

The NCID Defender

Matthew Hebrado

Amy Chen

Scott Kevil-Yeager

## **CONCEPT OF OPERATIONS**

## TEAM &lt;28&gt;

---

Rohith KumarDate

## Change Record

Rev.	Date	Originator	Approvals	Description
0	2/9/2022	Matthew Hebrado Amy Chen Scott Kevil-Yeager		CONOPS Report Submission
1	2/23/2022	Matthew Hebrado Amy Chen Scott Kevil-Yeager		Revision 1: suggested changes after grading
2	3/2/2022	Matthew Hebrado Amy Chen Scott Kevil-Yeager		Revision 2: changes for Midterm Report Submission
3	4/30/2022	Matthew Hebrado Amy Chen Scott Kevil-Yeager		Revision 3: changes for Final Report 403 Submission
4	12/2/2022	Matthew Hebrado Amy Chen Scott Kevil-Yeager		Revision 4: changes for Final Report 404 Submission

## Table of Contents

<b>Table of Contents .....</b>	<b>III</b>
<b>List of Tables .....</b>	<b>IV</b>
<b>No table of figures entries found.....</b>	<b>Error! Bookmark not defined.</b>
<b>List of Figures.....</b>	<b>V</b>
<b>No table of figures entries found.....</b>	<b>Error! Bookmark not defined.</b>
<b>1. Executive Summary .....</b>	<b>1</b>
<b>2. Introduction.....</b>	<b>2</b>
2.1. Background.....	2
2.2. Overview .....	3
2.3. Referenced Documents and Standards .....	3
2.3.1. Reference Documents.....	3
2.3.2. Standards.....	3
<b>3. Operating Concept.....</b>	<b>4</b>
3.1. Scope.....	4
3.2. Operational Description and Constraints.....	4
3.3. System Description .....	5
3.3.1. Machine Learning.....	5
3.3.2. Hardware for Interface.....	5
3.3.3. Firmware for Interface .....	5
3.3.4. Database.....	6
3.4. Modes of Operations.....	6
3.4.1. Blacklist.....	6
3.4.2. Whitelist.....	6
3.4.3. Unknown .....	6
3.5. Users .....	6
3.6. Support .....	6
<b>4. Scenario(s) .....</b>	<b>7</b>
4.1. Blacklist Setting.....	7
4.2. Whitelist Setting .....	7
4.3. Unknown Setting.....	7
<b>5. Analysis.....</b>	<b>8</b>
5.1. Summary of Proposed Improvements .....	8
5.2. Disadvantages and Limitations .....	8
5.3. Alternatives .....	8
5.4. Impact.....	8

## **List of Tables**

No table of figures entries found.

## List of Figures

<b>Figure 1. System Block Diagram .....</b>	<b>5</b>
---	----------

## 1. Executive Summary

Many of the elderly population in the United States still use a landline for communication. According to a survey from the CDC, about 40% of adults have a landline phone in their household [1]. Unfortunately, it is common for older residents to be scammed out of their retirement savings by phone scams. Although commodities like commercial call screening devices are available for fraud prevention, scammers continue discovering methods of bypassing.

Our project's objective is to create a device that will work with an existing Open Source Software (Network Caller ID) to filter incoming phone calls continuously. This software will also show Caller ID information in a more functional interface. With the emergence of new technology, there is a need for more devices that collect and present Caller ID information. However, many of these devices are no longer manufactured.

Our device will aim to protect against the 'grandma scam.' This scam is carried out by calling older people and pretending to be their family members while gathering important security information. The scammer will then socially engineer money or financial information from the victim. These scams can go on for long periods before being recognized by the victim or a family member.

Our device will address this scam by implementing a voice signature machine learning algorithm to store data in a local or cloud database to be referenced for future calls or other users. If a voice signature comes from a known scammer, we will be able to flag that voice signature as a known scammer, and the data will be shared across the database. Should a loved one call, the ML algorithm can match their voice to previous calls, confirming with high accuracy that the person calling is on the whitelist and has a matching voice signature to confirm that. Should someone call who is an 'unknown' caller, the system will display to the user that the incoming caller's voice does not match anyone on the whitelist or blacklist. This will ensure that if the 'unknown' caller is a scammer they cannot deceive the victim by impersonating someone else.

## 2. Introduction

Our device will address scams over the internet telephone (SPIT) by creating an in-line landline monitoring system. The system will collect incoming caller ID information and condition the incoming audio for local recording and storage. After the device records the call, the call is sent to the database from the local storage. As the call recording is uploaded and sent to the database, the database will take silent portions of audio and remove them to cut down on the uncompressed WAV file size to improve the accuracy of the machine learning algorithm. After removing the silence, the machine learning algorithm can then use the file to match against known callers and scammers. Suppose the incoming caller's voice signature does match a voice signature from the whitelist or blacklist. In that case, the device will display who the incoming call is from to the end-user using the gateway API developed by NCID. If the incoming call does not match the voice signature of someone on the whitelist or blacklist, then the device will display 'unknown caller' to the end-user to let them know that the incoming caller does not match anyone in the database.

### 2.1. Background

Current scam detection has been optimized for modern communications services. These situations inadvertently leave behind some of the most vulnerable in our society to scams, the elderly. Without proper caller ID or scam detection, many vulnerable individuals are left to their own devices to determine whether someone calling them is attempting to take advantage of them. The NCID Defender system will build upon an existing system for detecting VOIP (Voice Over Internet Protocol) and landline caller data, NCID. We aim to stop the scams before they take root by introducing a method to identify incoming callers through vocal signature matching.

A primary concern for elderly end-users is when a phone line rings, they feel compelled to answer. This response is similar to when a phone sends a notification. The urge to answer every call creates situations where people are more likely to be scammed. Our device will suppress the first ring of the end-users telephone lines until after a basic screening process. And then send a notification to the user when our system has analyzed the incoming caller's vocal signature.

Building off of the current infrastructure of the NCID modem (whitelisting and blacklisting), we plan to implement a machine learning aspect that will adapt to SPIT in the modern world of communication. The machine learning will consider the incoming caller's unique voice signature and match that against a list of known whitelisted and blacklisted scam callers. These voice signatures are as unique as fingerprints. They are used for identification and authentication purposes across many industries, so adapting this to the home landline will allow the end-user to verify whether someone is trying to deceive them, has tried to deceive them, or has scammed other users in the past. This method expands NCID's creation by adding a dynamic blacklist to allow more efficient scam detection and denial.



## **2.2. Overview**

Our system will monitor incoming calls and suppress the first ring to gather caller ID and call information. The device will then interpret whether the caller is 'scam likely' or 'scam unlikely' by checking against incoming callers' white and black lists already implemented by NCID. At the same time, the system will record a portion of the incoming caller's voice and upload that information to the database. The database will remove silent audio portions before sending the audio to the machine learning algorithm, where the voice signature will be analyzed. If the caller's voice signature is on the whitelist, the user will be notified who is calling them, as determined by the machine learning algorithm. If the caller's voice is on the blacklist, the device will notify the end-user that the person they are speaking to is a known scammer and display the accuracy of the match.

Suppose the caller is not identified on either list. In that case, the device will tell the user that the incoming caller's voice signature does not match any known voice signatures, and a flag will be added to the call so that it is easily identified in the database.

## **2.3. Referenced Documents and Standards**

### **2.3.1. Reference Documents**

[1] S. J. Blumberg and J. V. Luke, "Wireless Substitution: Early Release of Estimates from the National Health Interview Survey, January-June 2021," p. 8, Nov. 2021.

### **2.3.2. Standards**

[2] "AT Commands Reference Guide." Telit wireless solutions, Apr. 08, 2006.

[3] "Dual Tone Multi frequency (DTMF) Configuration." Media5, Feb. 02, 2022.

[4] "IEEE GET Program, GET 802 Standards." IEEE.

[5] Matthiesen, Wickert, and Lehrer, "LAWS ON RECORDING CONVERSATIONS IN ALL 50 STATES." MATTHIESEN, WICKERT & LEHRER, SC, Jan. 13, 2022.

## **3. Operating Concept**

### **3.1. Scope**

The scope of the project, The NCID Defender, is interpreting caller identification and information, analyzing this information against a database of scam calls, and then analyzing a portion of the incoming call for a voice signature that can be matched against the database. After the system has analyzed the voice of the incoming caller, a receipt of the call length, time, date, and other relevant information will be logged in the database, as well as a flag as to whether the incoming caller was a known 'good,' known 'bad,' or 'unknown' voice signature. If the incoming caller matches a known 'bad' voice signature, someone close to the victim will be notified about the call. This device will be an open-source product that individuals can build independently. The device may vary in end cost, size, and processing ability. Our code and device shall be designed to be compatible with consumer-level development kit boards such as the Raspberry Pi, TI LaunchPad, and Espressif-32.

### **3.2. Operational Description and Constraints**

The NCID Defender will filter calls using the existing NCID software and be improved with machine learning to filter calls by analyzing voice signatures and alerting the end-user when a voiceprint matches a flagged blacklisted signature. Known callers on the blacklist will follow a blacklist protocol where the call will be logged, and someone close to the victim will be notified. Likewise, whitelisted callers will be allowed to call through and display a message to the end-user with a message stating whom the voice signature matches. Lastly, for unknown callers, it will record the audio and flag the call in the database so that it is easily located in case of suspicious activity.

This project's constraints lie in maintaining our code's compatibility across several MCU platforms. Due to this project being an open-source project, the end-user may decide to implement this system on any consumer-level development kit board. We will need to maintain code compatibility to a reasonable degree across many products. Additionally, the project's audio analysis portion heavily relies on the WAV format since it contains a broader frequency range. At the time of writing, we have yet to test on other formats to see if the analysis is still possible; this is something we plan to explore in the future.

### 3.3. System Description

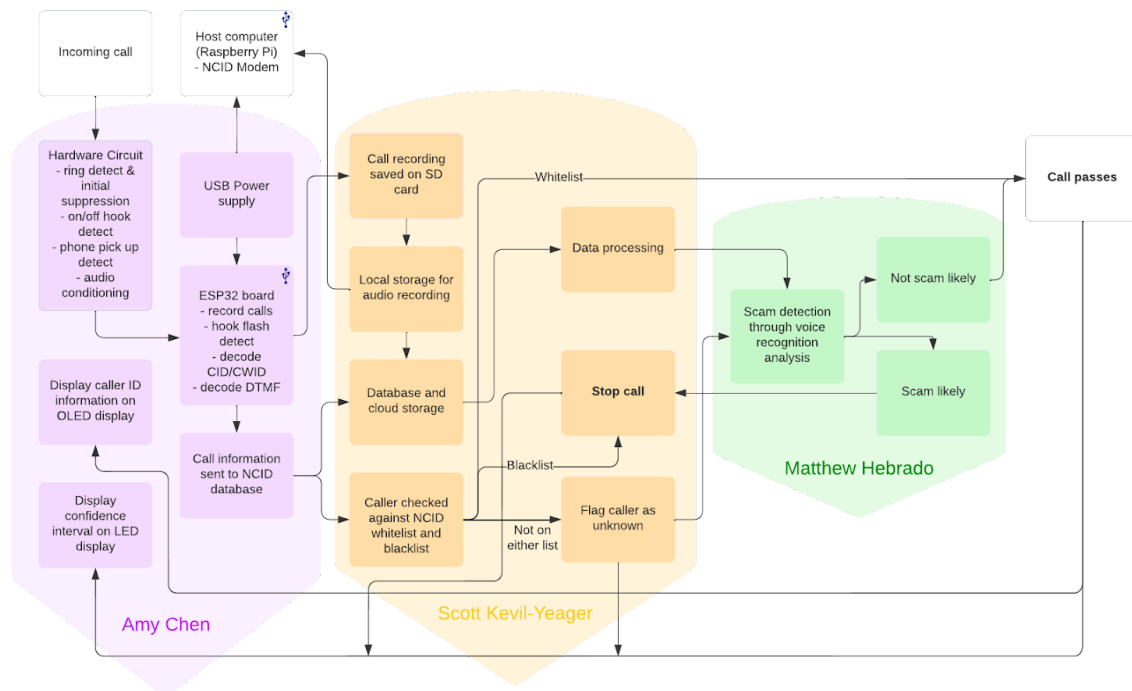


Figure 1. System Block Diagram

#### 3.3.1. Machine Learning

For the machine learning aspect, we must segment and classify the incoming caller's vocal signature and then compare it to a list of known vocal signatures before determining if there is a match.

#### 3.3.2. Hardware for Interface

The hardware circuit will detect the ring, on and off-hook, and phone pick-up. It will also suppress the initial ring and disconnect the downstream phones momentarily. Furthermore, it will take the incoming analog signal and condition the audio to feed into the microcontroller.

#### 3.3.3. Firmware for Interface

The microcontroller will convert the analog signal to a digital signal and decode caller ID and caller waiting ID. After receiving the digital signal, the microcontroller will start recording the call and storing the recording on an SD card to be sent to the local database and the machine learning algorithm. The microcontroller will also detect hook flash occurrence.

