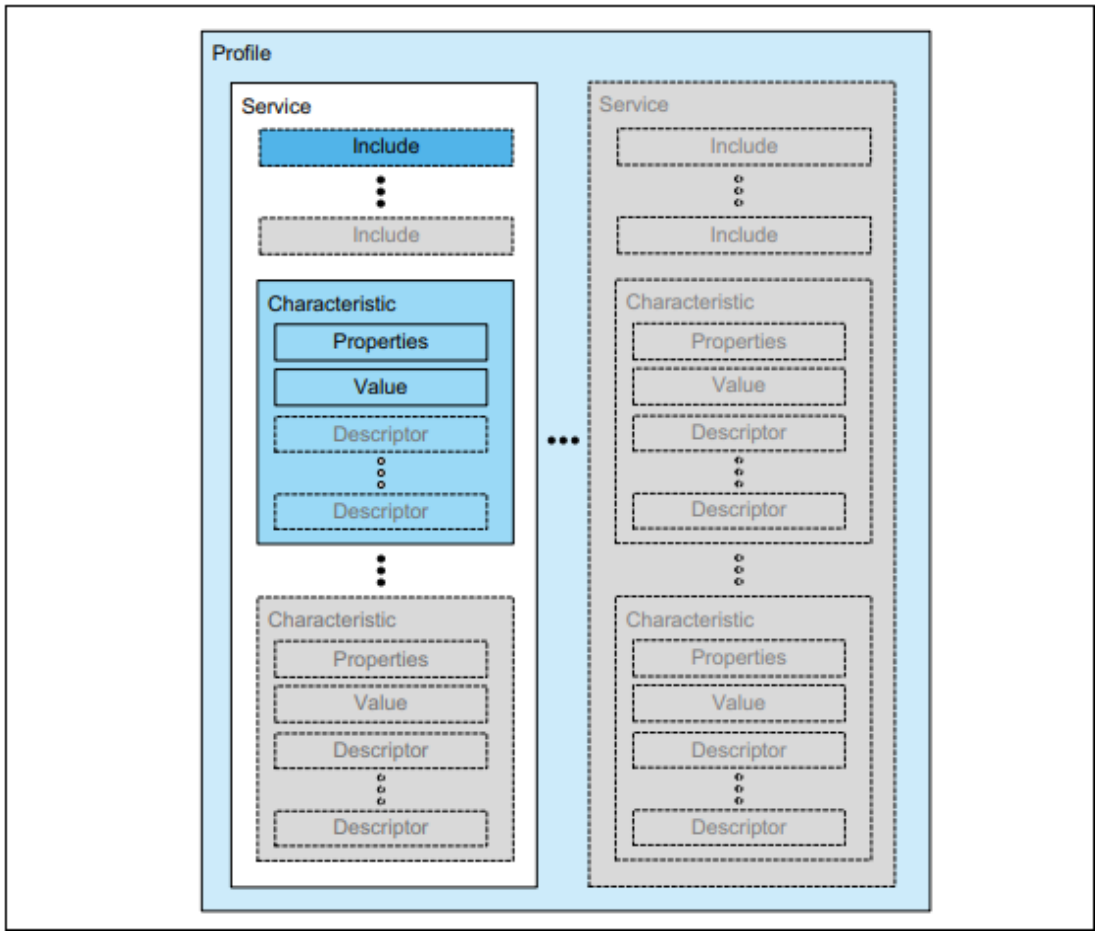


Figure 5: GATT-Based Profile hierarchy [14]

- Device, Services, Characteristics, Descriptors
- Characteristics have to be read once to populate data
  - Second read allows determining the data

Basics

- D-Bus developed and maintained by freedesktop.org
  - URL: www.freedesktop.org/wiki/Software/dbus
- D-Bus uses code agnostic low-level API reference implementation that works/portable to any Linux or UNIX flavor
  - Make use of Python API calls with tool
- D-Bus is a message bus system; providing a simple way for applications to talk to one another
  - Provides a system daemon and a per-user-login-session daemon [1]

Weighing D-Bus

- Pros vs Cons
  - Code agnostic API
  - Learning new tool set; from scratch
- Oooooo, What?!?
  - Appears to work as a network hub
  - Can share an internet connection via Bluetooth + D-Bus commands

Intent of Research

- Swiss-army knife tool
- Map the existing Bluetooth landscape
- Retain low-level granularity for I/O

Goal Tracking - Phz1

Research – Phase One - Goals	
Goal Posts	**Completion Metric**
Learn the PyBluez Python Library	✓
Create a basic device scanner; *Focus on Bluetooth Low Energy*	✓
Connect and Disconnect to/from a device	✓
Enumerate a device; *train on the BLE CTF*	✓
Establish basic Read and Write I/O capabilities	✓
Enumerate a second device; *Philips HUE BLE Light*	

Figure 6: Goal Table - Phase One - Expanded Goals

Goal Tracking - Phz1

Research – Phase One - Goals	
Goal Posts	**Completion Metric**
Learn the PyBluez Python Library	✓
Create a basic device scanner; *Focus on Bluetooth Low Energy*	✓
Connect and Disconnect to/from a device	✓
Enumerate a device; *train on the BLE CTF*	✓
Establish basic Read and Write I/O capabilities	✓
Enumerate a second device; *Philips HUE BLE Light*	X

[Encrypted Connection Failure]

Figure 7: Goal Table - Phase One - Unfortunate End

Intent of Research - Specifics

- Obtain the ability to:
  - Discover and connect to Bluetooth (BLE) devices
    - Expand to allow encryption
  - Identify properties and services accurately
  - Read + Write I/O
    - Arbitrary and directed
  - Wrap all learned capabilities into an easy to use User Interface (Classes and functions)

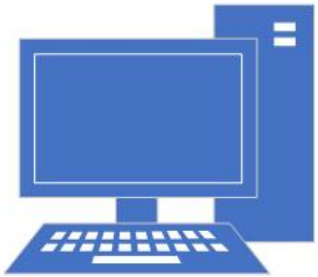
Where to Start?

Where to Start?

- Python for quick prototyping
  - Faster than C/C++
- Leverage the Linux D-Bus to provide improved granularity
- Relys on Bluez API documentation for interaction
  - Woe is the example-seeker

Starting Architecture

Wireless Rig:
Blue Hydra +
Interaction Platform



Wireless Rig

Sena UD-100

Bluetooth Classic +
Low Energy Devices

Sanity Check via Tools

- Learning the basics with tools
  - dbus-send, dbus-monitor, gdbus, busctl
  - btmon, btmgmt, bluetoothctl

Sanity Check via Tools - bluetoothctl

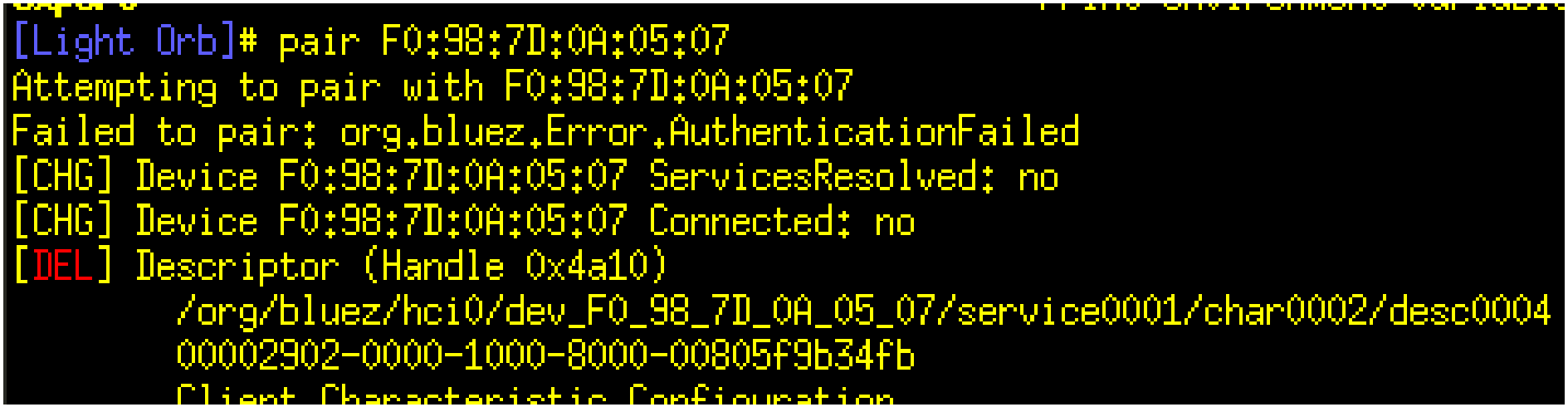


Figure 8: bluetoothctl - Light Orb Test

Goal Tracking - Phz2

Research – Phase Two - Goals	
Goal Posts	**Completion Metric**
Find replacement Python library for low level capability	✓
Learn Python D-Bus functionality	
Re-Create PyBluez code using D-Bus library functions	
Scan / Discover	
Connect / Disconnect	
Enumerate	
Read / Write I/O	
Create Classes for D-Bus & BLE structures	
Abstract the low level commands into higher level functions	
Create User Interface for all code functionality	
Allow for detailed enumeration of user selected devices	

Scan and Discover

- Scan for, discover, and identify potential devices
  - Begin to know GLib.MainLoop()
- While GLib.MainLoop() operates the discovery scanning continues

  - Ctrl-C terminated
  - If too long then addresses “no longer exist”

Scan and Discover

```
[duncan@ArtII BTInteract]$ python3 pybluez_device_basics.py
usage: python3 client_discover_devices.py [scantime (secs)]
[duncan@ArtII BTInteract]$ python3 pybluez_device_basics.py 60
Listing devices already known to BlueZ:
EXI path :  /org/bluez/hci0/dev_F0_98_7D_0A_05_07
EXI bdaddr:  F0:98:7D:0A:05:07
------------------------------
EXI path :  /org/bluez/hci0/dev_CC_50_E3_B6_BC_A6
EXI bdaddr:  CC:50:E3:B6:BC:A6
------------------------------
Found  2  managed device objects
Scanning
CHG path : /org/bluez/hci0/dev_F0_98_7D_0A_05_07
CHG bdaddr:  F0:98:7D:0A:05:07
CHG name :  Light Orb
CHG RSSI :  -27
------------------------------
CHG path : /org/bluez/hci0/dev_F0_98_7D_0A_05_07
CHG bdaddr:  F0:98:7D:0A:05:07
CHG name :  Light Orb
CHG RSSI :  -41
------------------------------
CHG path : /org/bluez/hci0/dev_F0_98_7D_0A_05_07
CHG bdaddr:  F0:98:7D:0A:05:07
CHG name :  Light Orb
CHG RSSI :  -31
------------------------------
Full list of devices 2 discovered:
------------------------------
F0:98:7D:0A:05:07
CC:50:E3:B6:BC:A6
```

Goal Tracking - Phz2

Research – Phase Two - Goals	
Goal Posts	**Completion Metric**
Find replacement Python library for low level capability	✓
Learn Python D-Bus functionality	65%
Re-Create PyBluez code using D-Bus library functions	15%
Scan / Discover	✓
Connect / Disconnect	
Enumerate	
Read / Write I/O	
Create Classes for D-Bus & BLE structures	35%
Abstract the low level commands into higher level functions	
Create User Interface for all code functionality	
Allow for detailed enumeration of user selected devices	

[Dis—C]onnection

- Connect + Disconnect

```
# Internal function for creating a GATTRequester for the BLEConnectionManager
def createRequester(self, address):
    if self.adapter == "":
        self.requester = GATTRequester(address, False)
    else:
        self.requester = GATTRequester(address, False, self.adapter)
    return self.requester

# Internal function for creating a GATTResponder for the BLEConnectionManager
def createResponder(self):
    try:
        self.response = GATTResponse()
    except:
        print("Uh oh.... GATTResponse() did not initialize?")
    return self.response

# Internal function for connecting to a target BLE device
#   - NOTE: Can have an issue where connection will be refused (e.g. non-advertising device)
def connect(self, conn_type):
    print("[*] Connecting...", end=" ")
    # Flush standard out
    sys.stdout.flush()
    # TODO: Add in error output to figure out why connections are/aren't allowed
    #   - NOTE: Figure out why I seem to need to RANDOMLY change between channel_types??
    #self.requester.connect(True, channel_type="random")     # NOTE: channel_type defaults to public
    #self.requester.connect(True, channel_type="public")     # NOTE: channel_type defaults to public
    print("\tusing channel_type [ {0} ]".format(conn_type))
    self.requester.connect(True, channel_type=conn_type)     # NOTE: channel_type defaults to public
    # ^ Using the above allows for communication back from BOTH the BLE CTF and the Office Orb
    # Certain devices may not have a public channel_type and we then use the line above
    print("[+] Connected")

# Internal function for checking the status of a connection
def check_status(self):
    status = "Connected" if self.requester.is_connected() else "Not Connected"
    print("[*] Checking current status:\t\t{0}".format(status))
    time.sleep(1)

# Internal function for disconnecting from a target BLE device
def disconnect(self):
    print("[*] Disconnecting...", end='\n')
    sys.stdout.flush()

    self.requester.disconnect()
    print("[-] Disconnected")

# Internal function for check the name of the given device      | NOTE: Not sure that this command works....
def get_name(self, address):
    print("[*] Reading device name...", end='\n')
    device_name = bluetooth.lookup_name(address)
    if device_name == '':
        print('\tName field was empty')
    else:
        print('\tName:\t{0}'.format(device_name))
    return device_name

# Internal function for Returning the UUID and ranges for the primary characteristics
def get_primaries(self):
    print("[*] Requesting a list of the Primary Characteristics")
    if not self.requester.is_connected():
        print("[-] ERROR: Not connected to a device...")
        return None
    else:
        print("[+] Connected to device! Retrieving Primary Characteristics")
```

pybluez_scanning.py

[Dis—C]onnection

- Connect + Disconnect

```
# Internal function for connecting to a target BLE device
#   - NOTE: Can have an issue where connection will be refused (e.g. non-advertising device)
def connect(self, conn_type):
    print("[*] Connecting...", end=" ")
    # Flush standard out
    sys.stdout.flush()
    # TODO: Add in error output to figure out why connections are/aren't allowed
    #   - NOTE: Figure out why I seem to need to RANDOMLY change between channel_types??
    #self.requester.connect(True, channel_type="random")     # NOTE: channel_type defaults to public
    #self.requester.connect(True, channel_type="public")     # NOTE: channel_type defaults to public
    print("\tusing channel_type [ {0} ]".format(conn_type))
    self.requester.connect(True, channel_type=conn_type)     # NOTE: channel_type defaults to public
    # ^ Using the above allows for communication back from BOTH the BLE CTF and the Office Orb
    # Certain devices may not have a public channel_type and we then use the line above
    print("[+] Connected")

# Internal function for checking the status of a connection
def check_status(self):
    status = "Connected" if self.requester.is_connected() else "Not Connected"
    print("[*] Checking current status:\t\t{0}".format(status))
    time.sleep(1)

# Internal function for disconnecting from a target BLE device
def disconnect(self):
    print("[*] Disconnecting...", end='\n')
    sys.stdout.flush()

    self.requester.disconnect()
    print("[-] Disconnected")
```

- Interaction with General Access Profile (GAP) Information

[Dis—C]onnection

```
def connect(self, conn_type):
    print("[*] Connecting...", end=" ")
    # Flush standard out
    sys.stdout.flush()
    # TODO: Add in error output to figure out why connection
    #   - NOTE: Figure out why I seem to need to RANDOMLY ch
    #self.requester.connect(True, channel_type="random")
    #self.requester.connect(True, channel_type="public")
    print("\tusing channel_type [ {0} ]".format(conn_type))
    self.requester.connect(True, channel_type=conn_type)
    # ^ Using the above allows for communication back from B
    # Certain devices may not have a public channel_type and
    print("[+] Connected")
```