### **3.3.4. Database**

The database will hold full-length audio recordings for a family member or end user to review calls that may be flagged as suspicious. It will also allow the machine learning to store the matched callers' confidence values, the call length, and the date the call was recorded. The database will also implement creation, read, update, and delete functions so that any future users can maintain their database.

## **3.4. Modes of Operations**

### **3.4.1. Blacklist**

When the call is received, and the vocal signature matches with a known blocked caller, our device can play an automated message to disconnect the call or fax tone should the person implementing our device decide to do so. It will then flag the incoming call should it be answered and send a notification to someone close to the victim.

### **3.4.2. Whitelist**

When the call is received and matches with a vocal signature matching with someone in the database, the end-user will receive a notification across their NCID modem, telling the user who is calling as determined by the machine learning algorithm.

### **3.4.3. Unknown**

When a caller does not match any voice signatures on the white or black list, the end-user will be notified that the incoming caller is 'unknown.' There will be a process by which the end-user can add the caller to a white or black list. If the user decides not to do that, the call will be flagged as an incoming call from an unknown user in the database.

## **3.5. Users**

The primary user of this device will be the older generation who still use landlines. We will need the user to have a computer or laptop, an internet connection, and a landline phone for this device. The end user's technical ability is expected to be limited. However, this system is meant as an open-source project that someone close to the end-user will have the technical ability to implement.

## **3.6. Support**

The end-user can access the SourceForge software documentation and forums to ask questions. Code documentation can be found within the files for the open-source NCID software.

## **4. Scenario(s)**

### **4.1. *Blacklist Setting***

- Grandparents receive a phone call from an established scam number.
- The number gets fed into The NCID Defender.
- NCID analyzes the incoming caller's information.
  - Finds the number in the blacklist.
- Machine learning matches voice signature to a known scammer's voice signature.
- The NCID Defender flags the call and alerts someone close to the victim and notifies the victim during the call that the person has matched the voice signature of a known scammer.

### **4.2. *Whitelist Setting***

- Grandparents receive a phone call from a known caller.
- The number gets fed into The NCID Defender.
- NCID analyzes the incoming caller's information.
  - Finds the number in the whitelist.
- Machine learning matches the signature to a known whitelisted voice signature.
- The NCID Defender passes the call through and notifies the user as to which voice signature the incoming caller matched (i.e., Grandson, Daughter, Friend from work, these identifications will be chosen by the end-user and will include their name).

### **4.3. *Unknown Setting***

- Grandparents receive a phone call from an unknown number.
- The number gets fed into The NCID Defender.
- NCID analyzes the incoming caller's information.
  - Does not find the number in the whitelist or blacklist.
- Machine learning does not match the voice signature to any known voice signatures.
- Notifies the user that the incoming caller is 'unknown', and then flags the recording of the call as an unknown caller in the database for easy identification.

## **5. Analysis**

### ***5.1. Summary of Proposed Improvements***

This device will add the ability for calls to be tagged as known 'good,' known 'bad,' and 'unknown.' The device will also be able to record, screen, and analyze the vocal signatures of incoming callers. After analyzing the vocal signature, the device will be able to notify the user across all NCID integrated devices whether the call was known 'good,' known 'bad,' or 'unknown.' The device will then alert someone if the incoming caller was known 'bad' and allow action to be taken quickly before a scam can become deeply rooted.

### ***5.2. Disadvantages and Limitations***

The most significant disadvantage will be the end-users technical abilities. Another disadvantage we must consider when designing this device is that the end-user may have outdated computing capabilities within their home. Our method to combat this disadvantage is to keep as much information as possible on the development kit board. We will have data on suspicious calls sent by email or similar communications technology so that the end-user does not have to store that data on their devices. We will also implement data management to reduce on the data stored on the device and ensure that old or whitelisted calls are prioritized as the first to be removed if space is needed.

### ***5.3. Alternatives***

An alternative is the current NCID software. However, the NCID software only allows blocking callers, whitelisting, and displaying caller ID. This device can also be implemented across various development boards or kits, as discussed throughout the CONOPS. Another alternative is a corporate voice signature matching program. It is possible to create one independently. However, it is very time-consuming

### ***5.4. Impact***

Our project does not produce waste or affect the ecology of any region. The only environmental impact of our product will be the energy cost of the materials required to build it. We will also be running the computers for the ML algorithm. Societally, our project aims to reduce the number of successful scam attempts.

The NCID Defender  
Matthew Hebrado  
Amy Chen  
Scott Kevil-Yeager

# **INTERFACE CONTROL DOCUMENT**

REVISION 3  
2 December 2022

# INTERFACE CONTROL DOCUMENT FOR The NCID Defender

TEAM <28>

PREPARED BY:

---

Author Date

APPROVED BY:

---

Project Leader Date

---

Dr. Wonhyeok Jang Date

---

Rohith Kumar Date



## Change Record

Rev.	Date	Originator	Approvals	Description
0	2/23/2022	Matthew Hebrado Amy Chen Scott Kevil-Yeager		Midterm Report Submission
1	3/2/2022	Matthew Hebrado Amy Chen Scott Kevil-Yeager		Revision 1: changes after grading
2	4/30/2022	Matthew Hebrado Amy Chen Scott Kevil-Yeager		Revision 2: changes for Final Report 403 Submission
3	12/2/2022	Matthew Hebrado Amy Chen Scott Kevil-Yeager		Revision 3: changes for Final Report 404 Submission

## Table of Contents

<b>Table of Contents .....</b>	<b>III</b>
<b>List of Tables .....</b>	<b>IV</b>
<b>No table of figures entries found.....</b>	<b>Error! Bookmark not defined.</b>
<b>List of Figures.....</b>	<b>V</b>
<b>No table of figures entries found.....</b>	<b>Error! Bookmark not defined.</b>
<b>1. Overview.....</b>	<b>1</b>
<b>2. References and Definitions.....</b>	<b>2</b>
2.1. References.....	2
2.2. Definitions .....	2
<b>3. Physical Interface .....</b>	<b>3</b>
3.1. Weight.....	3
3.2. Dimensions .....	3
<b>4. Thermal Interface .....</b>	<b>4</b>
<b>5. Electrical Interface .....</b>	<b>5</b>
5.1. Primary Input Power.....	5
5.2. Signal Interfaces .....	5
5.3. Display Interfaces .....	5
5.4. User Control Interface .....	5
<b>6. Communications / Device Interface Protocols .....</b>	<b>6</b>
6.1. Wireless Communications (WiFi) .....	6
6.2. Host Device.....	6
6.3. Device Peripheral Interface .....	6

## **List of Tables**

No table of figures entries found.

## **List of Figures**

No table of figures entries found.

## **1. Overview**

The Interface Control Document (ICD) for the Smart Caller ID for Landlines will provide more information about how the subsystems mentioned in the Concept of Operations and the Functional System Requirements (FSR) will be produced. This report will include physical descriptions of the various elements and the device's electrical, thermal, and device interface. The document will also explain how the subsystems will interface together to achieve the goals and requirements mentioned in the FSR and CONOPS documents.

## **2. References and Definitions**

### **2.1. References**

**Law on Recording Conversations In All 50 States**

5 Jan 2021

**WLAN Standards**

October/November 2003

### **2.2. Definitions**

CAT3	Category 3
RJ11	Registered Jack 11
LED	Light Emitting Diode
MCU	Micro-Controller Unit
ML	Machine Learning
SD	Secure Digital

### **3. Physical Interface**

#### **3.1. *Weight***

The weight of this device will be 3 pounds.

#### **3.2. *Dimensions***

The dimensions of this device will be 7 ½ inches in length by 5 ½ inches in height by 2 inches in width to accommodate for the hardware board, the ESP32, and any consumer-level development board or kit.

## **4. Thermal Interface**

This project will have minimal concerns about heat management. The ESP32 series of MCU can operate within -40 to 125 degrees Celsius (-40 to 257 degrees Fahrenheit). We will use a passive heatsink to mitigate high temperatures should the device be put in low ventilation or high-temperature environments, such as in a cabinet, behind a fridge, or in a window sill.



## **5. Electrical Interface**

### ***5.1. Primary Input Power***

The primary input for power will come from a wall socket. The input voltage will need to be adjusted to 3.3 V DC with a current of 500 mA for the ESP32 MCU.

### ***5.2. Signal Interfaces***

The primary signal interface will be a RJ11 or analog landline signal. This signal will be converted to a digital signal and sent to the MCU to be processed.

Another signal interface for this device will be the MCU to the SD card for the longer-term storage of voice files and call logs. This SD card will only be a temporary buffer that will eventually be uploaded to a database for permanent storage when the phone is not in use.

### ***5.3. Display Interfaces***

This device will display the incoming call information and the ML learning algorithm's interpretation of whether the call is likely to be a scam call or not. The display will also include an LED that will show green for whitelisted or known caller, orange for an unknown caller, and red for known scam or spam calls. This process will allow for easier identification across rooms or large spaces.

### ***5.4. User Control Interface***

Due to this device's target audience, this device will have a minimal user control interface in order to mitigate potential confusion or misuse of the device.

## **6. Communications / Device Interface Protocols**

### **6.1. *Wireless Communications (WiFi)***

IEEE 802.11ac WiFi protocols will be used for this device.

### **6.2. *Host Device***

The host device would be a local computer on the network that the NCID software is running on. This host will communicate with the ESP32 on our device and our database for recording spam calls. The ESP32 will record and decode caller ID, storing the info on a local storage device or SD card. The host machine will collect the call into the database from the local storage or SD card. The machine learning will be on the host machine and will gather the data from the database without any additional interface required.

The ESP32 will connect to the host device via USB.

### **6.3. *Device Peripheral Interface***

An SD card reader will be used to store data locally.

The NCID Defender  
Matthew Hebrado  
Amy Chen  
Scott Kevil-Yeager

# **FUNCTIONAL SYSTEM REQUIREMENTS**

REVISION 3  
2 December 2022

# FUNCTIONAL SYSTEM REQUIREMENTS FOR The NCID Defender

TEAM <28>

PREPARED BY:

---

Author Date

APPROVED BY:

---

Project Leader Date

---

Dr. Wonhyeok Jang Date

---

Rohith Kumar Date

## Change Record

Rev.	Date	Originator	Approvals	Description
0	2/23/2022	Matthew Hebrado Amy Chen Scott Kevil-Yeager		Midterm Report Submission
1	3/2/2022	Matthew Hebrado Amy Chen Scott Kevil-Yeager		Revision 1: changes after grading
2	4/30/2022	Matthew Hebrado Amy Chen Scott Kevil-Yeager		Revision 2: changes for Final Report 403 Submission
3	12/2/2022	Matthew Hebrado Amy Chen Scott Kevil-Yeager		Revision 3: changes for Final Report 404 Submission

## Table of Contents

<b>Table of Contents .....</b>	<b>III</b>
<b>List of Tables .....</b>	<b>IV</b>
<b>No table of figures entries found.....</b>	<b>IV</b>
<b>List of Figures.....</b>	<b>V</b>
<b>1. Introduction.....</b>	<b>1</b>
1.1. Purpose and Scope .....	1
1.2. Responsibility and Change Authority .....	1
<b>2. Applicable and Reference Documents.....</b>	<b>2</b>
2.1. Applicable Documents.....	2
2.2. Reference Documents.....	2
2.3. Order of Precedence .....	2
<b>3. Requirements.....</b>	<b>3</b>
3.1. System Definition .....	4
3.2. Characteristics.....	5
3.2.1. Functional / Performance Requirements .....	5
3.2.1.1. Requirement #1 .....	5
3.2.1.2. Search Probability of Detection .....	5
3.2.1.3. Operational Search Altitude.....	5
3.2.2. Physical Characteristics .....	5
3.2.2.1. Volume Envelope .....	5
3.2.3. Electrical Characteristics .....	5
3.2.3.1. Inputs .....	5
3.2.3.1.1 Power Consumption .....	5
3.2.3.1.2 Input Voltage Level .....	6
3.2.3.1.3 Input Noise and Ripple .....	6
3.2.3.2. Outputs.....	6
3.2.3.2.1 Data Output.....	6
3.2.4. Environmental Requirements .....	6
3.2.4.1. Thermal .....	6
3.2.5. Failure Propagation .....	6
3.2.5.1. Failure Detection, Isolation, and Recovery (FDIR).....	6
3.2.5.1.1 Recent Call Logs.....	6
<b>4. Support Requirements .....</b>	<b>7</b>
<b>Appendix A: Acronyms and Abbreviations .....</b>	<b>8</b>

## **List of Tables**

No table of figures entries found.

## List of Figures

<b>Figure 1. The NCID Defender Conceptual Image.....</b>	<b>1</b>
<b>Figure 2. Block Diagram of System.....</b>	<b>4</b>



# 1. Introduction

## 1.1. Purpose and Scope

The Functional System Requirements (FSR) document will cover this project's system requirements in detail. The NCID Defender is an efficient way to filter out spam callers. Our device will be most beneficial to older people who are more vulnerable to fraud. The device will have a screening process to block out spam callers and let non-spam callers go through. Figure 1 shows a representative integration of the project in the proposed CONOPS.

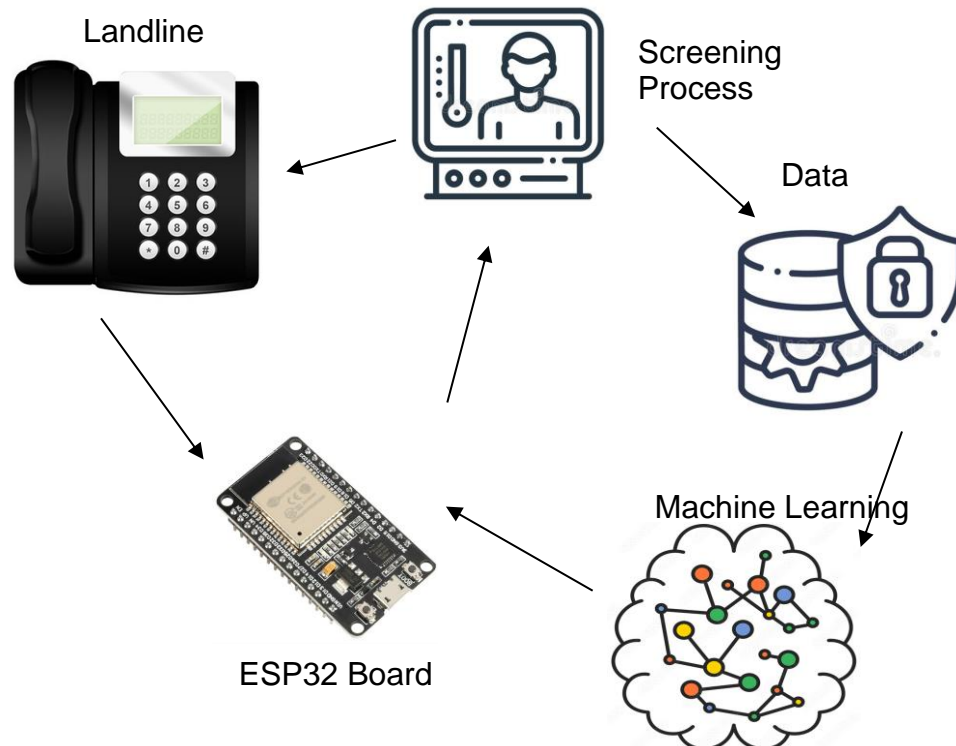


Figure 1. The NCID Defender Conceptual Image

## 1.2. Responsibility and Change Authority

The team leader is Scott; the client and the whole team can make changes to the project as necessary. Any changes that need to be addressed will be discussed by all members and relayed to the sponsor.

## **2. Applicable and Reference Documents**

### **2.1. Applicable Documents**

The following document(s), of the exact issue and revision shown, form a part of this specification to the extent specified herein:

<b>Document Number</b>	<b>Revision/Release Date</b>	<b>Document Title</b>
IEEE 802.11	October/November 2003	WLAN Standards

### **2.2. Reference Documents**

The following documents are reference documents utilized in the development of this specification. These documents do not form a part of this specification and are not controlled by their reference herein:

<b>Document Number</b>	<b>Revision/Release Date</b>	<b>Document Title</b>
IEEE 802.11	October/November 2003	WLAN Standards

### **2.3. Order of Precedence**

In the event of a conflict between the text of this specification and an applicable document cited herein, the text of this specification takes precedence without any exceptions.

All specifications, standards, exhibits, drawings or other documents that are invoked as “applicable” in this specification are incorporated as cited. All documents that are referred to within an applicable report are considered to be for guidance and information only, except ICDs that have their relevant documents considered to be incorporated as cited.

### 3. Requirements

This section defines the minimum requirements that the development item(s) must meet. The requirements and constraints that apply to performance, design, interoperability, reliability, etc., of the system, are covered.

The NCID Defender device requirements are as follows. The device must interface with a POTS using the standard North American caller ID protocols and detect the ringing of the landline. It must then suppress the first ring of a call until the call can be analyzed. The device must detect and report on-hook and off-hook handset states and hook-flash events where users can switch between the primary and waiting calls.

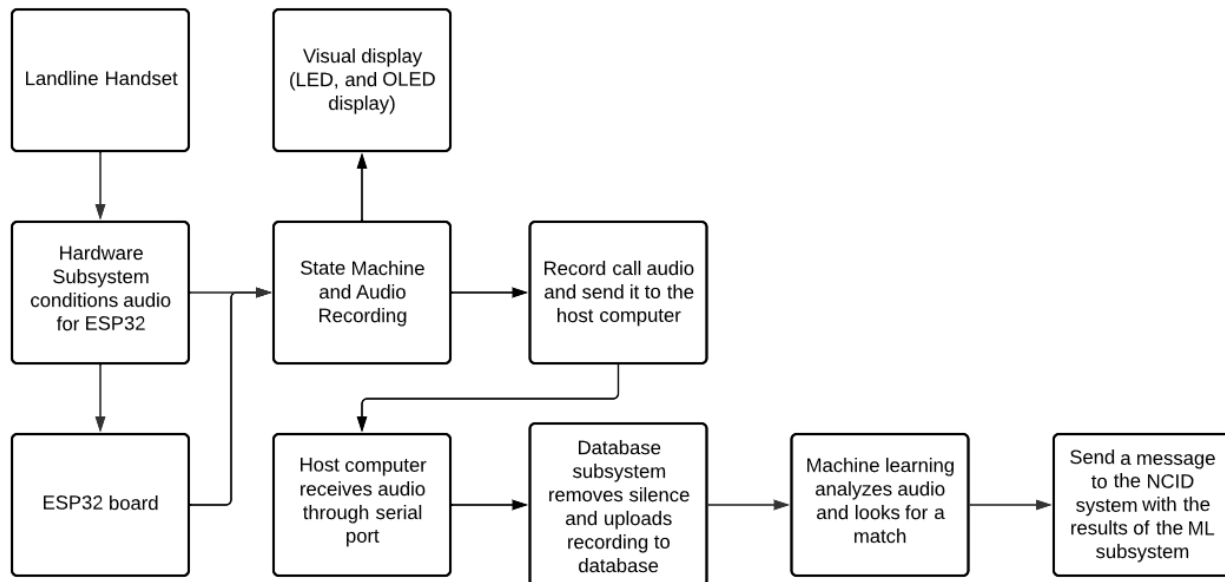
The device must then record the beginning and end times of calls and an audio recording of the calls. When a call is being made to the end-user, the device must decode the caller ID of the incoming call. The device must also detect incoming numbers from a blacklist and provide an automated message or disconnected line tone.

From the device recorded call audio, the device must be able to identify incoming callers by voice signature to prevent 'social engineering' scams where scammers call and gather information from an unsuspecting victim before calling back equipped with that personal information.

This device will have at least a 75% scam detection rate, and the code will be able to operate on consumer-level development kits such as Raspberry Pi, TI LaunchPad, and Espressif-32.

### 3.1. System Definition

The project will take analog signal inputs, decode the caller ID, check that caller ID against a database of known scam callers. The device will then record audio of the call, call time, call length, and number of times this caller ID has called the end-user. Then, the information will be sent to a database. A machine learning algorithm will incorporate that data into its scam detection and periodically update the end user's device. This process will ensure that newly detected scammers are caught in the future.



**Figure 2. Block Diagram of System**

The analog landline signal will be received and processed for the caller ID information. This information will be sent to the microcontroller, where the NCID software will check the number against a whitelist and blacklist known numbers. If this number is not on either list, then the call will be sent to a simple screening process that will be modifiable by the end-user. An example might be, "Who is calling?" or "Whom are you calling for?." After answering these questions, the vocal signature of the response will be analyzed against previous calls. If the vocal signature matches a known whitelisted voice signature, the device will display a green LED and indicate that the caller is known. If it matches the voice of a known scammer, then the device will display a red LED.

The end-user will be able to play an automated message or sound if a call is detected as spam. If the end-user accepts the call, a note of ML algorithms analysis of whether the call was spam or not will be recorded in the call log. After the call ends, the call log will be sent by WiFi to the ML algorithm, and the audio recording of the call will be kept on the device's SD card until space is needed to store another call. The vocal signature ML algorithm would analyze the vocal signature of the call and update the vocal signatures if it was a known spammer or a whitelisted caller calling from another number.

## **3.2. Characteristics**

### **3.2.1. Functional / Performance Requirements**

#### **3.2.1.1. Requirement #1**

The NCID Defender project's code will be able to run on consumer level development kit boards from TI, Espressif, and Raspberry Pi.

*Rationale: This device must be easy to assemble and accessible to the end user. These consumer boards will have the same functionality as the device we are developing and will meet minimum spec requirements for running the code.*

#### **3.2.1.2. Search Probability of Detection**

The NCID Defender device will be able to detect 75% of incoming SPIT calls and take the appropriate measures to ensure that the SPIT calls do not reach the end-user.

*Rationale: Although this number can be improved, we may be constrained by the amount of data that we will have to train this device.*

#### **3.2.1.3. Operational Search Altitude**

The NCID Defender device will support voice signature matching to cut down on false positive and negative results. This feature will allow the device to identify a 'whitelisted' user even when they call from a different caller ID.

*Rationale: Many older people have difficulty determining the voice of who is calling. Unfortunately, this leads to frequent scamming through 'social engineering' personal information about their family before calling again equipped with that information. If we can have the voice signature of family members, then we can determine, with much greater accuracy than the end-user, whether the person who is calling is a relative or a scammer calling from a different caller ID.*

### **3.2.2. Physical Characteristics**

#### **3.2.2.1. Volume Envelope**

The volume envelope of the smart caller ID device shall be less than or equal to 2 inches in height, 6 inches in width, and 6 inches in length.

*Rationale: This size envelope is specified by the potential range of dimensions of consumer boards.*

### **3.2.3. Electrical Characteristics**

#### **3.2.3.1. Inputs**

Analog landline telephone signal or digital VoIP signal will be used as an input signal. The analog signals will have FSK modulated information similar to FM radio; the VoIP signaling protocol is based on H.323 and Media Gateway Control Protocol (MGCP).

##### **3.2.3.1.1 Power Consumption**

The maximum peak power of the system shall not exceed 8 watts

*Rationale: These are the high-end range of consumer development kits or boards that will be used when assembling this project as an open-source DIY project, with additional power consumption accounted for the display and indicator LED's.*

### **3.2.3.1.2 Input Voltage Level**

The input voltage level for The NCID Defender device shall be +1.8 VDC to +3.6 VDC.

*Rationale: This is the range of operation for the ESP32 SoC that the team has selected for the project.*

### **3.2.3.1.3 Input Noise and Ripple**

The NCID Defender device will be able to process, record, and playback analog signals with minimal noise interference.

### **3.2.3.2. Outputs**

#### **3.2.3.2.1 Data Output**

The NCID Defender device's voice signature data will be locally stored until there is appropriate downtime to upload any anonymized data to the database, all other easily transmittable data (call logs, date of call, length of call, etc) will be sent to the database as soon as the call has ended using standard Wifi protocols.

*Rationale: This is to ensure that we are not throttling the end-users wireless data connectivity after every phone call as we do not know the end-users wireless data speeds.*

### **3.2.4. Environmental Requirements**

#### **3.2.4.1. Thermal**

The only environmental requirements of this system are that it can operate in low airflow or relatively high heat environments. Examples of these constraints might be the device being in a cabinet or on a windowsill where it may receive natural light for long hours.

### **3.2.5. Failure Propagation**

The NCID Defender device will use the onboard watchdog timers to prevent errors that would stop the operation of the device.

#### **3.2.5.1. Failure Detection, Isolation, and Recovery (FDIR)**

##### **3.2.5.1.1 Recent Call Logs**

Recent calls will be locally stored until space is needed for additional calls. This process will allow for recordings that may identify the extent of a scam to be reviewed should an attempted scam take place.

*Rationale: People who are scammed can often feel ashamed and attempt to hide the extent of a scam, furthering the damage done financially.*

## **4. Support Requirements**

The NCID Defender device will require a consumer development kit or board such as a TI LaunchPad, a Raspberry Pi, or an Espressif-32. The person who installs this device will need to be knowledgeable enough with their chosen device to compile and upload the code to the board and make all signal connections based on the README file provided in the SourceForge files.

The main avenue of technical support will be the SourceForge forum, where the open-source software is uploaded. There will be no warranty for the device.