[Dis—C]onnection

```
# Internal function for checking the status of a connection
def check_status(self):
    status = "Connected" if self.requester.is_connected() else "Not Connected"
    print("[*] Checking current status:\t\t{0}".format(status))
    time.sleep(1)

# Internal function for disconnecting from a target BLE device
def disconnect(self):
    print("[*] Disconnecting...", end='\n')
    sys.stdout.flush()

    self.requester.disconnect()
    print("[-] Disconnected")
```

Goal Tracking - Phz2

Research – Phase Two - Goals	
Goal Posts	**Completion Metric**
Find replacement Python library for low level capability	✓
Learn Python D-Bus functionality	70%
Re-Create PyBluez code using D-Bus library functions	45%
Scan / Discover	✓
Connect / Disconnect	✓
Enumerate	
Read / Write I/O	
Create Classes for D-Bus & BLE structures	50%
Abstract the low level commands into higher level functions	10%
Create User Interface for all code functionality	
Allow for detailed enumeration of user selected devices	

Classing It Up

- Build out Classes + Error Handling
  - Group functionality
  - Separate where necessary
  - Set boundaries for transparency of communication between Classes

Classing It Up - D-Bus

- Requirements?
  - D-Bus filter configurations (e.g. Classic, BLE, both)
  - Record of seen devices
- Purpose?
  - Identify devices based filters
  - Abstract D-Bus adapter
  - Simplify everything

Classing It Up - D-Bus

```
### Classes

## D-Bus Classes

 Adapter Interface Class   -   Used to Interact with the Adapter Interface, Set the Discovery Filter, and Search for Devices
lass system_dbus__bluez_adapter:

    '''
    Entry point for using and interacting with the system's adapter interface

    This class is intended to be used for the purpose of interacting with Bluetooth devices and for enabling the running of device scans
    '''

    # Initialization Function
    def __init__(self):
        # Note: Adding the '_' in front seems to work the same as the '.' in a linux directory
        self._bus = dbus.SystemBus()
        self.devices_found = None
        self.default_discovery_filter = {'Transport': 'auto'}       # Note: Uses whatever the system adapter has configured
        self.classic_discovery_filter = {'Transport': 'bredr'}      # ONLY search for Bluetooth Classic devices
        self.le_discovery_filter = {'Transport': 'le'}              # ONLY search for Bluetooth LE devices
        self.custom_discovery_filter = None                         # ONLY used by a user to set a custom discovery filter

    # Internal Function for Running a Device Scan
    def run_scan(self):
        # Check if the custom_discovery_filter has been set
        if not self.custom_discovery_filter:
            run_and_detect__bluetooth_devices__with_provided_filter(self.default_discovery_filter)
        else:
            run_and_detect__bluetooth_devices__with_provided_filter(self.custom_discovery_filter)
        self.devices_found = create_and_return__discovered_managed_objects()

    # Internal Functio for Setting the Discovery Filter
    def set_discovery_filter(self, new_discovery_filter):
        # Set the internal/customer discovery filter
        self.custom_discovery_filter = new_discovery_filter

# Device Object + Interface for Bluetooth Low Energy    -   Used to Interact with a Device, Determine the Device Properties, and Introspect the ``Lower Level'' Properties (i.e. Services, Characteristics, and Descriptors)
class system_dbus__bluez_device__low_energy:
    '''
    Entry point for creating, using, and interacting with a device via the system's D-Bus interfaces

    This class is intended to be used for the purpose of interacting with Bluetooth Low Energy (BLE) devices for connection + enumeration + interaction
    '''

    # Initialization Function
    def __init__(self, ble_device_address):
        # Note: Adding the '_' in front seems to work the same as the '.' in a linux directory
        self._bus = dbus.SystemBus()
        #self.device_address = ble_device_address
        # Create the full device path for setting to the class variable
        ble_device_path = bluetooth_utils.device_address_to_path(ble_device_address, bluetooth_constants.BLUEZ_NAMESPACE + bluetooth_constants.ADAPTER_NAME)
        # Path, Object, Interface, Properties, and Introspection for the Device
        self.device_path = ble_device_path
        self.device_object = None
        self.device_interface = None
        self.device_properties = None
        self.device_introspection = None
        # Properties related to the Device
        self.device_address = ble_device_address    #None                # Note: Later on this should match the ble_device_address passed
        self.device_address_type = None
        self.device_name = None
        self.device_alias = None
        self.device_connected = None
pybluez_dbus__now_with_classes.py                                                                                    260,1         42
[0] 0:nvim* 1:python3-                                                                                    "kali" 23:58 05-Apr-23
```

Figure 13: BLEEP - D-Bus Class Definition

Classing It Up - D-Bus

```
Adapter Interface Class   -   Used to Interact with the Adapter Interface, Set the Discovery Filter, and Search for Devices
lass system_dbus__bluez_adapter:

  '''
  Entry point for using and interacting with the system's adapter interface

  This class is intended to be used for the purpose of interacting with Bluetooth devices and for enabling the running of device scans
  '''

  # Initialization Function
  def __init__(self):
      # Note: Adding the '_' in front seems to work the same as the '.' in a linux directory
      self._bus = dbus.SystemBus()
      self.devices_found = None
      self.default_discovery_filter = {'Transport': 'auto'}       # Note: Uses whatever the system adapter has configured
      self.classic_discovery_filter = {'Transport': 'bredr'}      # ONLY search for Bluetooth Classic devices
      self.le_discovery_filter = {'Transport': 'le'}              # ONLY search for Blueooth LE devices
      self.custom_discovery_filter = None                         # ONLY used by a user to set a custom discovery filter

  # Internal Function for Running a Device Scan
  def run_scan(self):
      # Check if the custom_discovery_filter has been set
      if not self.custom_discovery_filter:
          run_and_detect__bluetooth_devices__with_provided_filter(self.default_discovery_filter)
      else:
          run_and_detect__bluetooth_devices__with_provided_filter(self.custom_discovery_filter)
      self.devices_found = create_and_return__discovered_managed_objects()

  # Internal Functio for Setting the Discovery Filter
  def set_discovery_filter(self, new_discovery_filter):
      # Set the internal/customer discovery filter
      self.custom_discovery_filter = new_discovery_filter
```

Figure 14: BLEEP - D-Bus Class Definition

Classing It Up - Bluetooth Low Energy

- Requirements?
  - Low level (i.e. D-Bus) element tracking
  - High level info (e.g. name, address)
  - Abstract operations with low-level
- Purpose?
  - Simplify user-run exploration
  - Create framework to automate enumeration
  - Platform for deep diving (e.g. capabilities, function, security)

Classing It Up - Bluetooth Low Energy

Figure 15: BLEEP - BLE Class Definition

Classing It Up - Bluetooth Low Energy

```
Device Object + Interface for Bluetooth Low Energy    -   Used to Interact with a Device, Determine the Device Properties, and Introspect the ``Lower Level'
lass system_dbus__bluez_device__low_energy:
    '''
    Entry point for creating, using, and interacting with a device via the system's D-Bus interfaces

    This class is intended to be used for the purpose of interacting with Bluetooth Low Energy (BLE) devices for connection + enumeration + interaction
    '''

    # Initialization Function
    def __init__(self, ble_device_address):
        # Note: Adding the '_' in front seems to work the same as the '.' in a linux directory
        self._bus = dbus.SystemBus()
        #self.device_address = ble_device_address
        # Create the full device path for setting to the class variable
        ble_device_path = bluetooth_utils.device_address_to_path(ble_device_address, bluetooth_constants.BLUEZ_NAMESPACE + bluetooth_constants.ADAPTER_NAME)
        # Path, Object, Interface, Properties, and Introspection for the Device
        self.device_path = ble_device_path
        self.device_object = None
        self.device_interface = None
        self.device_properties = None
        self.device_introspection = None
        # Properties related to the Device
        self.device_address = ble_device_address    #None                # Note: Later on this should match the ble_device_address passed
        self.device_address_type = None
        self.device_name = None
        self.device_alias = None
        self.device_connected = None
```

Figure 16: BLEEP - BLE Class Definition

Classing It Up - Error Handling

- But what about when things go wrong?
- Desire more than crashing and burning

```
# Internal Function for Handling Errors
def understand_and_handle__dbus_errors(self, exception_e):
    # Giant if statement determining what the error is and what the source of the error may be
    #if exception_e == DBusException:
        # Something
        #print("[!] Error: May need to Connect to the Device first?\n\t{0}".format(exception_e))
    if exception_e == AttributeError:
        # Attribute Error Condition
        print("[!] Error: May not have setup a D-Bus interface being used?\n\t{0}".format(exception_e))
    elif isinstance(exception_e, dbus.exceptions.DBusException):         ## Does not seem to work
        # Some form of DBusException
        if dbg != 0:
            print("[!] Error: D-Bus error has occured!\n\t{0}".format(exception_e))
            #raise RuntimeError(exception_e.get_dbus_message())
        #if "NotPermitted" in exception_e.args:
        if "Read not permitted" in exception_e.args:
            # Not Permitted Error Condition
            print("[!] Error: May not have permission for R/W; perhaps need more than just connection to device? OR the 'read' capability does not exist at the target\n\t{0}".format(exception_e))
        #elif "InvalidArguments" in exception_e.args:
        elif "Invalid offset" in exception_e.args:
            # Invalid Arguments Error Condition
            print("[!] Error: May have incorrectly passed R/W variables? (e.g. invalid offset)\n\t{0}".format(exception_e))
        else:
            # Unknown D-Bus Error
            print("[!] Error: D-Bus error\t-\t{0}".format(exception_e))
            print("\tType:\t{0}".format(exception_e))
            print("\tArgs:\t{0}".format(exception_e.args))
            print("\tD-Bus Message:\t{0}".format(exception_e.get_dbus_message()))
    else:
        print("[!] Error:\tUnknown Error\t-\t{0}".format(exception_e))
        if dbg != 0:
            print(exception_e.get_dbus_message())
            print(type(exception_e))
            print(exception_e.args)
            print(exception_e)
# Internal Function for Creating and Setting the Device Object relating to the BLE Device Path
def create_and_set__device_object(self):
    # Create the device object
    ble_device_object = self._bus.get_object(bluetooth_constants.BLUEZ_SERVICE_NAME, self.device_path)
    # Set the device object to the internal class variable
    self.device_object = ble_device_object

# Internal Function for Creating and Setting the Device Interface relating to the Device Object
def create_and_set__device_interface(self):
    # Check that the Device Object exists (is it None?)
    if not self.device_object:
        # Create the Device Object and set it before continuing
        self.create_and_set__device_object()
    # Continue with the creation and setting of the Device Interface
    self.device_interface = dbus.Interface(self.device_object, bluetooth_constants.DEVICE_INTERFACE)

# Internal Function for Creating and Setting the Device Properties relating to the Device Object
def create_and_set__device_properties(self):
    # Check that the Device Object exists (is it None?)
    if not self.device_object:
        # Create the Device Object and set it before continuing
        self.create_and_set__device_object()
    # Continue with the creation and setting of the Device Properties
    self.device_properties = dbus.Interface(self.device_object, bluetooth_constants.DBUS_PROPERTIES)

# Internal Function for Creating and Setting the Device Introspection relating to the Device Interface
##  - NOTE: This will provide a 'snapshot in time' of the Bluetooth Low Energy GATT Device Interface; Does NOT handle the need to double-read some properties
pybluez_dbus__now_with_classes.py                                                    331,1          62%
[0] 0:nvim* 1:python3-                                                   "kali" 23:59 05-Apr-23
```

Figure 17: BLEEP - Custom Error Handling

Classing It Up - Error Handling

```
# Internal Function for Handling Errors
def understand_and_handle_dbus_errors(self, exception_e):
    # Giant if statement determining what the error is and what the source of the error may be
    #if exception_e == DBusException:
        # Something
        #print("[!] Error: May need to Connect to the Device first?\n\t{0}".format(exception_e))
    if exception_e == AttributeError:
        # Attribute Error Condition
        print("[!] Error: May not have setup a D-Bus interface being used?\n\t{0}".format(exception_e))
    elif isinstance(exception_e, dbus.exceptions.DBusException):          ## Does not seem to work
        # Some form of DBusException
        if dbg != 0:
            print("[!] Error: D-Bus error has occured!\n\t{0}".format(exception_e))
            #raise RuntimeError(exception_e.get_dbus_message())
        #if "NotPermitted" in exception_e.args:
        if "Read not permitted" in exception_e.args:
            # Not Permitted Error Condition
            print("[!] Error: May not have permission for R/W; perhaps need more than just connection to device? OR the 'rea
        #elif "InvalidAgruments" in exception_e.args:
        elif "Invalid offset" in exception_e.args:
            # Invalid Arguments Error Condition
            print("[!] Error: May have incorrectly passed R/W variables? (e.g. invalid offset)\n\t{0}".format(exception_e))
        else:
            # Unknown D-Bus Error
            print("[!] Error: D-Bus error\t-\t{0}".format(exception_e))
            print("\tType:\t{0}".format(exception_e))
            print("\tArgs:\t{0}".format(exception_e.args))
            print("\tD-Bus Message:\t{0}".format(exception_e.get_dbus_message()))
    else:
        print("[!] Error:\tUnknown Error\t-\t{0}".format(exception_e))
        if dbg != 0:
            print(exception_e.get_dbus_message())
            print(type(exception_e))
            print(exception_e.args)
            print(exception_e)
```

Figure 18: BLEEP - Custom Error Handling

Classing It Up - Error Handling

```
else:
    # Unknown D-Bus Error
    print("[!] Error: D-Bus error\t-\t{0}".format(exception_e))
    print("\tType:\t{0}".format(exception_e))
    print("\tArgs:\t{0}".format(exception_e.args))
    print("\tD-Bus Message:\t{0}".format(exception_e.get_dbus_message()))
e:
```

Figure 19: BLEEP - Custom Error Handling

Classing It Up - Now What?

Improving the PoC

- Automate scanning process
  - Move away from Ctrl+C towards variable timer
- Creating an Arduino server
  - Control both sides
- Improving Read/Write
  - Purposeful, directed, intentional

G-G-G-GLib!