## **Appendix A: Acronyms and Abbreviations**

FM	Frequency Modulating
FSK	Frequency Shift Keying
LED	Light-Emitting Diode
MGCP	Media Gateway Control Protocol
ML	Machine Learning
POTS	Plain Old Telephone Service
SPIT	Spam Over Internet Telephony
VoIP	Voice Over Internet Protocol



The NCID Defender  
Matthew Hebrado  
Amy Chen  
Scott Kevil-Yeager

## **EXECUTION AND VALIDATION PLAN**

REVISION 1  
2 December 2022

## 1. Execution Plan

[illegible]

## 2. Validation Plan

Test	Detail	Data	Status	Responsible Student
Device powers on	Turns on Raspberry Pi and ESP32	Turns on	Complete	Amy Chen
Display powers on	Displays caller ID information		Complete	Amy Chen
Ring detect	LED1 lights up when detection occurs	48 V DC to sine wave	Complete	Amy Chen
Ring suppress	Initial ring is suppressed		Complete	Amy Chen
Phone pick up	LED2 lights up	LED lights up when CAL* is grounded	Complete	Amy Chen
Audio Conditioning Out	Phone audio to ADC1 pin		Complete	Amy Chen
Audio Conditioning In	DAC1 pin to audio		Complete	Amy Chen
Detect off-hook/on-hook	LED3 lights up when detection occurs		Complete	Amy Chen
Detect hook flash on ESP32	Detect hook flash in firmware	Code written, not tested	Incomplete	Amy Chen
Decode CID/CWID on ESP32	Decode CID/CWID information in firmware	Code written, not tested	Incomplete	Amy Chen
Decode DTMF and FSK on ESP32	Decode DTMF and FSK in firmware	Code written, not tested	Incomplete	Amy Chen
OLED program	Code for OLED display		Complete	Amy Chen
WS2812B program	Code for LED light		Complete	Amy Chen
Control WS2812B	Test code on LED light		Complete	Amy Chen
Retrieve file from database	The file will be in the given or created directory that the user has input		Complete	Scott Kevil-Yeager
UI works as expected, allowing users to input test folder directories	UI works as expected, allowing users to input test folder directories		Complete	Scott Kevil-Yeager
Upload folder	Files in given directory will be counted, processed, named, and uploaded to the database automatically		Complete	Scott Kevil-Yeager
Listen to recording	Properly allows the playback of recording audio through the host machine, this assumes that the host machine will have a speaker		Complete	Scott Kevil-Yeager
Error checking	If a folder directory or file directory is incorrectly given then a message is given and the user is prompted for another input		Complete	Scott Kevil-Yeager
Delete recording in database	Given a valid name the function removes a single entry from the database		Complete	Scott Kevil-Yeager
Delete local recording	If a folder path and file name are given then the function will delete the local file		Complete	Scott Kevil-Yeager
pyAudioAnalysis	Removes periods of silence in recordings to reduce file size		Complete	Scott Kevil-Yeager
Local storage receives recordings	A file can be written to the SD card		Complete	Scott Kevil-Yeager
ESP32 Captures incoming analog audio signal	Samples audio signal using the ESP32 ADC1		Complete	Scott Kevil-Yeager
Handset properly records through ESP32	Audio is sampled and then written to the SD card as a .raw format file		Complete	Scott Kevil-Yeager
Integrate with ML subsystem	Audio files with silence removed are written to filesToTest, and the database is accessible by the ML code		Complete	Scott Kevil-Yeager
Integrate with hardware subsystem	Audio files are recorded when the phone goes off hook, and data is sent to the serial port when the phone is put back on hook		Complete	Scott Kevil-Yeager
Integrate with NCID	Connect using TCPIP to the NCID platform and send messages to it		Incomplete	Scott Kevil-Yeager
Serial communication between microcontroller and database	Receive serial port data in python and write it to a file with a wav file header		Incomplete	Scott Kevil-Yeager
generate files used to train classification	take a source file and split it into 0.5 sec intervals	several wav files are produced	Complete	Matthew Hebrado
graph feature comparisons	based on the feature extraction graph is generated that displays a comparison of the two speakers (shown throughout ML documentation)	a graph that compares 2 speaker data is presented	Complete	Matthew Hebrado
create classification file	does feature extraction on all files in a directory and creates classification file		Complete	Matthew Hebrado
run tests/predictions from known speakers	More information on Section 3 of ML documentation	Returns highest confidence interval	Complete	Matthew Hebrado
run tests/predictions from unknown speakers	More information on Section 3 of ML documentation	Returns highest confidence interval	Complete	Matthew Hebrado
code runs on pi	Output is similar to Figures 3.2-4 in ML documentation	classification is generated and output is printed	Complete	Matthew Hebrado
send file to database	function exists; needs to be implemented		Complete	Matthew Hebrado
receive file from database	function exists; needs to be implemented		Complete	Matthew Hebrado
items on trained files produce at least 75% accuracy	More information on Section 3 of ML documentation		Not Working	Matthew Hebrado

# The NCID Defender

Matthew Hebrado

## **MACHINE LEARNING SUBSYSTEM**

REVISION 1  
2 December 2022

# MACHINE LEARNING SUBSYSTEM DOCUMENT FOR The NCID Defender

TEAM <28>

PREPARED BY:

---

Matthew Hebrado Date

APPROVED BY:

---

Project Leader Date

---

Dr. Wonhyeok Jang Date

---

Rohith Kumar Date

## Change Record

Rev.	Date	Originator	Approvals	Description
0	4/30/2022	Matthew Hebrado		Final Report 403 Submission
1	12/2/2022	Matthew Hebrado		Revision 1: changes for Final Report 404 Submission

## Table of Contents

<b>Table of Contents .....</b>	<b>III</b>
<b>List of Tables .....</b>	<b>IV</b>
<b>List of Figures .....</b>	<b>V</b>
<b>1. Introduction .....</b>	<b>1</b>
<b>2. Machine Learning Subsystem Report .....</b>	<b>2</b>
2.1. Subsystem Introduction .....	2
2.2. Feature Extraction .....	2
2.3. Subsystem Details/Programs Used – Testing Phase .....	3
2.3.1. testingAudioReading.py .....	4
2.3.2. testingDirectoryFE.py .....	5
2.3.3. testingTraining.py / DemoTrainingGraphs.py (old) .....	6
2.3.4. testingWrapping.py / DemoSVMFeatureWrapping.py (old) .....	6
2.3.5. testingClassification.py / DemoClassification.py (old) .....	6
2.4. Subsystem Details/Programs Used – Demo Phase .....	7
2.4.1. convertToWav.py .....	7
2.4.2. demoAudioSplit.py .....	7
2.4.3. demoFeatureWrapping.py .....	7
2.4.4. demoClassification.py / demoSingleClassification.py .....	7
<b>3. Conclusion .....</b>	<b>8</b>
<b>4. Validation .....</b>	<b>9</b>

## List of Tables

<b>Table 1. Validation and Test for Section 2.4.4. ....</b>	<b>9</b>
---	----------



## List of Figures

Figure 1. Time vs. Frame Energy.....	4
Figure 2. Spectral Centroid and Spectral Entropy on 2 Speakers (1).....	5
Figure 3. Spectral Centroid and Spectral Entropy on 2 Speakers (2).....	6
Figure 4. Testing An Unknown Voice.....	9
Figure 5. Testing Handset Recording Against All Voices on White List .....	10
Figure 6. Integration with Database/Data Processing Subsystem.....	10
Figure 7. Amy vs. Matthew.....	11
Figure 8. Amy vs. Scott .....	11
Figure 9. Matthew vs. Scott.....	12

## **1. Introduction**

The need for machine learning is to create voice prints of known good and known bad actors. When a call comes through to our user's landline phone, our device will process the call and send a message prompting the caller to provide their name and reason for calling. I should note that their answer here is irrelevant to the machine learning algorithm, it is simply used as a way of getting an audio clip of the caller. This clip is then fed to the machine learning algorithm and it will attempt to match it with the classifications that were made through a support vector machine (SVM) and provide a final result.

## **2. Machine Learning Subsystem Report**

This section is documentation of every function made during this project. Each function has an explanation as to what it does.

### ***2.1. Subsystem Introduction***

The machine learning subsystem: known good actors will be uploaded to the database from the user in the form of a WAV file. This file will get processed by several programs and classified by a gradient boost algorithm. The functions used to do the audio processing and classification were provided by an open source library by the name of pyAudioAnalysis by Theodoros Giannakopoulos. The existence of this library heavily influenced the decision in writing these programs in python. Additionally, a dataset of world speakers from Kaggle was used. This dataset included about 1500 one-second WAV files for each (five) speakers. This data set was used for initial testing and data acquisition and then later we included our own voices to do more testing on a smaller dataset.

### ***2.2. Feature Extraction***

The feature extraction that is used for the gradient boost is created from functions written in pyAudioAnalysis. For the context of audio feature extraction several short-term features must be taken from the audio sample. The way that pyAudioAnalysis deals with these short-term features is denoting step sizes and window sizes. For creating the gradient boost classification file, our short term step was and window size are equal to prevent overlapping segments and data to be processed multiple times which would increase runtime. When it comes to graphing, the short term step was less than the window size which meant that there would be overlapping segments. This is more useful for graphing since the gradient boost will have more training material to work with.

For each frame/window in a given audio segment, a set of short term features are extracted from it. Some of these features include, spectral centroid, spectral energy, and zero crossing rate just to name a few. pyAudioAnalysis is able to extract these features in python for use in training the model.

### **2.3. Subsystem Details/Programs Used – Testing Phase**

*\* The testing phase used SVM classification. More explanation about the switch is provided in section 2.4.*

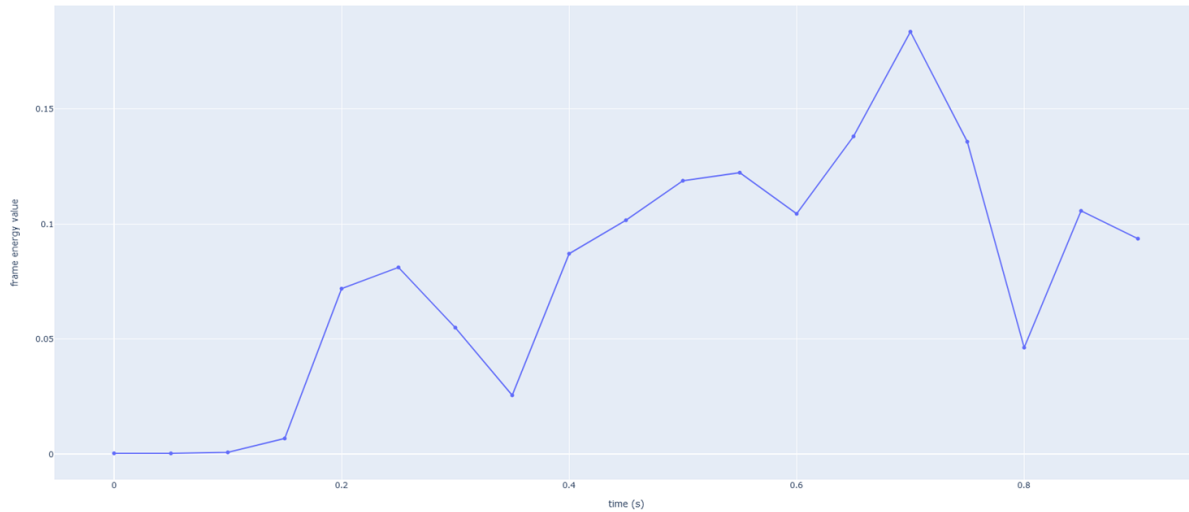
All python files prefixed with “testing” are using the dataset from Kaggle as the inputs and all the python files prefixed with “Demo” were made for demo for our own dataset of three speakers which had about (300 one-second WAV files each) with one exception. During testing, one of our member’s test data was receiving a considerably lower accuracy percentage when compared to the other two and we solved this issue by giving the SVM a longer file to help classify the features better. The issue with the original sample is that there were relatively frequent one second pauses that when the sample would be segmented it did not know how to categorize these leading to a lower prediction percentage when comparing test data to the SVM.

The dataset from Kaggle to was split into 1399 files to train the SVM and 101 to test. Our own dataset was split in 159 files to test for myself and Amy. As mentioned earlier Scott’s prediction values were low so a larger number of files to test were provided for his section totaling to 383 files to train. Each speaker from our own data set had 60 files to test. Each file consists of a one-second segment which was created using an audio splitting program written in python that I wrote.

A quick summary of that program is as follows: a directory, which can contain any number of source audio files, is input and the program marks these as source files. It then splits these source files into one second segments and saves each segment as a new file with the naming convention of 0.wav - [total duration in seconds - 1].wav. The segment duration is a variable number however I chose it to be one second because of the way pyAudioAnalysis does its feature extraction. For each audio clip only one feature matrix will be generated therefore it would be more efficient to have multiple audio clips in order to have more feature matrices which will strengthen the SVM classification.

### 2.3.1. testingAudioReading.py

This was the first program that I wrote. Its purpose was to do short-term feature extraction on a single audio file, print a list of all the features extracted, and produce a graph of the energy. The pyAudioAnalysis function (feature\_extraction()) returns a feature matrix of 68 x [number of frames] that will be useful for future programs.



**Figure 1. Time vs. Frame Energy**

Displayed in **Figure 1** is a graph of the time vs frame energy. I used this as a test to ensure I would be able to extract features using pyAudioAnalysis and that it would be a good fit for this project.