- GLib's MainLoop is central to D-Bus-2-BlueZ operation
  - A timeout can be provided in seconds or milliseconds
  - The **function** variable is the (callback) function to call
  - The **data** variable is the data to pass to **function**
- Created timeout for scans!
  - Remove requirement for humans

Arduino Server

- Require server with visibility to I/O
  - Simplified functionality
  - Control over landscape presented
  - Improve understanding of Server configurations vs Client “sight”

D-Bus Interfaces - Basics

- D-Bus interaction and enumeration is done via a series of interfaces
- Interfaces exist for each “layer” of interactivity with the GATT Server

D-Bus Interfaces - Little Bit Deeper Now

- Several types of D-Bus interfaces; each with separate purposes
  - Methods = Functions
  - Signals =?? Periodic Communications ?? __(o.O)__/
  - Properties = Properties
- Examples of interfaces:
  - Device - org.bluez.Device1
  - GATT Service - org.bluez.GattService1
  - Properties - org.freedesktop.DBus.Properties
  - Introspection - org.freedesktop.DBus.Introspectable

D-Bus Interfaces - busctl

```
┌──(user  kali)-[~/Documents/BTInteract]
└─$ busctl tree org.bluez
└─/org
  └─/org/bluez
    └─/org/bluez/hci0
      ├─/org/bluez/hci0/dev_CC_50_E3_B6_BC_A6
      │ ├─/org/bluez/hci0/dev_CC_50_E3_B6_BC_A6/service0001
      │ │ └─/org/bluez/hci0/dev_CC_50_E3_B6_BC_A6/service0001/char0002
      │ │   └─/org/bluez/hci0/dev_CC_50_E3_B6_BC_A6/service0001/char0002/desc0004
      │ └─/org/bluez/hci0/dev_CC_50_E3_B6_BC_A6/service0028
      │   ├─/org/bluez/hci0/dev_CC_50_E3_B6_BC_A6/service0028/char0029
      │   ├─/org/bluez/hci0/dev_CC_50_E3_B6_BC_A6/service0028/char002b
      │   ├─/org/bluez/hci0/dev_CC_50_E3_B6_BC_A6/service0028/char002d
      │   ├─/org/bluez/hci0/dev_CC_50_E3_B6_BC_A6/service0028/char002f
      │   ├─/org/bluez/hci0/dev_CC_50_E3_B6_BC_A6/service0028/char0031
      │   ├─/org/bluez/hci0/dev_CC_50_E3_B6_BC_A6/service0028/char0033
      │   ├─/org/bluez/hci0/dev_CC_50_E3_B6_BC_A6/service0028/char0035
      │   ├─/org/bluez/hci0/dev_CC_50_E3_B6_BC_A6/service0028/char0037
      │   ├─/org/bluez/hci0/dev_CC_50_E3_B6_BC_A6/service0028/char0039
      │   ├─/org/bluez/hci0/dev_CC_50_E3_B6_BC_A6/service0028/char003b
      │   ├─/org/bluez/hci0/dev_CC_50_E3_B6_BC_A6/service0028/char003d
      │   ├─/org/bluez/hci0/dev_CC_50_E3_B6_BC_A6/service0028/char003f
      │   ├─/org/bluez/hci0/dev_CC_50_E3_B6_BC_A6/service0028/char0041
      │   ├─/org/bluez/hci0/dev_CC_50_E3_B6_BC_A6/service0028/char0043
      │   ├─/org/bluez/hci0/dev_CC_50_E3_B6_BC_A6/service0028/char0045
```

D-Bus Interfaces - busctl - Introspection

```
┌──(user  kali)-[~/Documents/BTInteract]
└─$ busctl introspect org.bluez /org/bluez/hci0/dev_CC_50_E3_B6_BC_A6
NAME                                TYPE      SIGNATURE RESULT/VALUE                                FLAGS
org.bluez.Device1                   interface -         -                                           -
.CancelPairing                      method    -         -                                           -
.Connect                            method    -         -                                           -
.ConnectProfile                     method    s         -                                           -
.Disconnect                         method    -         -                                           -
.DisconnectProfile                  method    s         -                                           -
.Pair                               method    -         -                                           -
.Adapter                            property  o         "/org/bluez/hci0"                           emits-change
.Address                            property  s         "CC:50:E3:B6:BC:A6"                         emits-change
.AddressType                        property  s         "public"                                    emits-change
.Alias                              property  s         "2b00042f7481c7b056c4b410d28f33cf"          emits-change writable
.Appearance                         property  q         -                                           emits-change
.Blocked                            property  b         false                                       emits-change writable
.Class                              property  u         -                                           emits-change
.Connected                          property  b         true                                        emits-change
.Icon                               property  s         -                                           emits-change
.LegacyPairing                      property  b         false                                       emits-change
.ManufacturerData                   property  a{qv}     -                                           emits-change
.Modalias                           property  s         -                                           emits-change
.Name                               property  s         "2b00042f7481c7b056c4b410d28f33cf"          emits-change
.Paired                             property  b         false                                       emits-change
.RSSI                               property  n         -                                           emits-change
.ServiceData                        property  a{sv}     -                                           emits-change
.ServicesResolved                   property  b         true                                        emits-change
.Trusted                            property  b         false                                       emits-change writable
.TxPower                            property  n         -                                           emits-change
.UUIDs                              property  as        3 "000000ff-0000-1000-8000-00805f9b34fb"  emits-change
.WakeAllowed                        property  b         -                                           emits-change writable
org.freedesktop.DBus.Introspectable interface -         -                                           -
.Introspect                         method    -         s                                           -
org.freedesktop.DBus.Properties     interface -         -                                           -
.Get                                method    ss        v                                           -
.GetAll                             method    s         a{sv}                                       -
.Set                                method    ssv       -                                           -
```

D-Bus Interfaces - Introspection

- Compare returned data against 'busctl' visibility
- Create rough structures
  - Enough to proceed with enumeration
- Understanding: Each interface is explored via an **introspection** interface
  - Using the introspection interface's introspect method **(-_-;)**

D-Bus Interfaces - E-Tree Parsing

```python
    # Internal Function for Creating an eTree from Introspection and Returning its Contents
    #   - Note: The actions taken depend on the search_term provided and depends on what is being searched for (i.e, Service, Characteristics, or Descriptor)
    def find_and_get__device__etree_details(self, introspection_interface, search_term):
        # Ensure that the introspection_interface is not None
        if not introspection_interface:
            if dbg != 0:
                print("[!] Error: Introspection Interface is non-existant")
            return None
        # Create the eTree string to interacte through
        introspection_tree = ET.fromstring(introspection_interface)
        introspection_contents = []
        ## TODO: Add capability to extract out the 'interface' tags to know what capabilities a given introspection_tree returns
        # Loop through the eTree and collect all the child.attrib that are 'node's and match the provided search term
        for child in introspection_tree:
            # Confirm that examining a 'node'
            if child.tag == 'node':
                # Looking for GATT Services
                if search_term == 'service':
                    introspection_contents.append(child.attrib['name'])
                elif search_term == 'char':
                    introspection_contents.append(child.attrib['name'])
                elif search_term == 'desc':
                    introspection_contents.append(child.attrib['name'])
                else:
                    if dbg != 0:
                        print("[!] Error: Search term for introspection was unknown")
        # Return the Introspection Contents
        return introspection_contents

    # Internal Function for Enumerating the Services presented by the Device Introspection interface
    def find_and_get__device_introspection__services(self):
        # Ensure that the Necessary Introspection Interface
        if not self.device_introspection:
            # Create the Introspection Interface and set it before continuing
            self.create_and_set__device_introspection()
        # Call the class function and passing that Services are being searched for
        introspection_services_list = self.find_and_get__device__etree_details(self.device_introspection, 'service')
        # Return the found list of device services
        return introspection_services_list

    # Internal Function for Enumerating the Characteristics presented by the Service Introspection interface
    #   - Note: There is a variable number of characteristics PER service; may also be ZERO
    def find_and_get__device_introspection__characteristics(self, service_path, service_characteristics_list):
        introspection_characteristics_list = []
        # Call internal class function
        if not service_characteristics_list:
            if dbg != 0:
                print("[*] Service's characteristics list is empty:\t\t{0}".format(service_characteristics_list))
            return None
        for service_characteristic in service_characteristics_list:
            # Create the eTree string to interate through the characteristics of the service
            characteristic_path, characteristic_object, characteristic_interface, characteristic_properties, characteristic_introspection = self.create_and_return__characteristic__gatt_inspection_set(service_path, service_characteristic)
            # TODO: Figure out how to have this function return (ALL?) characteristics related to the device (? or just a single level's worth)
            #introspection_characteristics_list.append()
            introspection_characteristics_list.append(self.find_and_get__device__etree_details(characteristic_introspection, 'char'))
        # Return the found list
        return introspection_characteristics_list

    # Internal Function for Enumerating the Descriptors presented by the Characteristic Introspection interface
    #   - Note: There is a variable number of descriptors (???) PER characteristic; should ONLY be ONE (??)
    def find_and_get__device_introspection__descriptors(self, characteristic_descriptors_list):
        introspection_descriptors_list = []
        # Call internal class function
        if not characteristic_descriptors_list:
```


Figure 22: BLEEP - Introspection E-Tree Parsing

D-Bus Interfaces - E-Tree Parsing

```
# Internal Function for Creating an eTree from Introspection and Returning its Contents
#   - Note: The actions taken depend on the search_term provided and depends on what is
def find_and_get__device__etree_details(self, introspection_interface, search_term):
    # Ensure that the introspection_interface is not None
    if not introspection_interface:
        if dbg != 0:
            print("[!] Error: Introspection Interface is non-existant")
        return None
    # Create the eTree string to interacte through
    introspection_tree = ET.fromstring(introspection_interface)
    introspection_contents = []
    ## TODO: Add capability to extract out the 'interface' tags to know what capabiliti
    # Loop through the eTree and collect all the child.attrib that are 'node's and matc
    for child in introspection_tree:
        # Confirm that examining a 'node'
        if child.tag == 'node':
            # Looking for GATT Services
            if search_term == 'service':
                introspection_contents.append(child.attrib['name'])
            elif search_term == 'char':
                introspection_contents.append(child.attrib['name'])
            elif search_term == 'desc':
                introspection_contents.append(child.attrib['name'])
            else:
                if dbg != 0:
                    print("[!] Error: Search term for introspection was unknown")
    # Return the Introspection Contents
```

D-Bus Interfaces - E-Tree Parsing

```
for child in introspection_tree:
    # Confirm that examining a 'node'
    if child.tag == 'node':
        # Looking for GATT Services
        if search_term == 'service':
            introspection_contents.append(child.attrib['name'])
        elif search_term == 'char':
            introspection_contents.append(child.attrib['name'])
        elif search_term == 'desc':
            introspection_contents.append(child.attrib['name'])
        else:
            if dbg != 0:
                print("[!] Error: Search term for introspection
```

Figure 24: BLEEP - Introspection E-Tree Parsing

Abstracting Introspection

- Produces an XML map of the introspected interface's information
  - Contains (1) [Sub-]Interfaces and (2) [Sub-]Nodes associated to the introspected interface

```
[!] Introspection E-Tree Information:          <Element 'node' at 0x74b486f0c2c0>
        Child Tag:      [ interface ]          -              Child Attrib:   [ {'name': 'org.freedesktop.DBus.Introspectable'} ]
        Child Tag:      [ interface ]          -              Child Attrib:   [ {'name': 'org.bluez.GattService1'} ]
        Child Tag:      [ interface ]          -              Child Attrib:   [ {'name': 'org.freedesktop.DBus.Properties'} ]
        Child Tag:      [ node ]               -              Child Attrib:   [ {'name': 'char002f'} ]
        Child Tag:      [ node ]               -              Child Attrib:   [ {'name': 'char0031'} ]
        Child Tag:      [ node ]               -              Child Attrib:   [ {'name': 'char0034'} ]
        Child Tag:      [ node ]               -              Child Attrib:   [ {'name': 'char0037'} ]
        Child Tag:      [ node ]               -              Child Attrib:   [ {'name': 'char003a'} ]
        Child Tag:      [ node ]               -              Child Attrib:   [ {'name': 'char003d'} ]
        Child Tag:      [ node ]               -              Child Attrib:   [ {'name': 'char003f'} ]
        Child Tag:      [ node ]               -              Child Attrib:   [ {'name': 'char0042'} ]
[!] Introspection E-Tree Information:          <Element 'node' at 0x74b486f0cc70>
        Child Tag:      [ interface ]          -              Child Attrib:   [ {'name': 'org.freedesktop.DBus.Introspectable'} ]
        Child Tag:      [ interface ]          -              Child Attrib:   [ {'name': 'org.bluez.GattCharacteristic1'} ]
        Child Tag:      [ interface ]          -              Child Attrib:   [ {'name': 'org.freedesktop.DBus.Properties'} ]
```

Figure 25: Etree XML Example - Light Orb

Improvements - UI + Server(s)

```
Select an Action to Take: info
        Address:        94:B5:55:C0:C2:1E
Device  94:B5:55:C0:C2:1E
        Properties:             {dbus.String('Address'): '94:B5:55:C0:C2:1E', dbus.String('AddressType'): 'public', dbus.String('Nam
e'): 'DeviceName:BLE Serve', dbus.String('Alias'): 'DeviceName:BLE Serve', dbus.String('Paired'): False, dbus.String('Trusted'): Fal
se, dbus.String('Blocked'): False, dbus.String('LegacyPairing'): False, dbus.String('Connected'): True, dbus.String('UUIDs'): ['0000
1800-0000-1000-8000-00805f9b34fb', '00001801-0000-1000-8000-00805f9b34fb', '0000c103-0000-1000-8000-00805f9b34fb', '0000c104-0000-10
00-8000-00805f9b34fb'], dbus.String('Adapter'): '/org/bluez/hci0', dbus.String('ServicesResolved'): True}
        Address         -       Value:  94:B5:55:C0:C2:1E
        AddressType             -       Value:  public
        Name            -       Value:  DeviceName:BLE Serve
        Alias           -       Value:  DeviceName:BLE Serve
        Paired          -       Value:  False
        Trusted         -       Value:  False
        Blocked         -       Value:  False
        LegacyPairing           -       Value:  False
        Connected               -       Value:  True
        UUIDs           -       Value:  ['00001800-0000-1000-8000-00805f9b34fb', '00001801-0000-1000-8000-00805f9b34fb', '0000c103-0
000-1000-8000-00805f9b34fb', '0000c104-0000-1000-8000-00805f9b34fb']
        Adapter         -       Value:  /org/bluez/hci0
        ServicesResolved                -       Value:  True
Select an Action to Take:
```

Figure 26: Arduino Server - GAP Enumeration

Improvements - UI + Server(s)

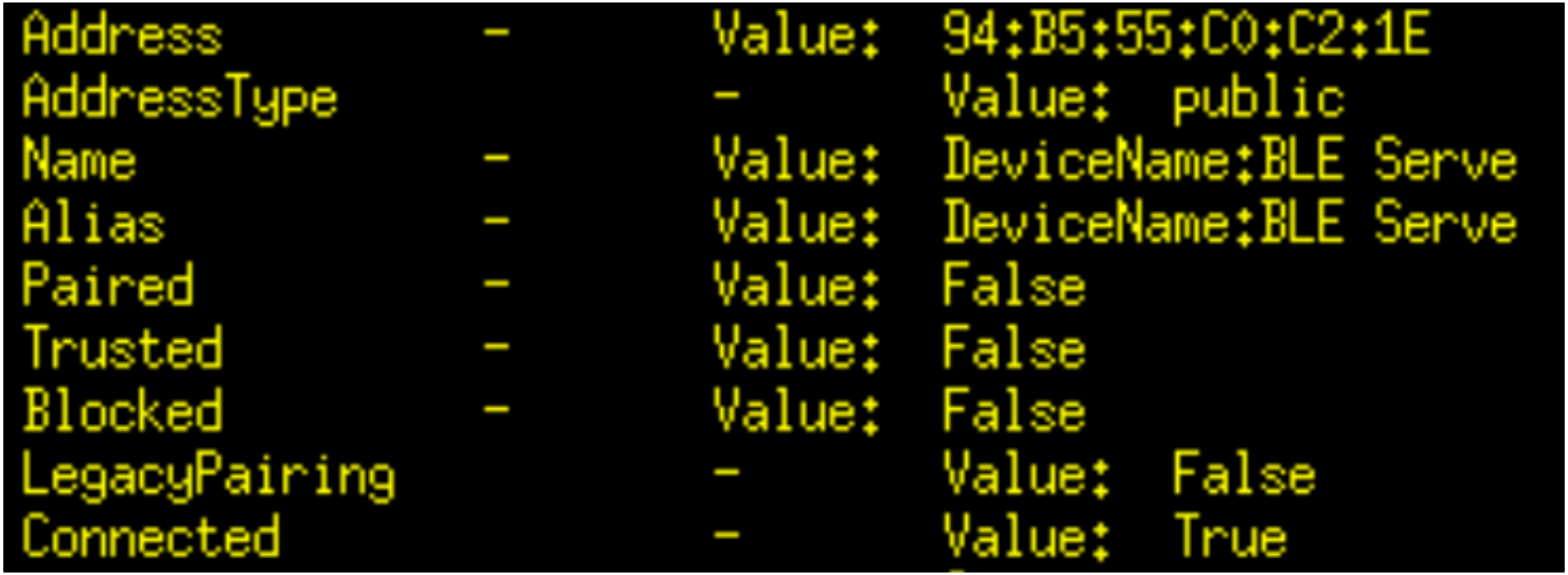


Figure 27: Arduino Server - GAP Enumeration - Zoom on GAP

Improvements - UI + Server(s)

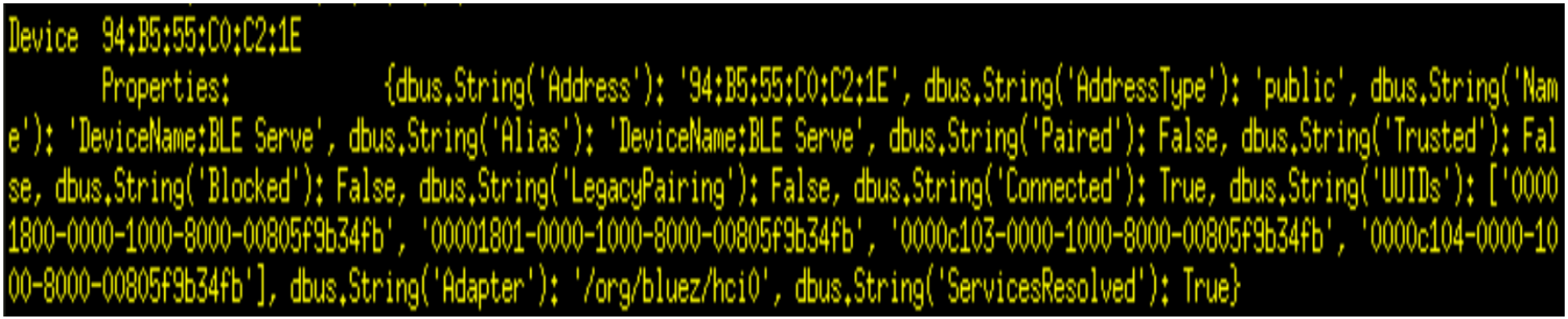


Figure 28: Arduino Server - GAP Enumeration - Zoom on Properties

Improvements - UI + Server(s)

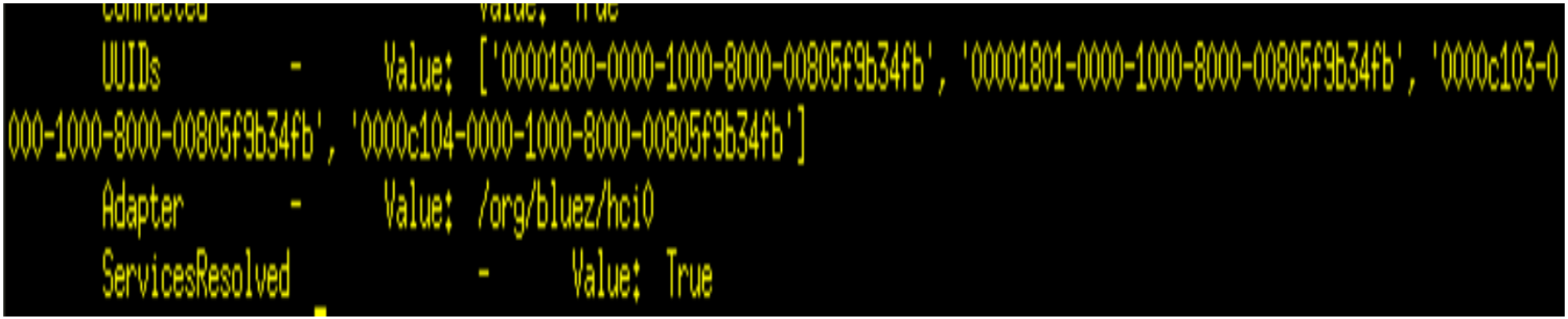


Figure 29: Arduino Server - GAP Enumeration - Zoom on UUIDs

Goal Tracking - Phz2

Research – Phase Two - Goals	
Goal Posts	**Completion Metric**
Find replacement Python library for low level capability	✓
Learn Python D-Bus functionality	85%
Re-Create PyBluez code using D-Bus library functions	65%
Scan / Discover	✓
Connect / Disconnect	✓
Enumerate	✓
Read / Write I/O	
Create Classes for D-Bus & BLE structures	75%
Abstract the low level commands into higher level functions	80%
Create User Interface for all code functionality	
Allow for detailed enumeration of user selected devices	

Signature-ing a Time Sink

- BlueZ git.kernel repository documentation [16]
  - API documentation for various components (e.g. Device, Service, Characteristic, Descriptor)
  - *Note:* Documentation is explanation of C/C++ code being abstracted via Python
- Read / Write interaction via D-Bus structures more complex than original PyBlueZ research variant
  - Defined interface (e.g. characteristic_interface) to access D-Bus methods
  - R+W **must** have "dict_options" passed; even if empty ({})

Signature-ing a Time Sink - Characteristic

```
┌──(user  kali)-[~/Documents/BTInteract]
└─$ busctl introspect org.bluez /org/bluez/hci0/dev_CC_50_E3_B6_BC_A6/service0001/char0002
NAME                                TYPE      SIGNATURE RESULT/VALUE                              FLAGS
org.bluez.GattCharacteristic1       interface -         -                                         -
.AcquireNotify                      method    a{sv}     hq                                        -
.AcquireWrite                       method    a{sv}     hq                                        -
.ReadValue                          method    a{sv}     ay                                        -
.StartNotify                        method    -         -                                         -
.StopNotify                         method    -         -                                         -
.WriteValue                         method    aya{sv}   -                                         -
.Flags                              property  as        1 "indicate"                              emits-change
.MTU                                property  q         500                                       emits-change
.NotifyAcquired                     property  b         -                                         emits-change
.Notifying                          property  b         false                                     emits-change
.Service                            property  o         "/org/bluez/hci0/dev_CC_50_E3_B6_BC_A6/   emits-change
.UUID                               property  s         "00002a05-0000-1000-8000-00805f9b34fb"    emits-change
.Value                              property  ay        0                                         emits-change
.WriteAcquired                      property  b         -                                         emits-change
org.freedesktop.DBus.Introspectable interface -         -                                         -
.Introspect                         method    -         s                                         -
org.freedesktop.DBus.Properties     interface -         -                                         -
.Get                                method    ss        v                                         -
.GetAll                             method    s         a{sv}                                     -
.Set                                method    ssv       -                                         -
.PropertiesChanged                  signal    sa{sv}as  -                                         -
```

Figure 31: busctl - Inspecting BlueZ Characteristic

Signature-ing a Time Sink

- Reading: straight forward and easy
  - Ex: `characteristic_interface.ReadValue({})`
  - Returns: `dbus.Array([dbus.Byte(0)], signature=dbus.signature('y'))`

Signature-ing a Time Sink

- Writing: more complex with lots of trail/error
  - Ex: characteristic_interface.WriteValue("1", {})
  - Returns: `ERROR: dbus.connection: Unable to set arguments ('1', {}) according to signature 'aya{sv}': < class 'TypeError' >: an integer is required (got type str)`

Signature-ing a Time Sink

- Requirement was to pass the value as an array
  - Ex: `.WriteValue([1], {})`
  - *Note:* Information that can be decoded from the signature

Improvements - UI + Server(s)

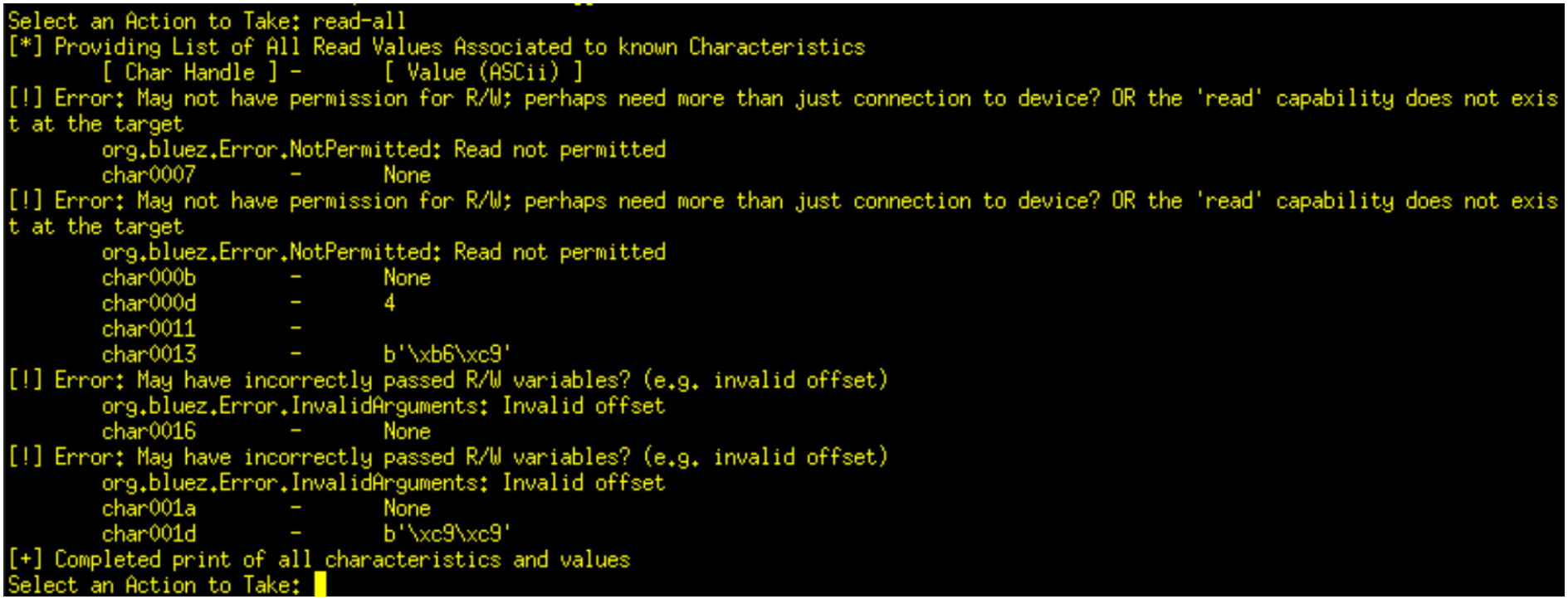


Figure 32: Arduino Server - Characteristics Enumeration

Goal Tracking - Phz2

Research – Phase Two - Goals	
Goal Posts	**Completion Metric**
Find replacement Python library for low level capability	✓
Learn Python D-Bus functionality	✓
Re-Create PyBluez code using D-Bus library functions	✓
Scan / Discover	✓
Connect / Disconnect	✓
Enumerate	✓
Read / Write I/O	✓
Create Classes for D-Bus & BLE structures	✓
Abstract the low level commands into higher level functions	✓
Create User Interface for all code functionality	
Allow for detailed enumeration of user selected devices	

Improvements - UI + Server(s)

```
        Characteristic Value (ASCii):   Set your connection MTU to 444
        Characteristic UUID:    0000ff14-0000-1000-8000-00805f9b34fb          -               Unknown
        Characteristic Flags:   ['read', 'write']
        Characteristic Value (ASCii):   Write+resp 'hello'
        Write Value:    1
        Write Type:     <class 'int'>
[*] Write Value is Integer
[*] Performing Single Byte Write
                [-] Write       -       Failed  -       UUID:   0000ff14-0000-1000-8000-00805f9b34fb
        Characteristic UUID:    0000ff15-0000-1000-8000-00805f9b34fb          -               Unknown
        Characteristic Flags:   ['read', 'write']
        Characteristic Value (ASCii):   No notifications here! really?
        Write Value:    1
        Write Type:     <class 'int'>
[*] Write Value is Integer
[*] Performing Single Byte Write
                [-] Write       -       Failed  -       UUID:   0000ff15-0000-1000-8000-00805f9b34fb
        Characteristic UUID:    0000ff16-0000-1000-8000-00805f9b34fb          -               Unknown
        Characteristic Flags:   ['broadcast', 'read', 'write', 'notify', 'extended-properties']
        Characteristic Value (ASCii):   fbb966958f
        Write Value:    1
        Write Type:     <class 'int'>
[*] Write Value is Integer
[*] Performing Single Byte Write
                [-] Write       -       Failed  -       UUID:   0000ff16-0000-1000-8000-00805f9b34fb
        Characteristic UUID:    0000ff17-0000-1000-8000-00805f9b34fb          -               Unknown
        Characteristic Flags:   ['read']
        Characteristic Value (ASCii):   md5 of author's twitter handle
[*] User Interactive Exploration Tool - Select Action
        - 'print' to Pretty Print the known user device internals map
        - 'info' to print device information
        - 'generate' to Access the Generation Sub-Menu
        - 'explore' to Access the Exploration Sub-Menu
        - 'read' to Access the Reading Sub-Menu
        - 'write' to Access the Writing Sub-Menu
        - 'help' to print this information
        - 'quit' to exit user exploration
        Nota Bene:      Complete Re-Read of the Device may be required to update the Device Internals Map
Select an Action to Take:
```

Improvements - UI + Server(s)

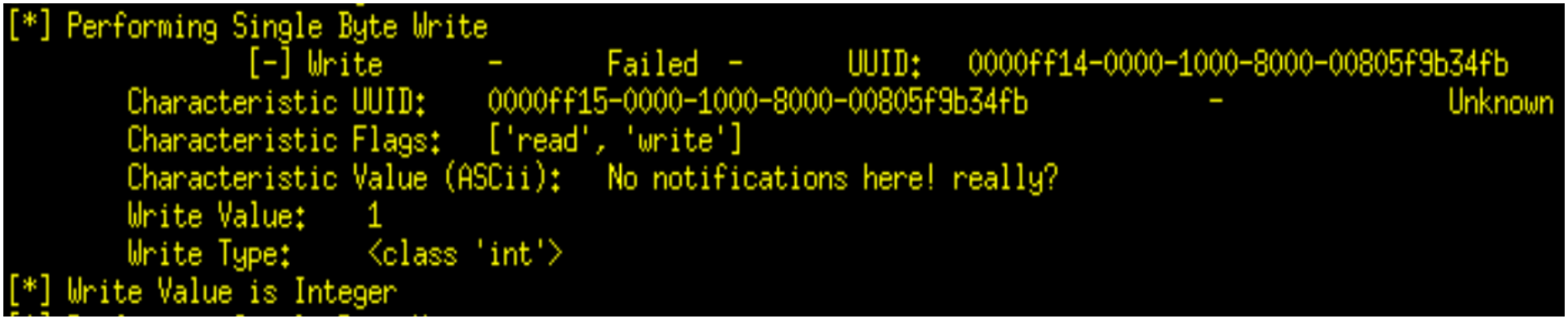
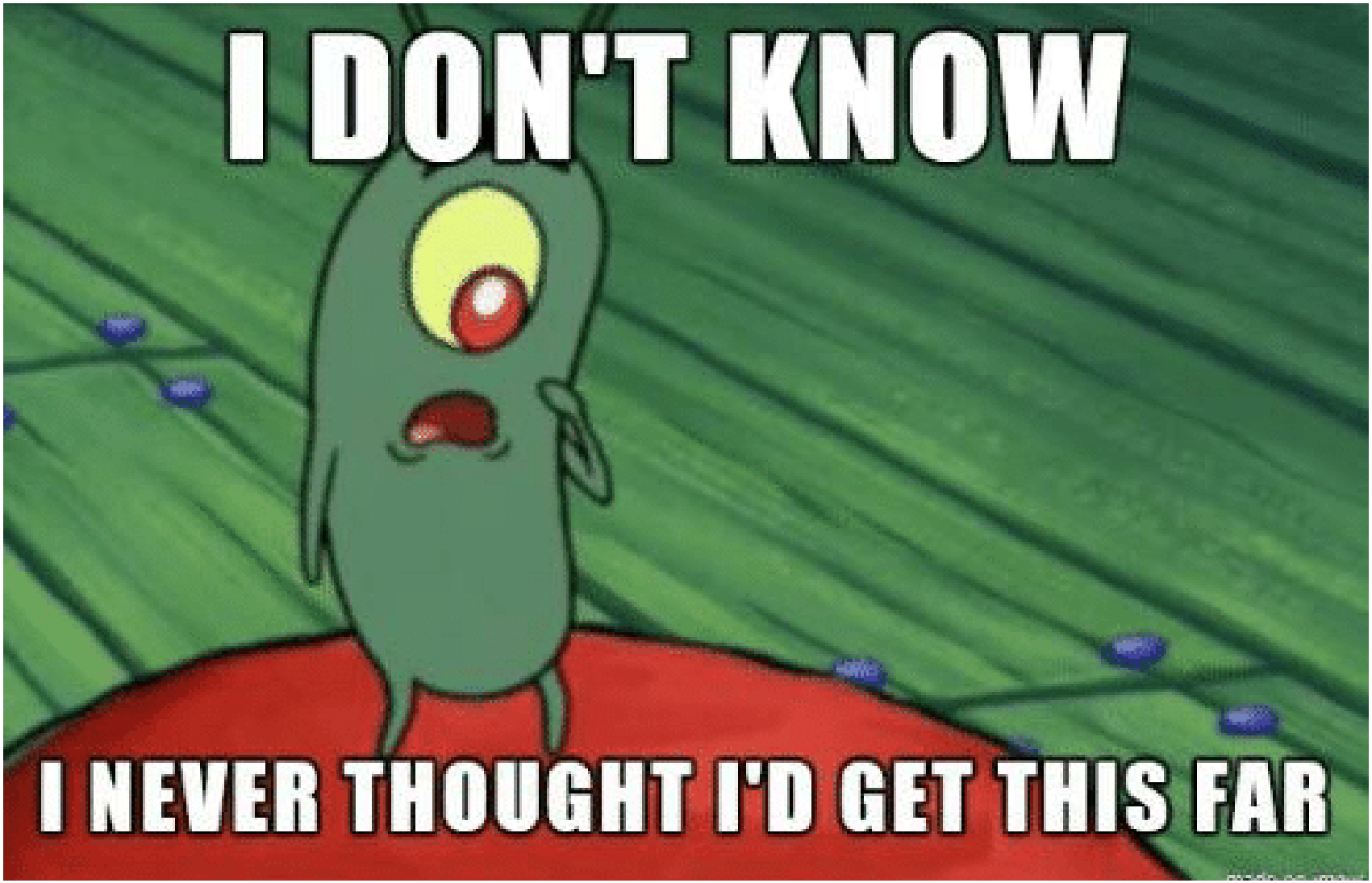