### 2.3.2. testingDirectoryFE.py

Since pyAudioAnalysis is working properly and can successfully extract features on a single audio file and short term extraction is also working, I moved onto the next step which is mid-term feature extraction and inputting a folder of files rather than an individual file. This is taken care of by another function provided by pyAudioAnalysis called `directory_feature_extraction()` what this does is create a feature matrix of [number of midterm steps] x 138 where 138 is the number of segment features/mid-term features. This program also produces a graph of spectral entropy vs spectral centroid data. In speech analysis the spectral centroid is feature rich. The way it translates to frequencies is higher frequencies it will have a higher spectral centroid, this is important when comparing two voice signatures.

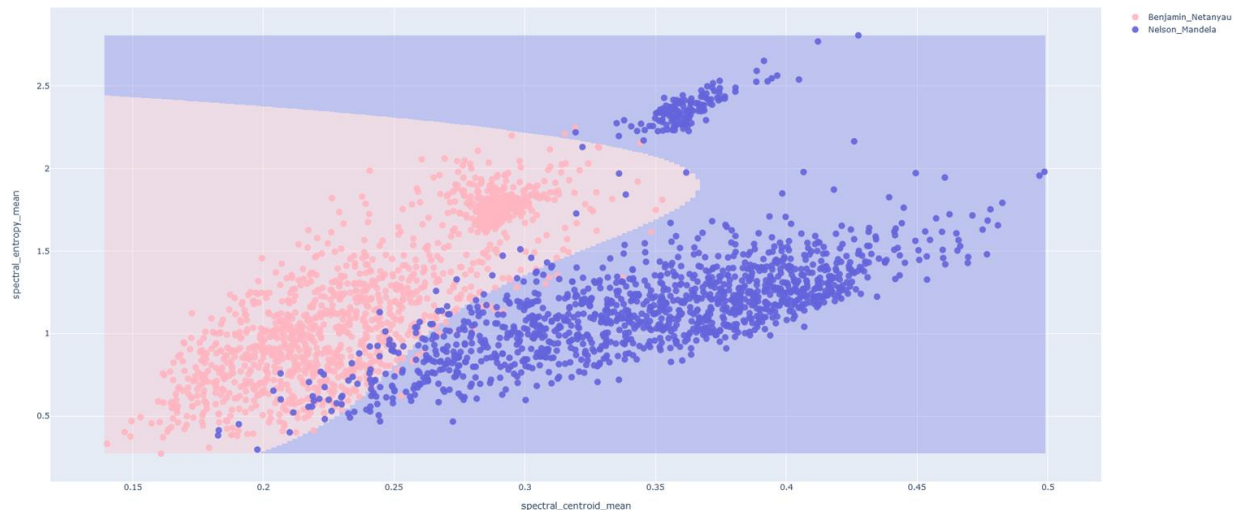


**Figure 2. Spectral Centroid and Spectral Entropy on 2 Speakers (1)**

Displayed in **Figure 2** is the graph of spectral centroid (x) and spectral entropy (y) on two speakers Benjamin Netanyahu (blue) and Nelson Mandela (orange).

### 2.3.3. testingTraining.py / DemoTrainingGraphs.py (old)

These two programs expand on the program described in **section 2.3.2** and add the training of an SVM classifier. The feature extraction remains the same between the two although there is a change to the graphing. It creates and trains an SVM that creates a heatmap around each speaker's data.



**Figure 3. Spectral Centroid and Spectral Entropy on 2 Speakers (2)**

Displayed in **Figure 3** is the graph of spectral centroid (x) and spectral entropy (y) on two speakers Benjamin Netanyahu (orange) and Nelson Mandela (blue). This is only highlighting the comparison of two of the 138 mid-term features.

### 2.3.4. testingWrapping.py / DemoSVMFeatureWrapping.py (old)

The wrapping program wraps all the 138 extracted mid-term features into a single file which I am calling the SVM classifier. The feature extraction works the same as described in previous sections. Once the feature extraction is completed the classifier parameters are set based on the type of machine learning algorithm we want to use. In our case we are sending "svm\_rbf" to pyAudioAnalysis where SVM is support vector machine and rbf is the radial basis function kernel. Lastly a classifier file is trained and created using sklearn in the same directory as the audio files.

### 2.3.5. testingClassification.py / DemoClassification.py (old)

This program runs the test files against the classification file made in **section 2.3.4**. For each test file the program creates a prediction on who it believes is speaking. These test files are subject to the same mid-term feature extraction and compared with the SVM classification file. As of right now the program is about 96% accurate on the data from Kaggle and 90% accurate on our own dataset. This is most likely because the dataset from Kaggle includes much more files to train the SVM.

## **2.4. Subsystem Details/Programs Used – Demo Phase**

During the second half of development when we began integration I learned that some changes needed to be made to the ML aspects to fit better with the new audio files that were coming in. First, let me define the two types of files that my subsystem works with: training files and test files. The training files are the files that go into the whitelist/blacklist and get classified through **section 2.4.3** and create the gradient boost classification file. The test files are the files that are received and get tested against the generated classification file.

In the first half of this project our training and test files were recorded under ideal conditions using a high quality microphone to gather the data. This provided the ML with feature-rich recordings that pyAudioAnalysis (pAA) can extract from. This is important for the generation of the classification file and allows a more accurate reading and prediction of future test files. The training files were rerecorded using iPhones and the testing files were updated to use the files from the handset. In doing so, a significant drop was observed when looking at the prediction values for the given voices added to the whitelist as shown in **Figure 5**.

The reason why the ML classification model was changed off of SVM to gradient boost was an attempt to compensate for the decreased prediction accuracy. More information can be found in the Validation section.

### **2.4.1. converToWav.py**

Since Apple uses a proprietary file extension for their voice memo outputs (.m4a) and all of the functions require a .wav file to read I created a program, convertToWav.py that uses FFMPEG to convert any supported file extension to WAV format. The flexibility of this program was to cover any cases given to it by the end user. As required by our sponsor, this is an open source project and is released for free therefore this program accounts for someone submitting a file that's of .mp3 or some other generic audio format.

### **2.4.2. demoAudioSplit.py**

Once the end user has set up the whitelist (by adding voice recordings) and a call is received by the handset and the audio recording gets processed and passed through the database into my directories demoAudioSplit finds those files and creates 0.5 second split segments of these files for use in generating the gradient boost classification file and classifying the test files.

### **2.4.3. demoFeatureWrapping.py**

This program's purpose is unchanged from the previous **section 2.3.4**. The only change that needs to be made from the previous section is that all of the mentions of SVM have to be switched to gradient boost.

### **2.4.4. demoClassification.py / demoSingleClassification.py**

This functionality of this program also remains unchanged from the previous **section 2.3.5**. The single classification was created for testing since in the real use case the end user will only be testing with one test file. In contrast, for my own testing purposes the demoClassification.py file tests three files one for each of the members of this team.



### **3. Conclusion**

In the terms of this project, an aspect that was overlooked was the depth of the audio received from the landline phone. The handset recording is limited in the frequency range as well as removes a lot of the complexity in the voice. When heard back with human ears, it is apparent that the voice sounds very flat. The machine learning cannot extract enough features from the provided audio file to give an accurate prediction percentage. This subsystem should be taken as proof-of-concept. Under ideal conditions, the voice recognition does match and meet the requirements provided by our sponsor.

## 4. Validation

**Table 1. Validation and Test for Section 2.4.4.**

	Optimal Conditions	Updated Training Files	Random Forest	Gradient Boost (1 sec splits)	Gradient Boost (0.5 sec splits)
Amy	96.8	85.1	50.8	76.3	68.6
Matthew	93.4	58.8	52.4	87.8	84.4
Scott	84.2	31.8	41.6	51.6	62.0

All three of our voices had been classified and tested against with our own voice. **Table 1** shows the validation and test for **section 2.4.4**. All of the data used used the optimal testing files recorded through a quality microphone. This test compares the added test files to the classification file that was generated by **section 2.4.3**. This uses functions from pAA to extract features from the incoming test file. Features such as the ones discussed in **section 2.2**. This test passed in the ideal case only due to providing accurate predictions attempting to match the tested audio file with the trained gradient boost classification file. However, in all the other cases it failed. The algorithm was unable to consistently match the voices provided in **Table 1**, the 'Optimal Conditions' and 'Updated Training files' columns are tested with SVMs then the following columns are tested with different classification models supported by pAA. Additionally, I split gradient boost into two sections to try to see if more data points would help out the prediction and it turns out it did.

```
Calculating Averages:  
Testing against Amy's Files:  47.73267326732673  
Testing against Matthew's Files:  38.54455445544554  
Testing against Scott's Files:  11.396039603960396  
  
Process finished with exit code 0
```

**Figure 4. Testing An Unknown Voice**

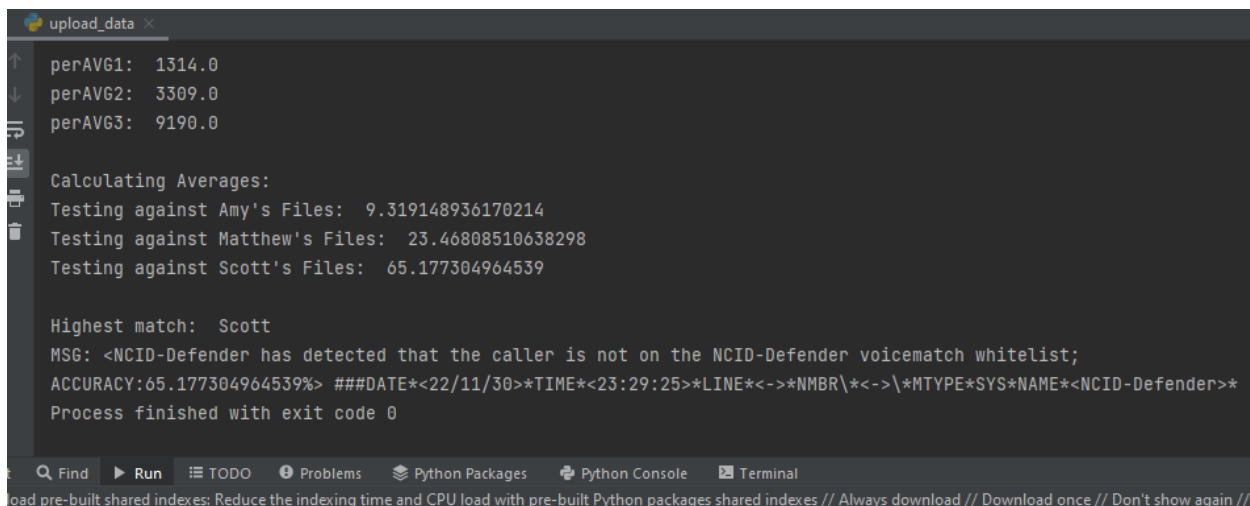
This is testing an unclassified voice (something that is not on the whitelist/blacklist) against the gradient boost model created from the current whitelist. **Figure 4** this is running the same test used in **Table 1**. This passes the test as none of the values result in a positive match (>75%) to any of the voices stored in the whitelist.

```
Calculating Averages:
Testing against Amy's Files:  9.319148936170214
Testing against Matthew's Files:  23.46808510638298
Testing against Scott's Files:  65.177304964539

Process finished with exit code 0
```

**Figure 5. Testing Handset Recording Against All Voices on White List**

**Figure 5** displays the result of testing the handset recording from the hardware subsystem against the gradient boost model created from the current whitelist. This test consisted of Amy talking into the microphone on the landline phone, that file is then sent to the database and processed to remove silence then sent to the machine learning to give a match. What is expected since Amy is the one speaking, her final average calculation should be high, however this is not the case. We believe this is due to the extra noise that the ESP32 creates in addition to the flat voice depth provided by the handset. The 300 - 3000 Hz frequency range isn't rich enough for pAA to extract enough data to provide an accurate prediction of who is speaking.