Figure 35: BLE CTF - Characteristics Enumeration

Goal Tracking - Phz2

Research – Phase Two - Goals	
Goal Posts	**Completion Metric**
Find replacement Python library for low level capability	✓
Learn Python D-Bus functionality	✓
Re-Create PyBluez code using D-Bus library functions	✓
Scan / Discover	✓
Connect / Disconnect	✓
Enumerate	✓
Read / Write I/O	✓
Create Classes for D-Bus & BLE structures	✓
Abstract the low level commands into higher level functions	✓
Create User Interface for all code functionality	✓
Allow for detailed enumeration of user selected devices	✓

Phase III - What Next?



Emissions - Signals + Notifications

- What are signals/emissions?
  - Messages which might be received asynchronously [11]
  - Application must register interest in the signal
  - Provide callback function to process the signal and any arguments

Goal Tracking - Phz3

Research – Phase Three - Goals	
Goal Posts	**Completion Metric**
Augment framework for multi-reads	✓
Augment framework for brute-force writes	✓
Augment framework for specific write-types	✓
Learning D-Bus signature decoding / translation	20%
Expand framework to allow capture of signals / notifications	
Automate discovery + enumeration	
Basic logging capability for offline analysis	

Figure 37: Goal Table - Phase Three - Starting Signals

Emissions - Signals + Notifications - Initial Assumption

- How complex can signals be?
  - Send out a signal
  - Receive a signal
  - Act on either aforementioned action

Emissions - Signals + Notifications - Documentation Research

- Minimum steps for setting up notify listener:
  - Find a characteristic that has a Notify flag *(e.g. BLE CTF char003f; handle 0x0040)*
  - Create the associated characteristic interface
    - Ex: `char_path, char_obj, char_int, char_prop, char_intro = user_device.create_and_return__characteristic__gatt_introspection__set(characteristic_service_path, characteristic_name)`
  - Acquire Notify using the characteristic interface
    - Ex: `characteristic_interface.AcquireNotify({})`
  - **Note**: Causes the characteristic property flag to change; BUT will require an updated read to be visible (caching)

Emissions - Signals + Notifications - Class Definition - Initialization

```
# Signal Catching Class - For catching/intercepting notification, indicate, or other Signal-Type (i.e. signal, method_call, method_response, error) and purposed with returning the information for other uses (e.g. tracking, updates, CTFs)
#   - Note: Similar to Adapter Class's functionality but much broader than just scans
class system_dbus__bluez_signals:
    '''
    Entry point for using and interacting with emitted signals over the D-Bus

    This class is intended to be used for the purpose of interacting with Bluetooth devices and for catching D-Bus/Bluetooth/GATT signals
    '''
    # Initialization Function
    def __init__(self):
        # Note: Adding the '_' in front seems to work the same as the '.' in a linux directory
        self._bus = dbus.SystemBus()
        self.devices_found = None
        #self.default_discovery_filter = {'Transport': 'auto'}      # Note: Uses whatever the system adapter has configured
        #self.classic_discovery_filter = {'Transport': 'bredr'}     # ONLY search for Bluetooth Classic devices
        #self.le_discovery_filter = {'Transport': 'le'}             # ONLY search for Bluetooth LE devices
        #self.custom_discovery_filter = None                        # ONLY used by a user to set a custom discovery filter
        self.timer_id__last = None                                 # Tracking the last Timer ID created (i.e. .timeout_add() for GLib MainLoop)
        self.timer_id__list = []                                   # Tracking all the Timer IDs created by this D-Bus BlueZ Adapter Class
        self.timer__default_time__ms = 5000                        # Default setting of scanning time to 5000 milliseconds
        self.mainloop_run__default_time__ms = 5000                 # Default setting of the amount of time the MainLoop will run at any given time
        self.signal_receiver__list = {}                            # Tracking Signal Receivers that have been added to the D-Bus

    ## Definitions for Internal Property Tracking
    # Internal Function for Clearing/Canceling All Active Timers
    def clear_all_timers(self):
        # Interate through all the timers in the internal array
        for timer_id__item in self.timer_id__list:
            #GLib.source_remove(self.timer_id__array[timer_id__item])
            # Remove the timer source from GLib
            GLib.source_remove(timer_id__item)
            # Remove the Timer ID from the list
            self.timer_id__list.remove(timer_id__item)
        # Clear/Re-Set the Timer Id Array
        #self.timer_id__array = {}

    # Internal Function for Clearing/Canceling a Single Timer - Identified via Timer ID
    def clear_single_timer(self, timer_id):
        # Ensure that the timer_id presented is known to the Adapter Class
        if timer_id not in self.timer_id__list:
            print("[!] ERROR:\tTimer ID is Unknown to Adapter")
            return False
        # Remove the Timer Source from GLib
        GLib.source_remove(timer_id)
        # Remove the Timer ID from the Timer ID Array
        self.timer_id__list.remove(timer_id)
        # Check if the Timer removed is the Last one seen
        if timer_id == self.timer_id__last:
            self.timer_id__last = None
        # Confirm everything got done
        return True
```

Figure 38: Signals Class - Initialization

Emissions - Signals + Notifications - Class Definition - Initialization

```
# Initialization Function
def __init__(self):
    # Note: Adding the '_' in front seems to work the same as
    self._bus = dbus.SystemBus()
    self.devices_found = None
    #self.default_discovery_filter = {'Transport': 'auto'}
    #self.classic_discovery_filter = {'Transport': 'bredr'}
    #self.le_discovery_filter = {'Transport': 'le'}
    #self.custom_discovery_filter = None
    self.timer_id__last = None
    self.timer_id__list = []
    self.timer__default_time__ms = 5000
    self.mainloop_run__default_time__ms = 5000
    self.signal_receiver__list = {}
```

Figure 39: Signals Class - Initialization Zoomed on Init

Emissions - Signals + Notifications - Class Definition - Initialization

```
## Definitions for Internal Property Tracking
# Internal Function for Clearing/Canceling All Active Timers
def clear_all_timers(self):
    # Interate through all the timers in the internal array
    for timer_id__item in self.timer_id__list:
        #GLib.source_remove(self.timer_id__array[timer_id__item])
        # Remove the timer source from GLib
        GLib.source_remove(timer_id__item)
        # Remove the Timer ID from the list
        self.timer_id__list.remove(timer_id__item)
    # Clear/Re-Set the Timer Id Array
    #self.timer_id__array = {}
```

Figure 40: Signals Class - Initialization Zoomed on All Timer Clear

Goal Tracking - Phz3

Research – Phase Three - Goals	
Goal Posts	**Completion Metric**
Augment framework for multi-reads	✓
Augment framework for brute-force writes	✓
Augment framework for specific write-types	✓
~~Learning D-Bus signature decoding / translation~~	~~20%~~
Expand framework to allow capture of signals / notifications	40%
Learn signal emittion & capture	30%
Establish callback functionality	
Automate discovery + enumeration	
Basic logging capability for offline analysis	

Figure 41: Goal Table - Phase Three - Learning Signal - Take Two

Emissions - Signals + Notifications - Back to the Research

- "... need details of the signal to be received so [one] can specify these details when registering with the system bus" [11]
  - Ex: `bus.add_signal_reciever(greeting_signal_received, dbus_interface = ‘‘com.example.greeting’’, signal_name = ‘‘Greeting signal’’)`
- Learned via Pain:
  - *Every* `add_signal_reciever()` generated gets called **whenever** a corresponding signal event occurs
  - Object and Interface aspects are **not** interchangeable; Characteristic Object has no StartNotify

Emissions - Signals + Notifications - Class Definition

```
    # Function for Catching an Interface Added Signal
    def callback__signal__interface_added(self, path, interfaces):
        # Check that the interface is the expected Bluetooth Device Interface
        if not bluetooth_constants.DEVICE_INTERFACE in interfaces:
            return
        # Extract the Device Properties
        device_properties = interfaces[bluetooth_constants.DEVICE_INTERFACE]
        ## Examining the Device Properties for specifics
        # Create logging string for general logging
        dbus_signal__interface_added__details_string = "[*] Interfaces Removed Signal Caught\n\tDevice Path:\t{0}".format(path)
        # Iterate through received information
        if 'Address' in device_properties:
            dbus_signal__interface_added__details_string += "\n\tAddress:\t{0}".format(bluetooth_utils.dbus_to_python(device_properties['Address']))
        if 'Name' in device_properties:
            dbus_signal__interface_added__details_string += "\n\tDevice Name:\t{0}".format(bluetooth_utils.dbus_to_python(device_properties['Name']))
        if 'Alias' in device_properties:
            dbus_signal__interface_added__details_string += "\n\tDevice Alias:\t{0}".format(bluetooth_utils.dbus_to_python(device_properties['Alias']))
        if 'RSSI' in device_properties:
            dbus_signal__interface_added__details_string += "\n\tDevice RSSI:\t{0}".format(bluetooth_utils.dbus_to_python(device_properties['RSSI']))
        if 'TxPower' in device_properties:
            dbus_signal__interface_added__details_string += "\n\tTx Power:\t{0}".format(bluetooth_utils.dbus_to_python(device_properties['TxPower']))
        logging__log_event(LOG__GENERAL, dbus_signal__interface_added__details_string)
        if dbg != 0:
            print(dbus_signal__interface_added__details_string)

    # Function for Catching an Interface Removed Signal
    def callback__signal__interface_removed(self, path, interfaces):
        # Check that the interface is the expected Bluetooth Device Interface
        if not bluetooth_constants.DEVICE_INTERFACE in interfaces:
            return
        # Extract the Device Properties
        device_properties = interfaces[bluetooth_constants.DEVICE_INTERFACE]
        ## Examining the Device Properties for specifics
        # Create logging string for general logging
        dbus_signal__interface_added__details_string = "[*] Interfaces Added Signal Caught\n\tDevice Path:\t{0}\n\tAddress:\t{1}\n\tDevice Name:\t{2}\nDevice RSSI:\t{3}".format(path, bluetooth_utils.dbus_to_python(device_properties['Address']), bluetooth_utils.dbus
_to_python(device_properties['Name']), bluetooth_utils.dbus_to_python(device_properties['RSSI']))
        logging__log_event(LOG__GENERAL, dbus_signal__interface_added__details_string)
        if dbg != 0:
            print(dbus_signal__interface_added__details_string)

    # Function for Catching a Bluez Device Properties Changed Signal
    def callback__signal__bluez__device__properties_changed(self, interface, changed, invalidated, path):
        # Check that the interface is the expected Bluetooth Device Interface
        if interface != bluetooth_constants.DEVICE_INTERFACE:
            return
        ## Examining the Changed Information
        dbus_signal__properties_changed__details_string = "[*] Properties Changed Signal Caught\n\tDevice Path:\t{0}".format(path)
        # Iterate through the changed properties; Note not all information might be present (all signals suffer from this?)
        for changed_property in changed.items():
            # Add the items to the string
            if 'Address' in changed_property:
                dbus_signal__properties_changed__details_string += "\n\tAddress:\t{0}".format(bluetooth_utils.dbus_to_python(changed.items()[changed_property]))
            if 'Name' in changed_property:
                dbus_signal__properties_changed__details_string += "\n\tDevice Name:\t{0}".format(bluetooth_utils.dbus_to_python(changed.items()[changed_property]))
            if 'RSSI' in changed_property:
                dbus_signal__properties_changed__details_string += "\n\tDevice RSSI:\t{0}".format(bluetooth_utils.dbus_to_python(changed.items()[changed_property]))
            if 'TxPower' in changed_property:
                dbus_signal__properties_changed__details_string += "\n\tTx Power:\t{0}".format(bluetooth_utils.dbus_to_python(changed.items()[changed_property]))
        # Add the information to the general log
        logging__log_event(LOG__GENERAL, dbus_signal__properties_changed__details_string)
```

Figure 42: Signals Class - Signal Processing

Emissions - Signals + Notifications - Class Definition

```
# Function for Catching a Bluez Device Properties Changed Signal
def callback__signal__bluez__device__properties_changed(self, interface, changed, invalidated, path):
    # Check that the interface is the expected Bluetooth Device Interface
    if interface != bluetooth_constants.DEVICE_INTERFACE:
        return
    ## Examining the Changed Information
    dbus_signal__properties_changed__details_string = "[*] Properties Changed Signal Caught\n\tDevice Path:\t{0}".format(path)
    # Iterate through the changed properties; Note not all information might be present (all signals suffer from this?)
    for changed_property in changed.items():
        # Add the items to the string
        if 'Address' in changed_property:
            dbus_signal__properties_changed__details_string += "\n\tAddress:\t{0}".format(bluetooth_utils.dbus_to_python(changed.items()[changed_property]))
        if 'Name' in changed_property:
            dbus_signal__properties_changed__details_string += "\n\tDevice Name:\t{0}".format(bluetooth_utils.dbus_to_python(changed.items()[changed_property]))
        if 'RSSI' in changed_property:
            dbus_signal__properties_changed__details_string += "\n\tDevice RSSI:\t{0}".format(bluetooth_utils.dbus_to_python(changed.items()[changed_property]))
        if 'TxPower' in changed_property:
            dbus_signal__properties_changed__details_string += "\n\tTx Power:\t{0}".format(bluetooth_utils.dbus_to_python(changed.items()[changed_property]))
    # Add the information to the general log
    logging__log_event(LOG__GENERAL, dbus_signal__properties_changed__details_string)
```

Figure 43: Signals Class - Signal Processing - Zoom on Properties Changed

```
# Check that the interface is the expected Bluetooth De
if interface != bluetooth_constants.DEVICE_INTERFACE:
    return
## Examining the Changed Information
dbus_signal__properties_changed__details_string = "[*]
# Iterate through the changed properties; Note not all
```

Figure 44: Signals Class - Signal Processing - Zoom on Properties Changed - Check Interface

Emissions - Signals + Notifications - Class Definition

```
# Iterate through the changed properties; Note not all information m
for changed_property in changed.items():
    # Add the items to the string
    if 'Address' in changed_property:
        dbus_signal__properties_changed__details_string += "\n\tAddr
    if 'Name' in changed_property:
        dbus_signal__properties_changed__details_string += "\n\tDevi
    if 'RSSI' in changed_property:
        dbus_signal__properties_changed__details_string += "\n\tDevi
    if 'TxPower' in changed_property:
        dbus_signal__properties_changed__details_string += "\n\tTx P
# Add the information to the general log
logging__log_event(LOG__GENERAL, dbus_signal__properties_changed__de
```

Figure 45: Signals Class - Signal Processing - Zoom on Properties Changed - Properties

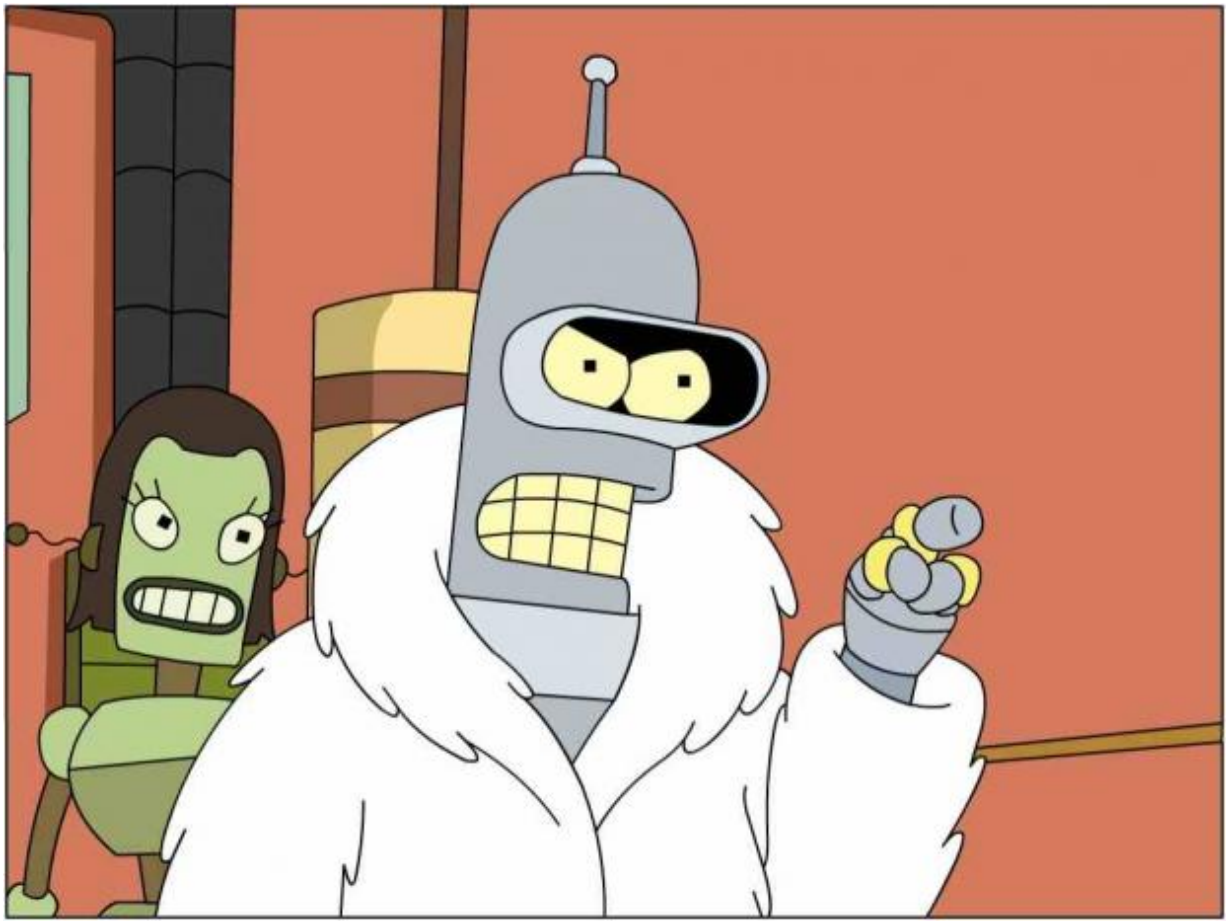
Goal Tracking - Phz3

Research – Phase Three - Goals	
Goal Posts	**Completion Metric**
Augment framework for multi-reads	✓
Augment framework for brute-force writes	✓
Augment framework for specific write-types	✓
~~Learning D-Bus signature decoding / translation~~	~~20%~~
Expand framework to allow capture of signals / notifications	50%
Learn signal emittion & capture	65%
Establish callback functionality	40%
Automate discovery + enumeration	
Basic logging capability for offline analysis	

Figure 46: Goal Table - Phase Three - Learning Signal - Take Three

BLE C[w]TF to Pico W - Signals Suck....

- Not seeing BLE CTF signals!?!?!!!
- Make a newer, better BLE server!



Signal-ing Madness Herein

- Begin w/ attempt to **only** monitor the hci0 BlueZ device
  - `sudo dbus-monitor --system ‘‘path=‘/org/bluez/hci0’ ’’`
- Return of the GLib...
  - While notificaiton signals (e.g. PropertiesChanged) arrive on D-Bus after .StartNotify() method call, signal receiver not actively get data until GLib MainLoop running
  - **MAJOR** pain point in getting signals off the ground
- The use of ‘path_keyword=“path” ’ within add_signal_receiver() **sets** what the name of the variable that will contain the path information

Emissions - Signals + Notifications - Class Definition - Single Thread Catching

```
  # Function for running the actual capture of notification signals
  def run__signal_catch__timed(self, notifying_interface, callback_function, signal_to_catch="PropertiesChanged"):
      ## Configuration of the Interface
      # Add the information to the general log
      dbus__signal_catch__start_string = "[*] Starting D-Bus Signal Catching"
      logging__log_event(LOG__GENERAL, dbus__signal_catch__start_string)
      # Setup the notifications capture
      #self.mainloop__notifications__capture(self, notification_interface, callback_function, dbus_interface_to_watch=bluetooth_constants.DBUS_PROPERTIES, signal_to_catch="PropertiesChanged",
      # Note: Passing the defaults to the function above; Note getting an error about "got multiple values for argument 'dbus_interface_to_watch'"
      #   - The issue is that because the 'dbus_interface_to_watch' variable is passed as "dbus_interface_to_watch=bluetooth_constants.DBUS_PROPERTIES" which means Python interprets this as a
definition)
      self.mainloop__notifications__capture(callback_function)
      # dbus_interface_to_watch=bluetooth_constants.DBUS_PROPERTIES, signal_to_catch="PropertiesChanged", listening_time__ms=5000)

      # Add the information to the general log
      dbus__signal_catch__start_notify = "[*] Starting Notifications for Interface"
      logging__log_event(LOG__GENERAL, dbus__signal_catch__start_notify)

      # Start the notification on the given characteristic; (Does NOT matter what order I do these actions in?)
      notifying_interface.StartNotify()

      # Add the information to the general log
      dbus__signal_catch__create_mainloop = "[*] Creating GLib MainLoop"
      logging__log_event(LOG__GENERAL, dbus__signal_catch__create_mainloop)

      # Create MainLoop GLib Object for use below
      mainloop = GLib.MainLoop()

      # Add the information to the general log
      dbus__signal_catch__adding_timeout = "[*] Adding Timeout to GLib MainLoop"
      logging__log_event(LOG__GENERAL, dbus__signal_catch__adding_timeout)

      # Add the Timer to GLib
      self.mainloop__configure__add_timeout(self.mainloop_run__default_time__ms, self.timer__signal_catch, mainloop)
      # Note the massing of the mainloop structure to the timeout callback function; hence needing to call this here within the function (after mainloop is defined)

      ## Running the MainLoop
      ## Running the MainLoop            <----- This MUST be in the MAIN TIMED FUNCTION; MOVE THIS AND ABOVE TO THE RUNNING TIMED LISTEN FUNCTION
      # Loop for listening on the D-Bus for Notify signals (e.g. "PropertiesChanged")
      try:
          if dbg != 0:
              print("[*] Listening for Emittions...")
          # Add the information to the general log
          dbus__signal_catch__mainloop_start = "[*] Starting GLib MainLoop"
          logging__log_event(LOG__GENERAL, dbus__signal_catch__mainloop_start)
          mainloop.run()
      except KeyboardInterrupt:
          mainloop.quit()
          if dbg != 0:
              print("[+] Done Listening")
          # Add the information to the general log
          dbus__signal_catch__mainloop_stop = "[+] GLib MainLoop Succsesfully Stopped"
          logging__log_event(LOG__GENERAL, dbus__signal_catch__mainloop_stop)
      # TODO: Turn the above into an internal function
```

Emissions - Signals + Notifications - Class Definition - Single Thread Catching

```
 #    - The issue is that because the 'dbus_interface_to_wat
ition)
 self.mainloop__notifications__capture(callback_function)
 # dbus_interface_to_watch=bluetooth_constants.DBUS_PROPERT

 # Add the information to the general log
 dbus__signal_catch__start_notify = "[*] Starting Notificat
 logging__log_event(LOG__GENERAL, dbus__signal_catch__start

 # Start the notification on the given characteristic; (Doe
 notifying_interface.StartNotify()
```

Figure 48: Signals Class - Timed Catch - Zoom on Setup

Emissions - Signals + Notifications - Class Definition - Single Thread Catching

```
# Create MainLoop GLib Object for use below
mainloop = GLib.MainLoop()

# Add the information to the general log
dbus__signal_catch__adding_timeout = "[*] Adding Timeout to GLib MainLoop"
logging__log_event(LOG__GENERAL, dbus__signal_catch__adding_timeout)

# Add the Timer to GLib
self.mainloop__configure__add_timeout(self.mainloop_run__default_time__ms, self.timer__signal_catch, mainloop)
# Note the massing of the mainloop structure to the timeout callback function; hence needing to call this here
```

Figure 49: Signals Class - Timed Catch - Zoom on GLib Configure

Emissions - Signals + Notifications - Class Definition - Single Thread Catching

```
# Loop for listening on the D-Bus for Notify signals (e.g. "PropertiesChanged")
try:
    if dbg != 0:
        print("[*] Listening for Emittions...")
    # Add the information to the general log
    dbus__signal_catch__mainloop_start = "[*] Starting GLib MainLoop"
    logging__log_event(LOG__GENERAL, dbus__signal_catch__mainloop_start)
    mainloop.run()
except KeyboardInterrupt:
    mainloop.quit()
    if dbg != 0:
        print("[+] Done Listening")
    # Add the information to the general log
    dbus__signal_catch__mainloop_stop = "[+] GLib MainLoop Succsesfully Stopped"
    logging__log_event(LOG__GENERAL, dbus__signal_catch__mainloop_stop)
```

Figure 50: Signals Class - Timed Catch - Zoom on GLib Run

Emissions - Signals + Notifications - Class Definition - Single Thread Catching

```
## Clean-up of Notification Capture
# Stop the notification signals
notifying_interface.StopNotify()
```

Figure 51: Signals Class - Timed Catch - Zoom on Stop Notification

Goal Tracking - Phz3

Research – Phase Three - Goals	
Goal Posts	**Completion Metric**
Augment framework for multi-reads	✓
Augment framework for brute-force writes	✓
Augment framework for specific write-types	✓
~~Learning D-Bus signature decoding / translation~~	~~20%~~
Expand framework to allow capture of signals / notifications	85%
Learn signal emittion & capture	✓
Establish callback functionality	✓
Threading to allow non-blocking signal capture	
Automate discovery + enumeration	
Basic logging capability for offline analysis	✓

Signal-ing Madness Herein

```
[+] connect_and_enumeration__bluetooth__low_energy::Device services resolved
Device  D8:3A:DD:1F:D0:BC
        Properties:             {dbus.String('Address'): 'D8:3A:DD:1F:D0:BC', dbus.String('AddressType'): 'public', dbus.String('Name'): 'MPY BTSTACK', dbus.String('Alias'): 'MPY BTSTACK', dbus.String('Paired'): False, dbus.String('Bonded'): False, dbus.String('Tru
sted'): False, dbus.String('Blocked'): False, dbus.String('LegacyPairing'): False, dbus.String('Connected'): True, dbus.String('UUIDs'): ['00001800-0000-1000-8000-00805f9b34fb', '00001801-0000-1000-8000-00805f9b34fb', '0003217e-0000-1111-2222-333344445555', '13372e
ad-0000-1111-2222-333344445555', '6e400001-b5a3-f393-e0a9-e50e24dcca9e'], dbus.String('Adapter'): '/org/bluez/hci0', dbus.String('ServicesResolved'): True}
        Address         -       Value:  D8:3A:DD:1F:D0:BC
        AddressType             -       Value:  public
        Name            -       Value:  MPY BTSTACK
        Alias           -       Value:  MPY BTSTACK
        Paired          -       Value:  False
        Bonded          -       Value:  False
        Trusted         -       Value:  False
        Blocked         -       Value:  False
        LegacyPairing           -       Value:  False
        Connected               -       Value:  True
        UUIDs           -       Value:  ['00001800-0000-1000-8000-00805f9b34fb', '00001801-0000-1000-8000-00805f9b34fb', '0003217e-0000-1111-2222-333344445555', '13372ead-0000-1111-2222-333344445555', '6e400001-b5a3-f393-e0a9-e50e24dcca9e']
        Adapter         -       Value:  /org/bluez/hci0
        ServicesResolved                -       Value:  True
[*] Characteristic [ /org/bluez/hci0/dev_D8_3A_DD_1F_D0_BC/service0004/char0005 ] Flags:       ['read']
[*] Characteristic [ /org/bluez/hci0/dev_D8_3A_DD_1F_D0_BC/service0007/char0008 ] Flags:       ['read', 'notify']
[*] Characteristic [ /org/bluez/hci0/dev_D8_3A_DD_1F_D0_BC/service0007/char000b ] Flags:       ['write-without-response', 'write']
[!] Error: May not have permission for R/W; perhaps need more than just connection to device? OR the 'read' capability does not exist at the target
        org.bluez.Error.NotPermitted: Read not permitted
[-] connect_and_enumerate__bluetooth__low_energy::Error Read Not Permitted for Target Characteristic
[*] Characteristic [ /org/bluez/hci0/dev_D8_3A_DD_1F_D0_BC/service000d/char000e ] Flags:       ['read']
[*] Characteristic [ /org/bluez/hci0/dev_D8_3A_DD_1F_D0_BC/service000d/char0010 ] Flags:       ['read']
[*] Characteristic [ /org/bluez/hci0/dev_D8_3A_DD_1F_D0_BC/service0012/char0013 ] Flags:       ['write-without-response', 'write']
[!] Error: May not have permission for R/W; perhaps need more than just connection to device? OR the 'read' capability does not exist at the target
        org.bluez.Error.NotPermitted: Read not permitted
[-] connect_and_enumerate__bluetooth__low_energy::Error Read Not Permitted for Target Characteristic
[*] Characteristic [ /org/bluez/hci0/dev_D8_3A_DD_1F_D0_BC/service0012/char0015 ] Flags:       ['write-without-response', 'write']
[!] Error: May not have permission for R/W; perhaps need more than just connection to device? OR the 'read' capability does not exist at the target
        org.bluez.Error.NotPermitted: Read not permitted
[-] connect_and_enumerate__bluetooth__low_energy::Error Read Not Permitted for Target Characteristic
[=] Warning! The generated Device Map will only be a skeleton of the target device. Reads have NOT been performed against the device yet...
[+] BLE Class::Reconnect_Check  -       Device [D8:3A:DD:1F:D0:BC] is already connected
Service UUID:   00001801-0000-1000-8000-00805f9b34fb            -               Generic Attribute Service               -               [service0004]
        Characteristic UUID:    00002a05-0000-1000-8000-00805f9b34fb            -               Service Changed         -       [char0005]
        Characteristic Flags:   ['read']
        Characteristic Value (ASCii):   None
Service UUID:   6e400001-b5a3-f393-e0a9-e50e24dcca9e            -               Unknown         -               [service0007]
        Characteristic UUID:    6e400003-b5a3-f393-e0a9-e50e24dcca9e            -               Unknown         -       [char0008]
        Characteristic Flags:   ['read', 'notify']
        Characteristic Value (ASCii):   None
                Descriptor UUID:        00002902-0000-1000-8000-00805f9b34fb            -               Client Characteristic Configuration             -               [desc000a]
                Descriptor Value:       []
        Characteristic UUID:    6e400002-b5a3-f393-e0a9-e50e24dcca9e            -               Unknown         -       [char000b]
        Characteristic Flags:   ['write-without-response', 'write']
                [-] Writes not being attempted  -       Passive Scan
Service UUID:   13372ead-0000-1111-2222-333344445555            -               Unknown         -               [service000d]
        Characteristic UUID:    13372ead-0001-1111-2222-333344445555            -               Unknown         -       [char000e]
        Characteristic Flags:   ['read']
        Characteristic Value (ASCii):   R-Serv Char 01
        Characteristic UUID:    13372ead-0002-1111-2222-333344445555            -               Unknown         -       [char0010]
        Characteristic Flags:   ['read']
        Characteristic Value (ASCii):   R-Serv Char Var
Service UUID:   0003217e-0000-1111-2222-333344445555            -               Unknown         -               [service0012]
        Characteristic UUID:    0003217e-0001-1111-2222-333344445555            -               Unknown         -       [char0013]
        Characteristic Flags:   ['write-without-response', 'write']
                [-] Writes not being attempted  -       Passive Scan
        Characteristic UUID:    0003217e-0002-1111-2222-333344445555            -               Unknown         -       [char0015]
        Characteristic Flags:   ['write-without-response', 'write']
                [-] Writes not being attempted  -       Passive Scan
[+] Emittion Listening Completed
[+] Completed Notify Testing
[+] Finished Main()
[duncan@ArtII BTInteract]$
```