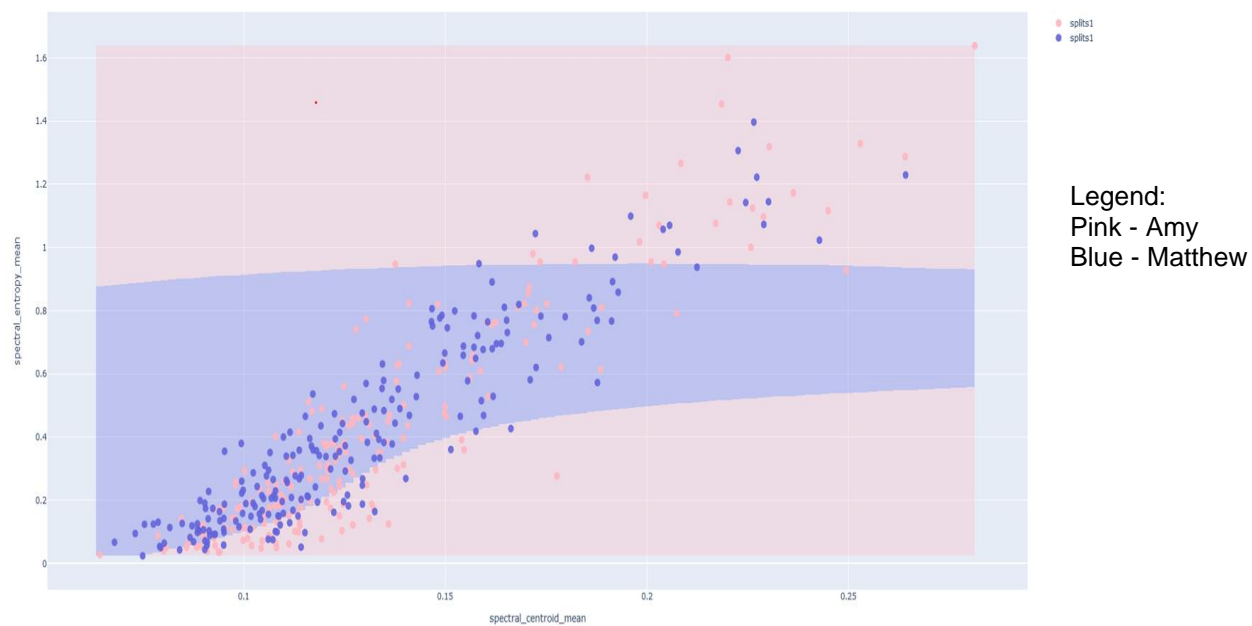
```
upload_data x
perAVG1: 1314.0
perAVG2: 3309.0
perAVG3: 9190.0

Calculating Averages:
Testing against Amy's Files:  9.319148936170214
Testing against Matthew's Files:  23.46808510638298
Testing against Scott's Files:  65.177304964539

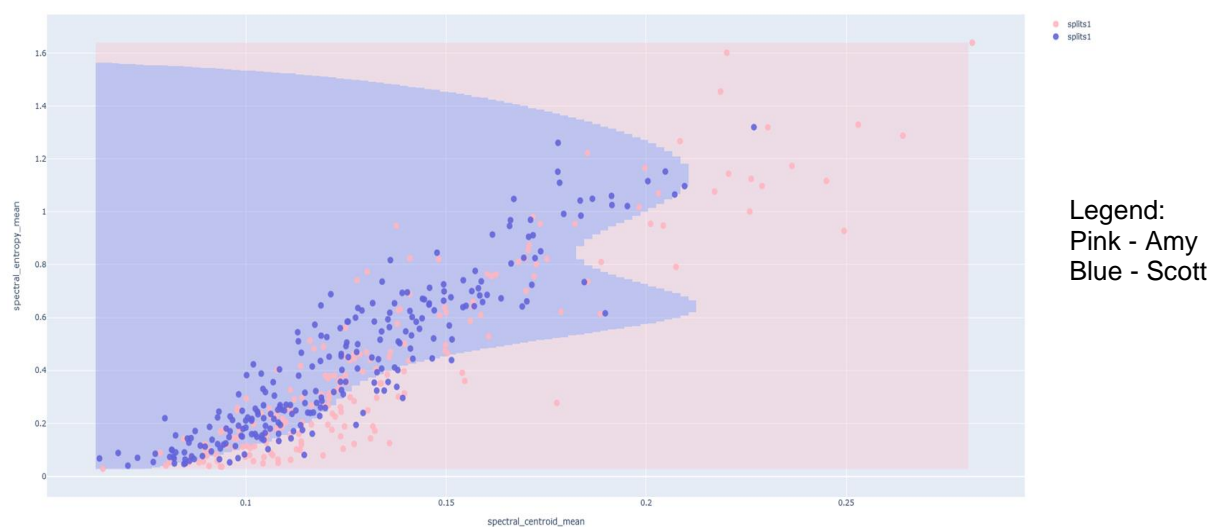
Highest match:  Scott
MSG: <NCID-Defender has detected that the caller is not on the NCID-Defender voicematch whitelist;
ACCURACY:65.177304964539%> ###DATE*<22/11/30>*TIME*<23:29:25>*LINE*<->*NMBR\*<->*MTYPE*SYS*NAME*<NCID-Defender>*
Process finished with exit code 0
```

**Figure 6. Integration with Database/Data Processing Subsystem**

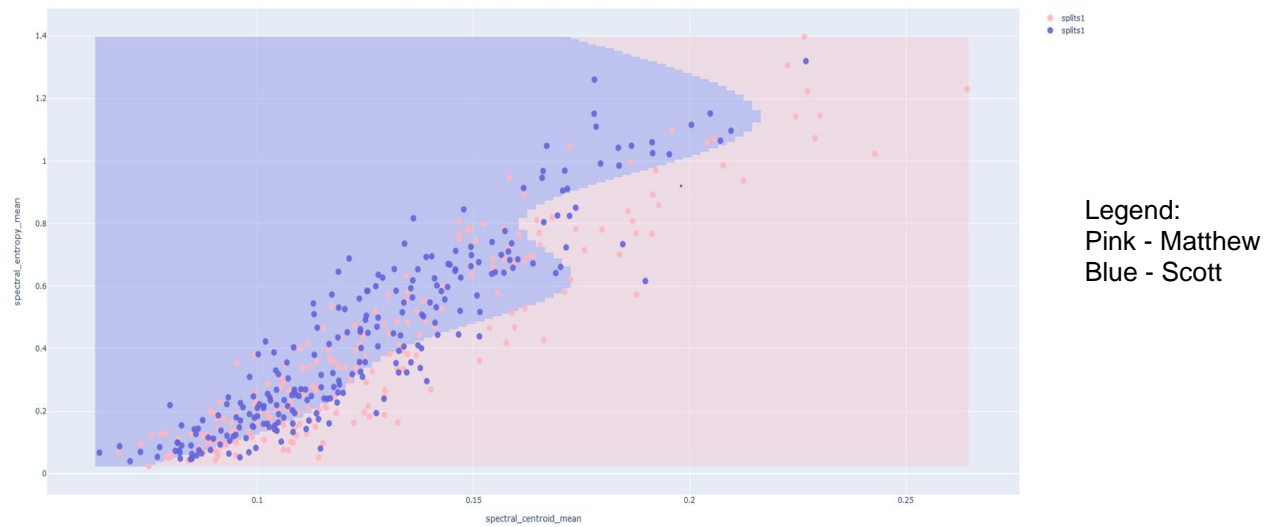
**Figure 6** displays the final integration between the machine learning subsystem and the database/data processing subsystem. Once the machine learning finishes the predictions it returns the highest confidence value as well as the name of the whitelist entry to the database subsystem and then the confidence value gets sent to the hardware and gets displayed on the LEDs on the board showing green if (>75% whitelist match) or red otherwise.



**Figure 7. Amy vs. Matthew**



**Figure 8. Amy vs. Scott**



**Figure 9. Matthew vs. Scott**

The three graphs in **Figure 7-9** display the same graphs just with different data from **section 2.3.3**. As it can be seen, when given less data the feature extraction cannot provide an accurate grouping of the data points. This is especially true in **Figure 7**. As addressed earlier, pAA is heavily reliant on feature-rich WAV files in order to do the processing and classification.

# The NCID Defender

Amy Chen

## **HARDWARE AND FIRMWARE SUBSYSTEM**

REVISION 1  
2 December 2022

# HARDWARE AND FIRMWARE SUBSYSTEM DOCUMENT FOR The NCID Defender

TEAM <28>

PREPARED BY:

---

Amy Chen Date

APPROVED BY:

---

Project Leader Date

---

Dr. Wonhyeok Jang Date

---

Rohith Kumar Date

## Change Record

Rev.	Date	Originator	Approvals	Description
0	4/30/2022	Amy Chen		Final Report 403 Submission
1	12/2/2022	Amy Chen		Revision 1: changes for Final Report 404 Submission



## Table of Contents

<b>Table of Contents .....</b>	<b>III</b>
<b>List of Tables .....</b>	<b>V</b>
<b>List of Figures .....</b>	<b>VI</b>
<b>1. Subsystem Introduction .....</b>	<b>1</b>
<b>2. Power Supply .....</b>	<b>2</b>
2.1. Raspberry Pi 4b .....	2
2.2. Espressif-32 .....	2
2.3. WS2818B LEDs .....	2
2.4. 128x32 OLED Displays .....	2
<b>3. Hardware Circuit Design .....</b>	<b>3</b>
3.1. EasyEDA .....	3
3.1.1. LCSC Electronics .....	3
3.1.2. JLCPCB .....	3
3.2. The NCID Defender Schematic .....	4
3.2.1. Ring Detection and Ring Suppression .....	4
3.2.2. On/Off Hook Detection .....	4
3.2.3. Phone Pick-Up Detection .....	5
3.2.4. Audio Conditioning .....	5
3.2.5. Displays .....	5
3.3. PCB Design .....	5
3.3.1. New Changes for Future Improvement .....	7
3.4. PCB Testing .....	7
3.4.1. Tip/Ring Voltage .....	9
3.4.2. Ring Detection/Suppression .....	9
3.4.3. Phone Pick-Up .....	10
3.4.4. On/Off Hook .....	11
3.4.5. Audio Conditioning .....	12
3.4.6. Displays .....	15
<b>4. Firmware Code (C++) for Espressif-32 .....</b>	<b>17</b>
4.1. AFSK.cpp / AFSK.h .....	17
4.1.1. void AFSK_hw_init(void) .....	17
4.2. Config.cpp / Config.h .....	17
4.3. constants.h .....	17
4.4. device.h .....	17
4.5. FIFO.h .....	17
4.6. Globals.h .....	17

4.7.	main.cpp.....	18
4.7.1.	void cidSM() .....	18
4.7.2.	void record(I2SSampler *input, const char *fname).....	18
4.7.3.	void setup() .....	18
4.7.4.	void loop() .....	18
4.7.4.1.	void onHookNoCall() .....	18
4.7.4.2.	void offHookCallConn() .....	19
4.7.4.3.	void offHookOutgoing().....	19
4.7.4.4.	void onHookCallEnd().....	19
4.8.	PhoneDTMF.cpp / PhoneDTMF.h .....	19
4.9.	XMDF.cpp / XMDF.h .....	19
<b>5.</b>	<b>Conclusion .....</b>	<b>20</b>
<b>6.</b>	<b>Validation Plan .....</b>	<b>21</b>
<b>7.</b>	<b>Additional References .....</b>	<b>22</b>

## **List of Tables**

<b>Table 1. Hardware and Firmware Validation Tasks List.....</b>	<b>21</b>
--	-----------

## List of Figures

Figure 1. Subsystem Block Diagram.....	1
Figure 2. The NCID Defender Schematic in EasyEDA.....	4
Figure 3. PCB Design .....	5
Figure 4. PCB Design (2D View) .....	6
Figure 5. PCB Design (3D View) .....	6
Figure 6. The NCID Defender Case.....	7
Figure 7. PCB Prototype Version X2 .....	7
Figure 8. PCB Hooked up with Landline and Laptop .....	8
Figure 9. Tip/Ring Voltage.....	9
Figure 10. Phone Pick-Up Voltage.....	10
Figure 11. On/Off Hook Voltage and Hook Flash .....	11
Figure 12. Audio Attenuation Waveform.....	12
Figure 13. Speaking into Landline Phone Audio Waveform.....	13
Figure 14. Speaking into Landline Phone Close-Up Audio Waveform .....	14
Figure 15. OLED Display Phone State.....	15
Figure 16. WS2818B LEDs Scam Call .....	15
Figure 17. The NCID Defender Case When Scammer Calls.....	16

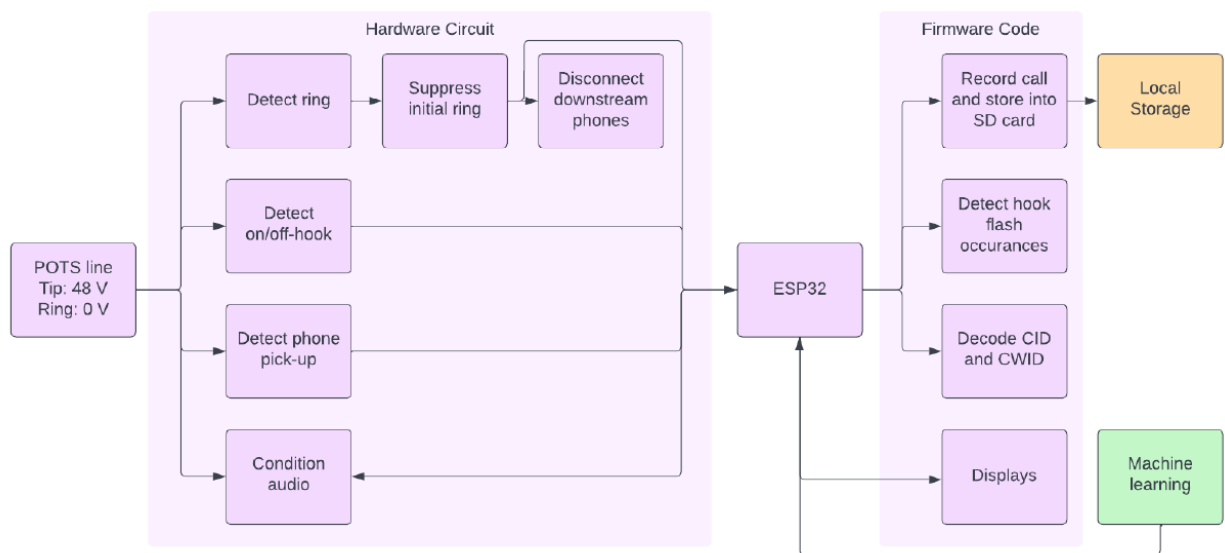
## 1. Subsystem Introduction

The NCID Defender Hardware and Firmware subsystem receives incoming analog signals from the POTS line and feeds the signals into the Espressif-32 (ESP32) for analysis. This subsystem consists of determining the power supply, designing the circuit, coding firmware decoding signals for the ESP32, and setting up display indicators.