Figure 53: Signal Testing - Pico W Server

Signal-ing Madness Herein

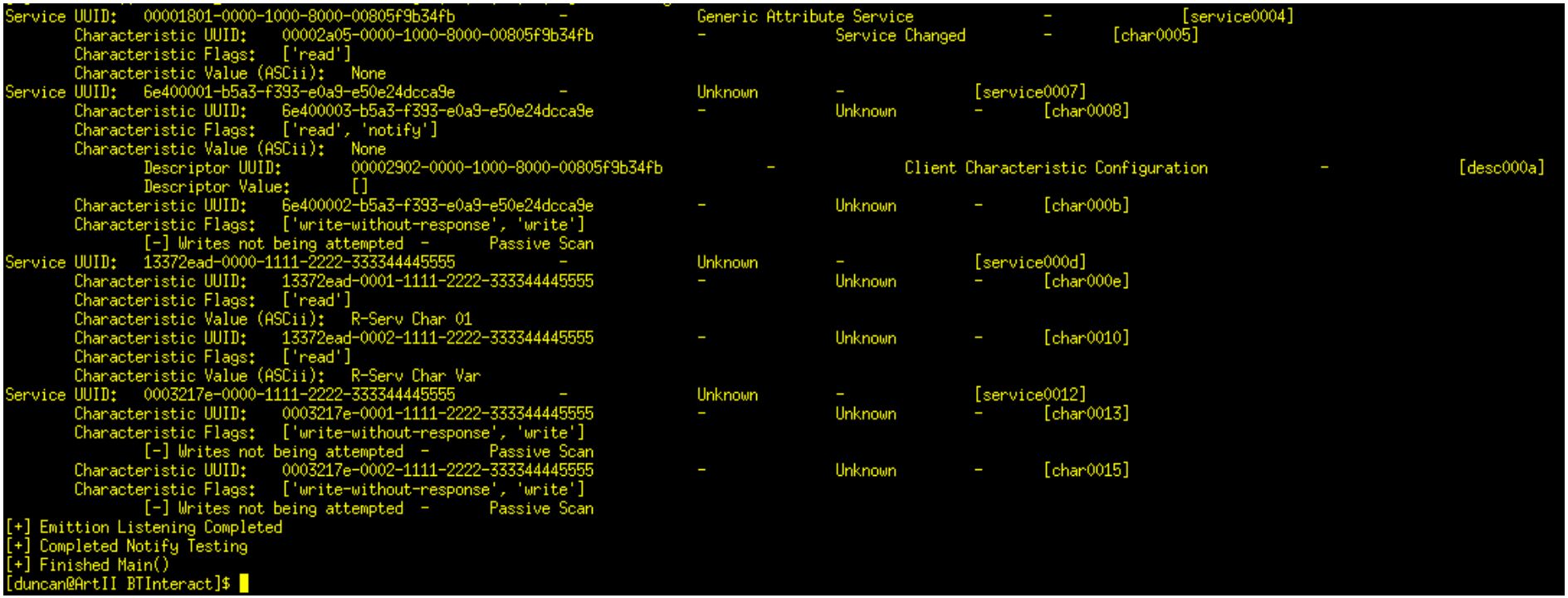
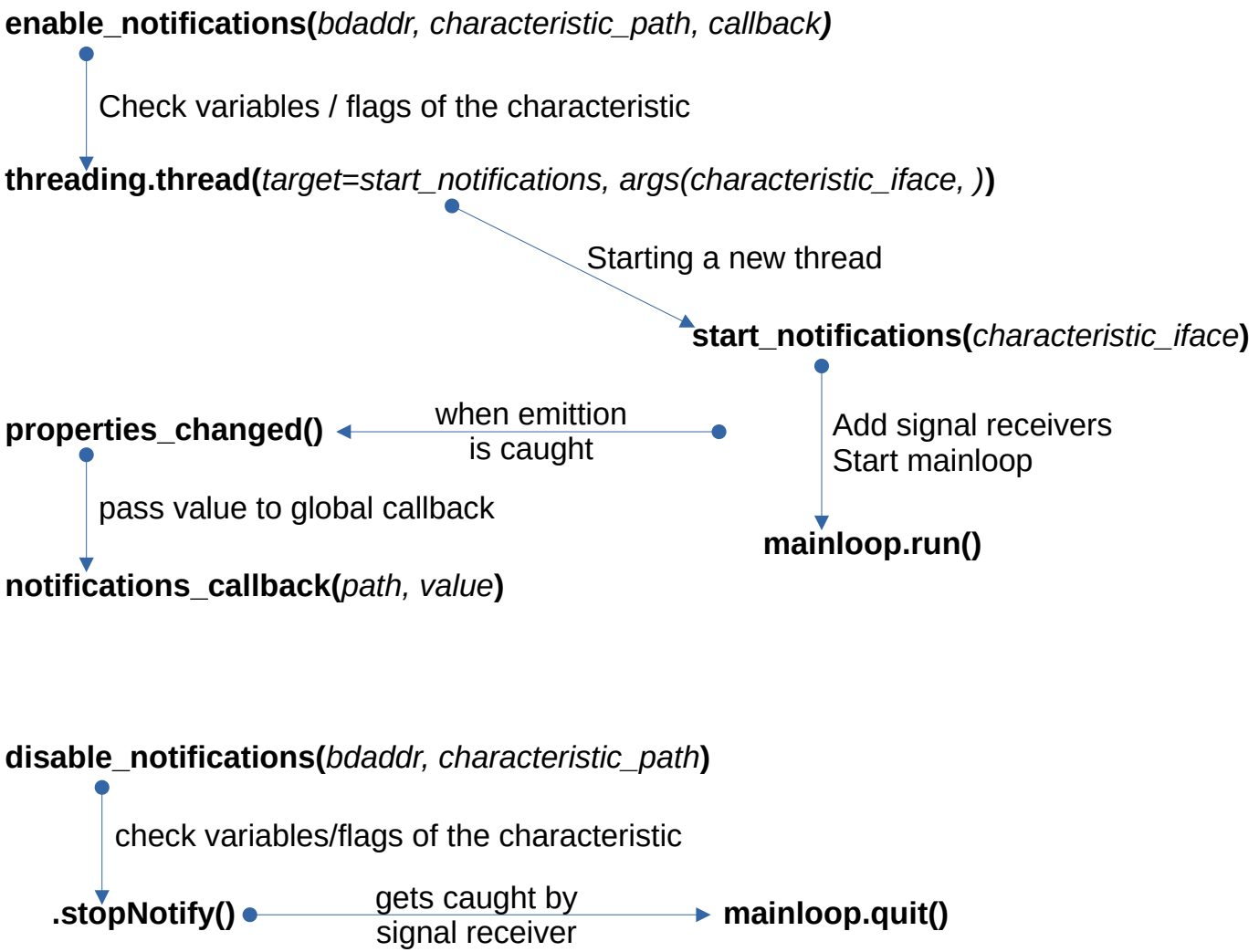


Figure 54: Signal Testing - Pico W Server - Zoom In

Figure 55: Signal Testing - Pico W Server - Double Zoom

Emissions - Signals + Notifications - BT SIG Example Flow



Emissions - Signals + Notifications - Pseudo Code

- Functionality Requirements:
  - Dissection of “PropertiesChanged” Emittion
  - Function to stop the GLib MainLoop
  - Function to configure D-Bus & GLib MainLoop
  - Function to validate the passed characteristic
  - Function to configure & run the thread
  - Function to act on Emission’s data

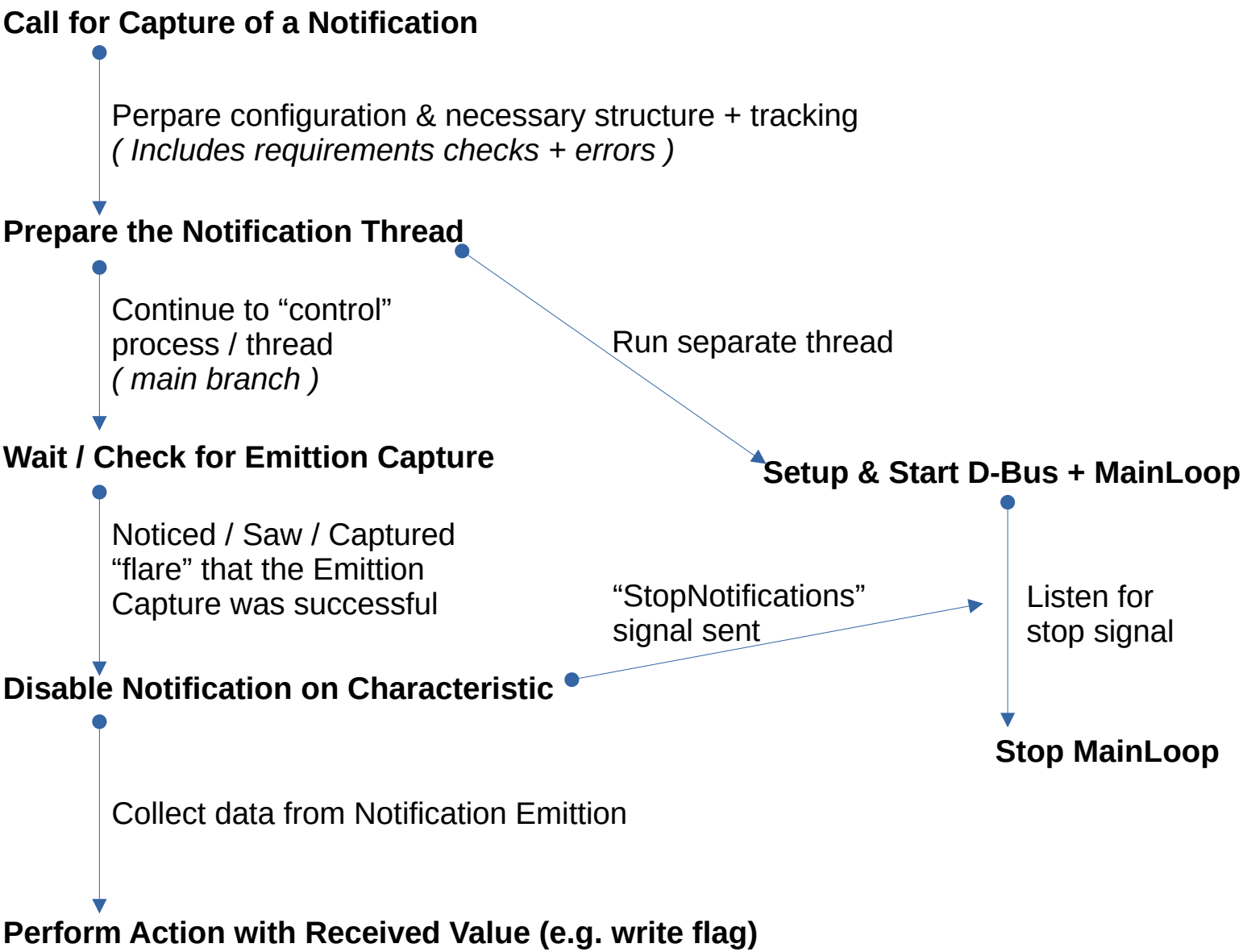
Goal Tracking - Phz3

Research – Phase Three - Goals	
Goal Posts	**Completion Metric**
Augment framework for multi-reads	✓
Augment framework for brute-force writes	✓
Augment framework for specific write-types	✓
~~Learning D-Bus signature decoding / translation~~	~~20%~~
Expand framework to allow capture of signals / notifications	85%
Learn signal emittion & capture	✓
Establish callback functionality	✓
Threading to allow non-blocking signal capture	???
Automate discovery + enumeration	
Basic logging capability for offline analysis	✓

Emissions - Signals + Notifications - Pseudo Code

- Verifying PoC:
  - Determine what Service/Characteristic/Descriptor (S/C/D) desired for notification capture
    - Bonus: Decide emission capture once or continuous
  - Perform requirement/error check on provided S/C/D
    - Prepare required configuration & tracking structures
  - Configure thread for notification capture
    - Include **both** callback function & teardown process (e.g. disbale notification, stop MainLoop)
  - Check/Verify "Action" was performed using received value

Emissions - Signals + Notifications - Developed Flow



Emissions - Signals + Notifications - Class Definition

```
# Function for Testing the Above Notification + Threads + Callback outlined above in the scratch space
def capture_and_act__emittion__gatt_characteristic(self, user_device_object, characteristic_name):
    # Determine what element (e.g. S/C/D) is desired to capture a notification from
    #   - Bonus:    Decide if emittion capture once or continuous
    ## TODO: Add step(s) here

    # Perform requirement/error check on provided S/C/D + Generate the Characteristic Interface needed
    #   - Prepare required configuration and tracking structures
    print("[*] Creating Characteristic Structures to Emittion Capture")
    characteristic_properties, characteristic_interface = self.validate__dbus__gatt_characteristic(user_device_object, characteristic_name)
    #sanity_check = self.validate__dbus__gatt_characteristic(user_device_object, characteristic_name)
    #if not sanity_check:
    #    print("[-] Characteristic [ {0} ] Failed Pre-Check".format(characteristic_name))
    #    return
    if characteristic_interface is None:
        print("[-] No Characteristic Interface Generated for [ {0} ]".format(characteristic_name))
        return

    # Configure the Thread for notification capture
    #   - Include BOTH callback function and teardown process (e.g. disable notification, stop MainLoop)
    print("[*] Configuring and running the Thread")
    self.threads__configure_and_run(self.start_notification, characteristic_interface)      # Note: This call to the threads function will set the receivers, start notification, and run the MainLoop
    # Function below gets used how?? Gets called by the above function, which will pass the characteristic_interface to it as well
    #start_notification(self, characteristic_interface):

    # Artifical Timing on the Notification Running/Being Captured
    print("[*] Waiting 20 Seconds....\n\tStart:\t{0}".format(time.ctime()))
    time.sleep(20)
    print("\tEnd:\t{0}\n[+] Done waiting time".format(time.ctime()))

    # Stop the Notification on the Characteristic Interface
    print("[*] Validating Characteristic to Stop Notification")
    self.validate__dbus__gatt_characteristic__pre_stop(user_device_object, characteristic_properties, characteristic_interface)
    print("[*] Stopping Notification on the Characteristic Interface")
    self.stop_notification(characteristic_interface)
```