This subsystem is responsible for deciding on the power supply to switch on the host computer (in this project, we will be utilizing the Raspberry Pi as the host computer) and ESP32. The hardware will condition the incoming audio to an acceptable voltage level so that the ESP32 microcontroller can detect and decode signals without failure. In addition, it be detecting an incoming phone ring, suppressing the initial ring, disconnecting the downstream phones momentarily, detecting on and off-hook, and detecting when the phone is picked up. The firmware will record calls, detect hook flash occurrences, detect DTMF tones, and decoding caller identification (CID) and caller waiting identification (CWID).

The NCID Defender device will display CID on an OLED display and indicate scam callers with an LED indicator (75% confidence interval). The most considerable challenge for this subsystem is creating an online simulation to verify my circuit design.

Below is a diagram for the hardware and firmware detailing the connecting pieces and how it interacts with the other subsystems.

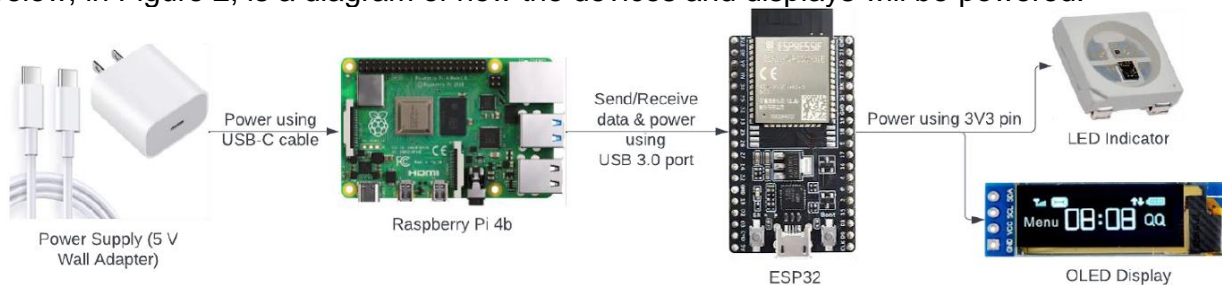


**Figure 1. Subsystem Block Diagram**

## 2. Power Supply

Our team decided to use the Raspberry Pi as our host computer. The device may vary in end cost, size, and processing ability because our code and device are designed to be compatible with other consumer-level development kit boards such as the TI LaunchPad and Espressif-32 (ESP32). We decided to use the ESP32 microcontroller (MCU) as our primary device because it already has an audio bus for call recording. It also has a dual-core and ultra-low power co-processor. The ESP32 can perform as a complete standalone system or as a slave device to a host MCU (in our case, the Raspberry Pi), reducing communication stack overhead on the central application processor. Additionally, the ESP32 can interface with other systems to provide Wi-Fi and Bluetooth functionality through its SPI/SDIO or I2C/UART interfaces.

Below, in Figure 2, is a diagram of how the devices and displays will be powered.



### 2.1. Raspberry Pi 4b

As per official documentation from Raspberry Pi, the Raspberry Pi 4 requires a 15.3 W USB Type-C power supply with a DC output of 5.1 V with a minimum of 3.0 A.

### 2.2. Espressif-32

The ESP32 can be powered using a Micro USB port and VIN pin (External Supply Pin). The power required by ESP32 is 600 mA, as ESP32 pulls as much as 250 mA during RF transmissions. During boot or WiFi operation, it draws more than 200 mA current. However, the ESP32 will be powered by the Micro USB port, so WiFi will only be used when the micro USB connection is unstable.

### 2.3. WS2818B LEDs

The WS2812B LEDs require about 5 V to work. The WS2812B should operate anywhere between 3.3 V to 5 V. Therefore, the WS2818B LEDs will be powered by the 3.3 V pin on the ESP32.

### 2.4. 128x32 OLED Displays

The OLED and driver require a 3.3 V power supply and 3.3 V logic levels for communication. The power requirements depend a little on how much of the display is lit, but on average, the display uses about 20 mA from the 3.3 V supply. Built into the OLED driver is a simple switch-cap charge pump that turns 3.3 V to 5 V into a high voltage drive for the OLEDs. It is possible to run the entire display off one 3.3 V supply. Therefore, the 128x32 OLED display will be powered by the 3.3 V pin on the ESP32.

## **3. Hardware Circuit Design**

### **3.1. *EasyEDA***

EasyEDA is a free and easy online circuit design, circuit simulator, and PCB design tool. This EDA was chosen based off of our project being open-sourced. Schematics and PCB designs can be easily downloaded from a shareable link. Furthermore, EasyEDA works closely with LCSC Electronics and JLCPCB.

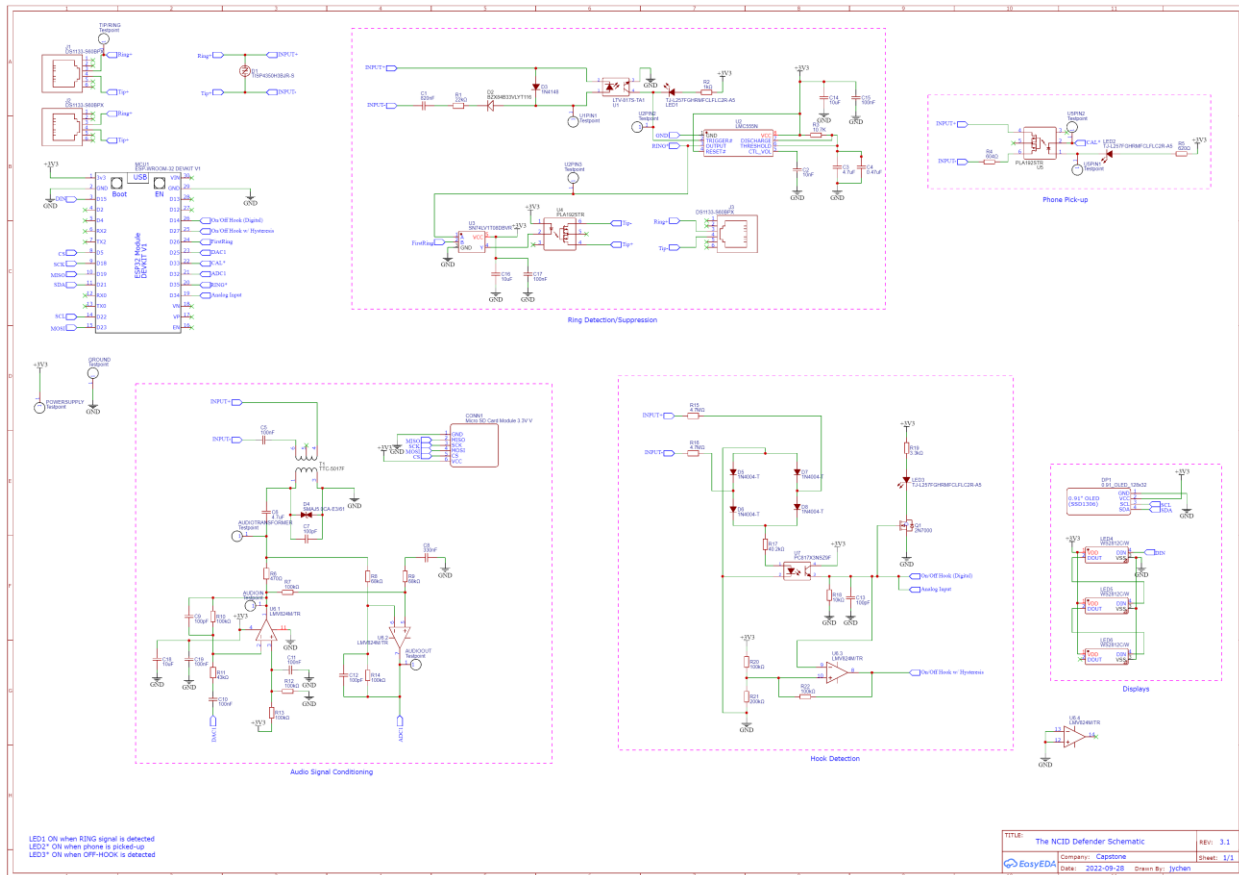
#### **3.1.1. LCSC Electronics**

LCSC Electronics is China's leading electronic components supplier. The company provides different electronic components such as passive components, sensors, integrated circuit (IC) chips, development modules, and many more. Their components are high quality and cheap in price. They have a wide collection of both surface mount and through hole components. They are also known for their high-speed delivery network.

#### **3.1.2. JLCPCB**

JLCPCB is China's largest PCB prototype enterprise, and their high-tech manufacturer specializes in developing these prototypes within hours. The company works everyday, all day, nonstop. The company can develop PCBs of various complexities, from simple and cheap with single layer boards for hobbyists to multi-layer boards for industrial applications.

## 3.2. The NCID Defender Schematic



**Figure 2. The NCID Defender Schematic in EasyEDA**

This schematic is based on Locus Engineering's E2210 Telephone Line Interface device.

### 3.2.1. Ring Detection and Ring Suppression

On the top middle section of Figure 2 is the Ring Detection and Ring Suppression subcircuit. The circuitry for detect phone rings (indicated by LEDs) and suppresses the initial ring. The incoming signal is fed through the optocoupler. Then, the output signal from the optocoupler will be fed into a 555 timer to determine when the initial ring is. Finally, it is fed into a solid state relay (SSR) to disconnect the downstream phones to send data to the firmware and software to decode incoming caller ID.

### 3.2.2. On/Off Hook Detection

On the bottom middle section of Figure 2 is the On/Off Hook Detection subcircuit. The circuitry is designed to connect in parallel with the telephone line to monitor and detect if any telephone is on or off-hook. When off-hook is detected, an LED turns off. There are three output signals in the schematic above, one without hysteresis, one with hysteresis using op-amps as comparators, and one analog output. Hysteresis is to test if there is too much noise in the signal. A signal with hysteresis clears up the incoming signal so that process of detecting on/off-hook is easier. The analog output is for future firmware implementation of the project.



### 3.2.3. Phone Pick-Up Detection

On the top right of Figure 2 is the Phone Pick-Up subcircuit. When the solid-state relay (SSR) detects that the phone is being picked-up (CAL\* is grounded), the LED will light up.

### 3.2.4. Audio Conditioning

On the bottom right of Figure 2 is the Audio Conditioning subcircuit. The incoming analog audio signal will go through a transformer and be conditioned through the op-amp. The output voltage signal of the op-amp will be an input into the ESP32 through the analog-to-digital converter (ADC) so that the signal can be transformed into a signal that the ESP32 can read and analyzed. After sending the signal to the ESP32, the ESP32 will start recording the call to an SD card. Then, the ESP32 will send the saved recording to the local database to be further analyzed. When the ESP32 receives a package from the database to play a sound file, the sound will be played to the phone through the digital-to-analog (DAC) pin on the ESP32.

### 3.2.5. Displays

On the bottom right of Figure 2 is the displays subcircuit. The firmware code will activate these displays based on a 75% confidence interval.