Figure 59: Signals Class - Threaded Catch

Emissions - Signals + Notifications - Class Definition

```
print("[*] Creating Characteristic Structures to Emittion Capture")
characteristic_properties, characteristic_interface = self.validate__dbus__gatt_characteristic(user_device_object, characteristic_name)
#sanity_check = self.validate__dbus__gatt_characteristic(user_device_object, characteristic_name)
#if not sanity_check:
#    print("[-] Characteristic [ {0} ] Failed Pre-Check".format(characteristic_name))
#    return
if characteristic_interface is None:
    print("[-] No Characteristic Interface Generated for [ {0} ]".format(characteristic_name))
    return
```

Figure 60: Signals Class - Threaded Catch - Zoom on Initialization

Emissions - Signals + Notifications - Class Definition

```
# Configure the Thread for notification capture
#   - Include BOTH callback function and teardown process (e.g. disable notification, stop MainLoop)
print("[*] Configuring and running the Thread")
self.threads__configure_and_run(self,start_notification, characteristic_interface)      # Note: This
```

Figure 61: Signals Class - Threaded Catch - Zoom on Thread Configuration

Emissions - Signals + Notifications - Class Definition

```
# Artifical Timing on the Notification Running/Being Captured
print("[*] Waiting 20 Seconds....\n\tStart:\t{0}".format(time.ctime()))
time.sleep(20)
print("\tEnd:\t{0}\n[+] Done waiting time".format(time.ctime()))
```

Figure 62: Signals Class - Threaded Catch - Zoom on Forced Wait

Emissions - Signals + Notifications - Class Definition

```
# Stop the Notification on the Characteristic Interface
print("[*] Validating Characteristic to Stop Notification")
self.validate_dbus_gatt_characteristic_pre_stop(user_device_object, characteristic_properties, characteristic_interface)
print("[*] Stopping Notification on the Characteristic Interface")
self.stop_notification(characteristic_interface)
```

Figure 63: Signals Class - Threaded Catch- Zoom on Final Validation and Stop

Automation + Logging

- Automated functionality for signal receiver, signal generation, and other necessary structures
- Implemented logging operation for post-operation analysis

Automation + Logging

```
[!] Introspection E-Tree Information:          <Element 'node' at 0x7de56403b600>
Root Tag:       [ node ]                -              Root Attrib:    [ {} ]
       Child Tag:      [ interface ]           -              Child Attrib:   [ {'name': 'org.freedesktop.DBus.Introspectable'} ]
       Child Tag:      [ interface ]           -              Child Attrib:   [ {'name': 'org.bluez.GattCharacteristic1'} ]
       Child Tag:      [ interface ]           -              Child Attrib:   [ {'name': 'org.freedesktop.DBus.Properties'} ]
[*] Listening for Emittions...
[!] Received Signal! Debugging Signal   -       Catchall
Args:
       Interface:      org.bluez.GattCharacteristic1
       Changed:        dbus.Dictionary({dbus.String('Value'): dbus.Array([dbus.Byte(49), dbus.Byte(49), dbus.Byte(48), dbus.Byte(56), dbus.Byte(50), dbus.Byte(95)], signature=dbus.Signature('y'), variant_level=1)}, signature=dbus.Signature('sv'))
       Invalidated:    dbus.Array([], signature=dbus.Signature('s'))
Kwargs:
[+] Signal Debug Complete
[!] Received Signal! Debugging Signal   -       Catchall
Args:
       Interface:      org.bluez.GattCharacteristic1
       Changed:        dbus.Dictionary({dbus.String('Value'): dbus.Array([dbus.Byte(49), dbus.Byte(49), dbus.Byte(48), dbus.Byte(56), dbus.Byte(51), dbus.Byte(95)], signature=dbus.Signature('y'), variant_level=1)}, signature=dbus.Signature('sv'))
       Invalidated:    dbus.Array([], signature=dbus.Signature('s'))
Kwargs:
[+] Signal Debug Complete
[!] Received Signal! Debugging Signal   -       Catchall
Args:
       Interface:      org.bluez.GattCharacteristic1
       Changed:        dbus.Dictionary({dbus.String('Value'): dbus.Array([dbus.Byte(49), dbus.Byte(49), dbus.Byte(48), dbus.Byte(56), dbus.Byte(52), dbus.Byte(95)], signature=dbus.Signature('y'), variant_level=1)}, signature=dbus.Signature('sv'))
       Invalidated:    dbus.Array([], signature=dbus.Signature('s'))
Kwargs:
[+] Signal Debug Complete
[!] Received Signal! Debugging Signal   -       Catchall
Args:
       Interface:      org.bluez.GattCharacteristic1
       Changed:        dbus.Dictionary({dbus.String('Notifying'): dbus.Boolean(True, variant_level=1)}, signature=dbus.Signature('sv'))
       Invalidated:    dbus.Array([], signature=dbus.Signature('s'))
Kwargs:
[+] Signal Debug Complete
[!] Received Signal! Debugging Signal   -       Catchall
Args:
       Interface:      org.bluez.GattCharacteristic1
       Changed:        dbus.Dictionary({dbus.String('Value'): dbus.Array([dbus.Byte(49), dbus.Byte(49), dbus.Byte(48), dbus.Byte(56), dbus.Byte(53), dbus.Byte(95)], signature=dbus.Signature('y'), variant_level=1)}, signature=dbus.Signature('sv'))
       Invalidated:    dbus.Array([], signature=dbus.Signature('s'))
Kwargs:
[+] Signal Debug Complete
[!] Received Signal! Debugging Signal   -       Catchall
Args:
       Interface:      org.bluez.GattCharacteristic1
       Changed:        dbus.Dictionary({dbus.String('Value'): dbus.Array([dbus.Byte(49), dbus.Byte(49), dbus.Byte(48), dbus.Byte(56), dbus.Byte(54), dbus.Byte(95)], signature=dbus.Signature('y'), variant_level=1)}, signature=dbus.Signature('sv'))
       Invalidated:    dbus.Array([], signature=dbus.Signature('s'))
Kwargs:
[+] Signal Debug Complete
```

Figure 64: Signal Testing - Debug Logging

Automation + Logging

```
[!] Received Signal! Debugging Signal   -       Catchall
Args:
        Interface:      org.bluez.GattCharacteristic1
        Changed:        dbus.Dictionary({dbus.String('Value'): dbus.Array([dbus.Byte(49), dbu
        Invalidated:    dbus.Array([], signature=dbus.Signature('s'))
Kwargs:
[+] Signal Debug Complete
```

Figure 65: Signal Testing - Debug Logging

Goal Tracking - Phz3

Research – Phase Three - Goals	
Goal Posts	**Completion Metric**
Augment framework for multi-reads	✓
Augment framework for brute-force writes	✓
Augment framework for specific write-types	✓
~~Learning D-Bus signature decoding / translation~~	~~20%~~
Expand framework to allow capture of signals / notifications	✓
Learn signal emittion & capture	✓
Establish callback functionality	✓
Threading to allow non-blocking signal capture	✓
Automate discovery + enumeration	✓
Basic logging capability for offline analysis	✓

State of the Code

- Devices discovery and enumeration
- Read, Write, and Signal Capture ***
- Automation, logging, and UI

Lessons / Observations

- Make, take, and create clear documentation
  - Can you understand it in a week? month? year?
- Odd Behavior
  - Double read the GATT, Initialized GATT cannot be altered
  - D-Bus willingly forgets
  - W to Arduino Descriptors = panic crash
  - Default buffer character limit
- How to create and customize a basic BLE server
  - Remove blind testing and finger-crossing
  - **Note:** Not all BLE are the same!!

Horizons of the Landscape

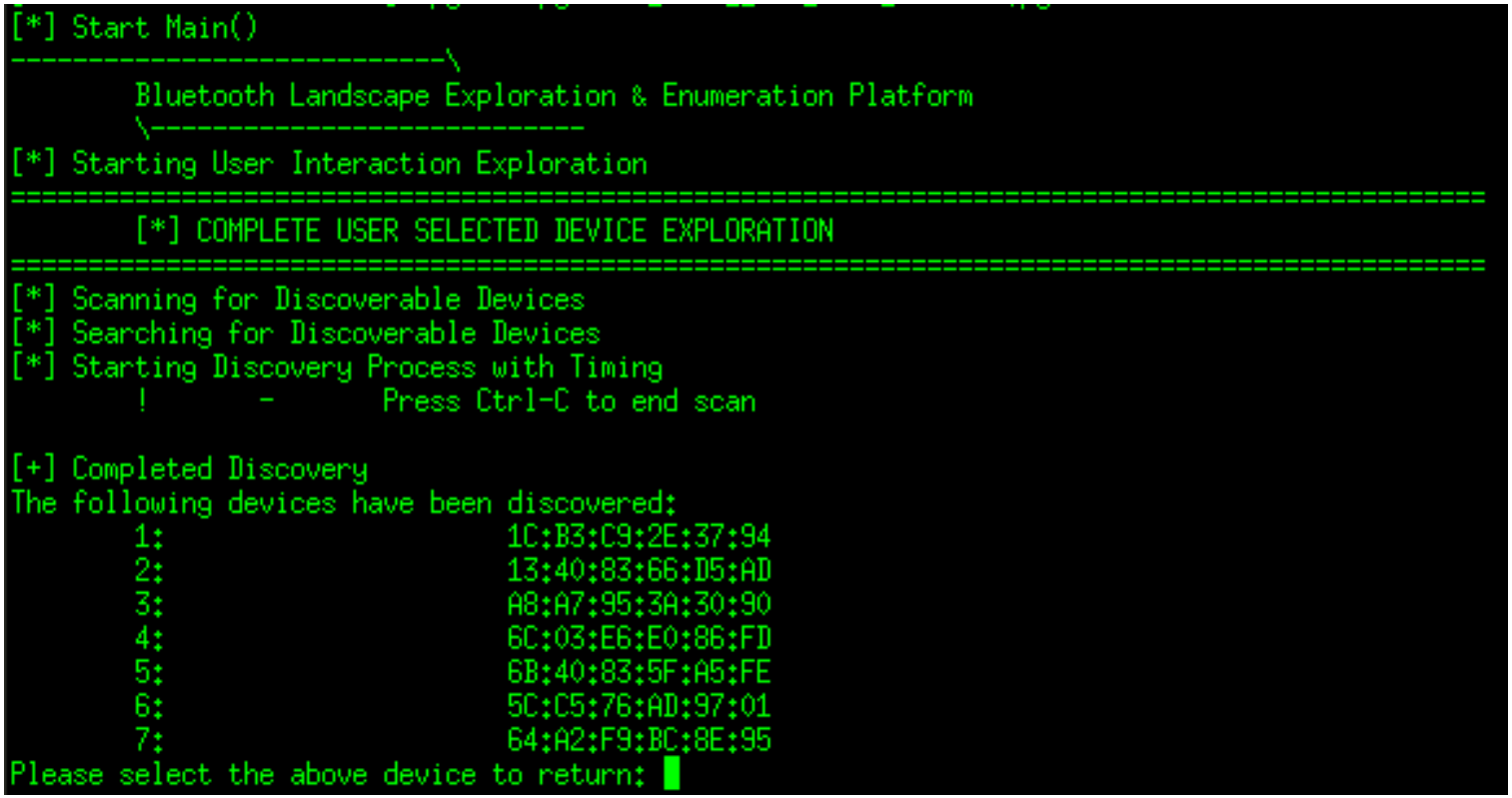
- Survey the Bluetooth Low Energy Landscape
- Augment encryption + security capabilities
- Automate enumeration and dissection of findings

Questions

Questions?

Research Details

- Git Repo: https://github.com/Mauddib28/bleep-tool



Bibliography References I

Freedesktop

What is D-Bus, Published 2022

https://www.freedesktop.org/wiki/Software/dbus/

Last Accessed: 2024-03-28 22:43:19 EST

GNU

Knowing the Details of D-Bus Services

https://www.gnu.org/software/emacs/manual/html_node/dbus/Introspection.html

Last Accessed: 2024-04-01 19:48:34 EST

Programiz

Python Decorators

https://www.programiz.com/python-programming/decorator

Last Accessed: 2024-04-01 19:52:34 EST

Bibliography References II

hbldh

characteristic.py

https://github.com/hbldh/bleak/blob/63adefa24cb6ed11c8cf154fa41f51ecff1df98c/bleak/backends/characteristic.py

Last Accessed: 2024-04-01 19:56:34 EST

elsampsa

Bluetooth + DBus + gobject demo

https://github.com/elsampsa/btdemo

Last Accessed: 2024-04-01 19:54:52 EST

Freedesktop

DbusTools, Published May 07 2021

https://www.freedesktop.org/wiki/Software/DbusTools/

Last Accessed: 2024-03-28 22:57:29 EST

Bibliography References III


Bluetooth SIG
assigned_numbers
https://bitbucket.org/bluetooth-SIG/public/src/main/assigned_numbers/
Last Accessed: 2024-04-01 21:00:29 EST


Bluetooth SIG
public bitbucket
https://bitbucket.org/bluetooth-SIG/public/src/main/
Last Accessed: 2024-04-02 15:13:33 EST


Archlinux Forum
Activation via systemd failed for unit dbus-org.bluez.service
https://bbs.archlinux.org/viewtopic.php?id=155714
Last Accessed: 2024-04-02 15:09:42 EST

Bibliography References IV

oscaracena

gattlib.h

https://github.com/oscaracena/pygattlib/blob/7d08c0805313201b2ab12628e19544bb180218a8/src/gattlib.h

Last Accessed: 2024-04-02 15:09:42 EST

Bluetooth SIG

Bluetooth for Linux Developers Study Guide - Versions 1.0, 1.0.1

https://www.bluetooth.com/bluetooth-resources/bluetooth-for-linux/

Last Accessed: 2021-12-29 10:26:27 EST, 2022-10-18 18:54:02 EST

Bibliography References V

Bluetooth SIG

Developer Study Guide Bluetooth Internet Gateways - Version 2.0.0

https://www.bluetooth.com/blog/the-bluetooth-internet-gateway-study-guide/

Last Accessed: 2021-07-21 14:46:56 EST

Bluetooth SIG

Bluetooth LE Developer Study Guide - Version 5.2.0

https://www.bluetooth.com/bluetooth-resources/bluetooth-le-developer-starter-kit/

Last Accessed: 2023-02-16 11:36:57 EST

Bibliography References VI

Bluetooth SIG

Bluetooth Core Specification - Versions 5.3, 5.4

https://www.bluetooth.com/specifications/specs/core-specification-5-[3—4]/

Last Accessed: 2023-12-19 13:24:22 EST, 2023-12-16 11:33:37 EST

Bluetooth SIG

Generic Attribute Profile (GATT)

https://www.bluetooth.com/wp-content/uploads/Files/Specification/HTML/Core-54/out/en/host/generic-attribute-profile–gatt-.html

Last Accessed: 2023-12-16 11:22:03 EST

Marcel Holtmann, Maxim Krasnyansky, Qualcomm

BlueZ - Bluetooth protocol stack for Linux

https://git.kernel.org/pub/scm/bluetooth/bluez.git/tree/doc

Last Accessed: 2024-04-11 10:39:23 EST