## 3.3. PCB Design

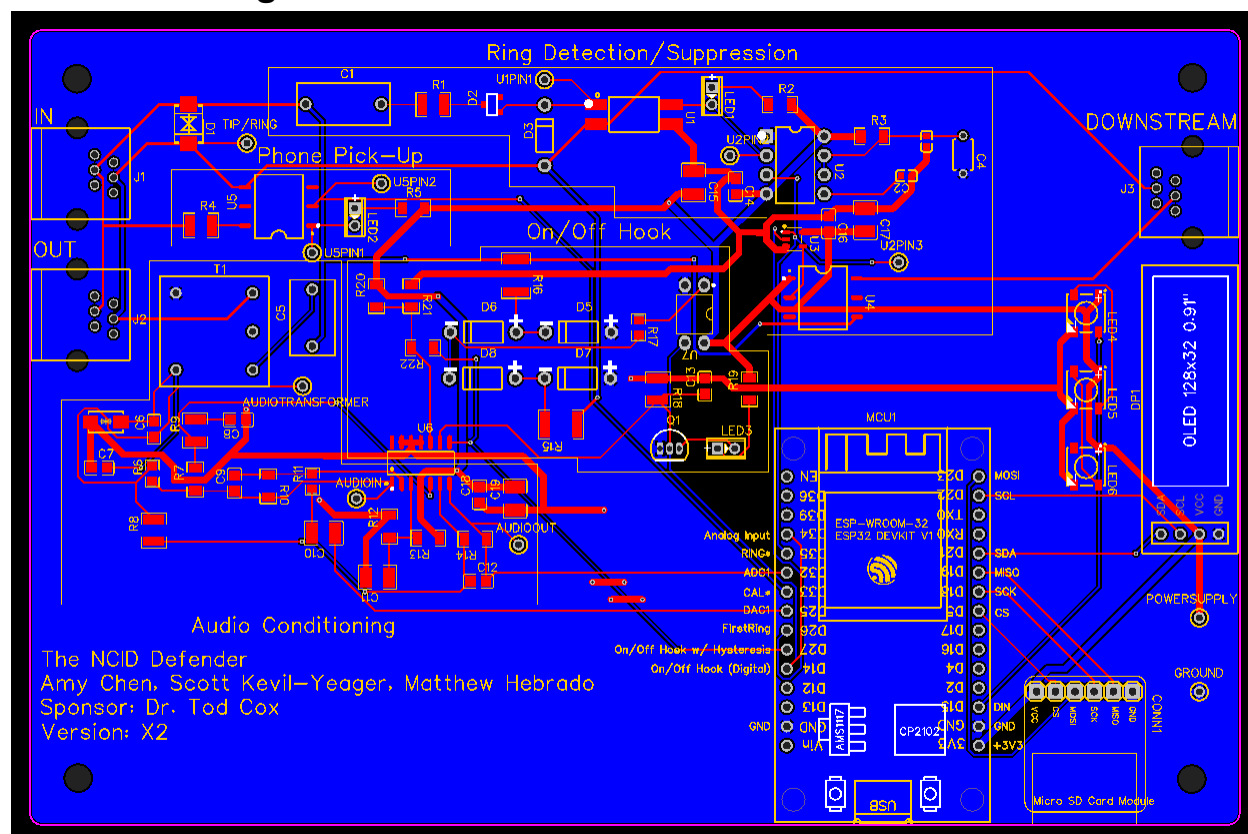


Figure 3. PCB Design



### 3.3.1. New Changes for Future Improvement

Some features of the PCB have been reworked to keep in mind before the future final initial release of the board:

- Connect floating ground on C4
- Replace R4 from 604  $\Omega$  to 220  $\Omega$
- Remove R15 and R16
- Add testpoints for on and off-hook pins
- Add a push button for easy testing of recording

### 3.4. PCB Testing



Figure 6. The NCID Defender Case

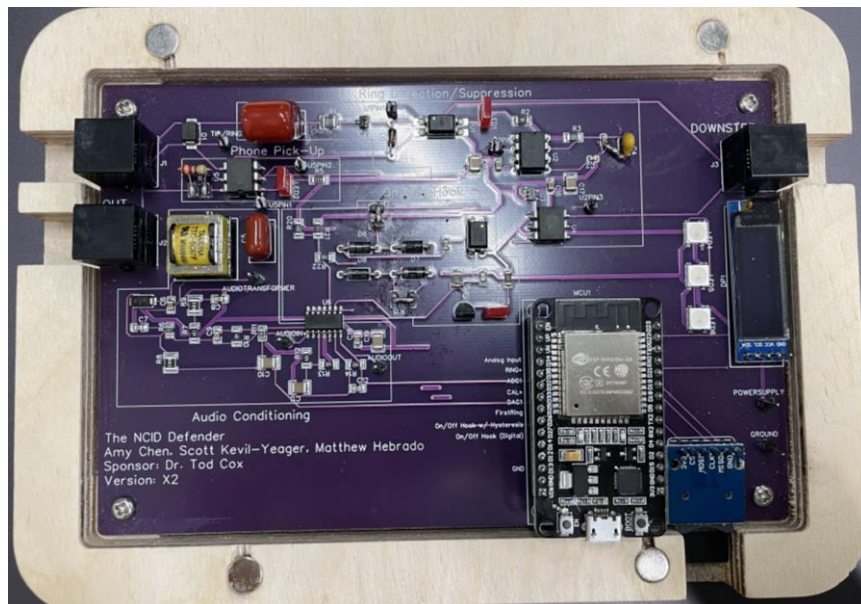


Figure 7. PCB Prototype Version X2



Figure 8. PCB Hooked up with Landline and Laptop

### 3.4.1. Tip/Ring Voltage

The voltage on a POTS phone line should measure around 48 V DC. The oscilloscope reading of tip/ring through The NCID Defender PCB measured around 45 V DC. During ringing, the voltage of the phone momentarily becomes 90 V<sub>rms</sub> with a 20 Hz AC signal.

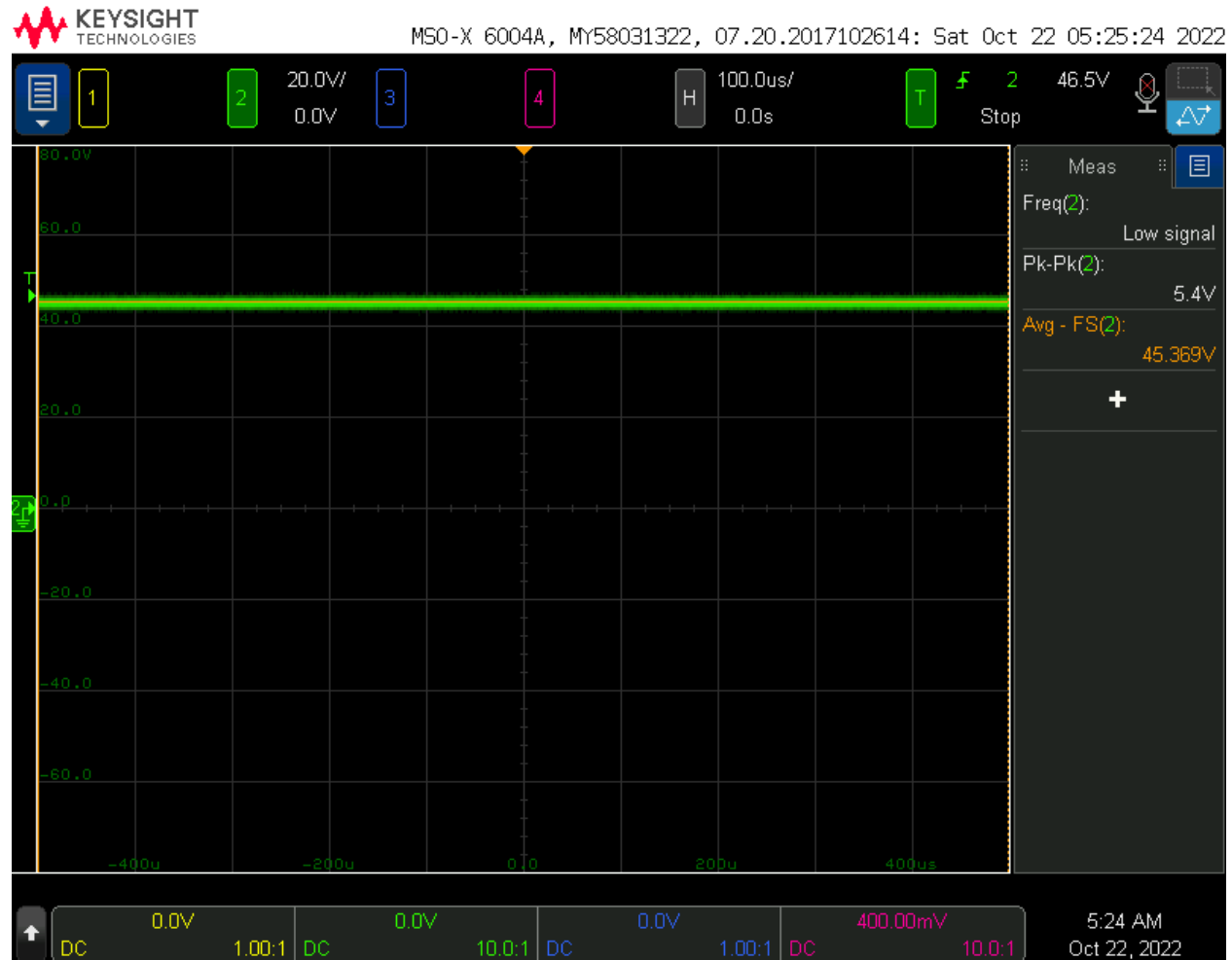


Figure 9. Tip/Ring Voltage

### 3.4.2. Ring Detection/Suppression

LED1 flickers whenever the phone is ringing, detecting ring.

When U3PIN2 (FirstRing) pin is pulled to ground, ring is suppressed. Initial ring suppression is implemented in the hardware.



### 3.4.3. Phone Pick-Up

When U5PIN2 (CAL\*) pin is pulled to ground, LED2 will light up, indicating that the phone is in use. The phone shows “Line in use” about 7 seconds after CAL\* is pulled to ground. The tip/ring voltage is 4.3543 V DC.

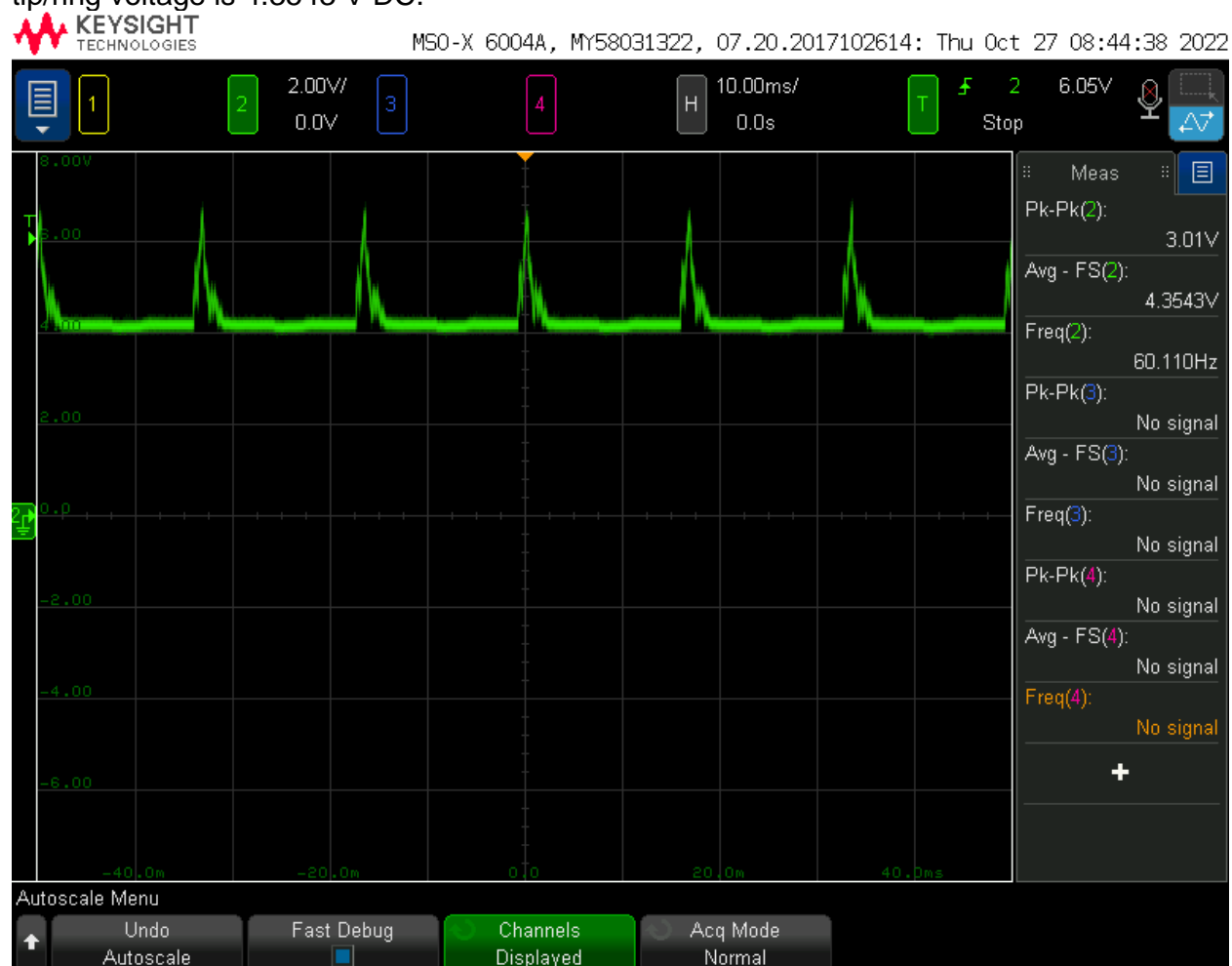


Figure 10. Phone Pick-Up Voltage

### 3.4.4. On/Off Hook

On-hook voltage measured 3.1 V and off-hook voltage measured 325 mV. Hook flash lasts about 604 ms, this is useful for the ESP32 to detect hook flash occurrences.

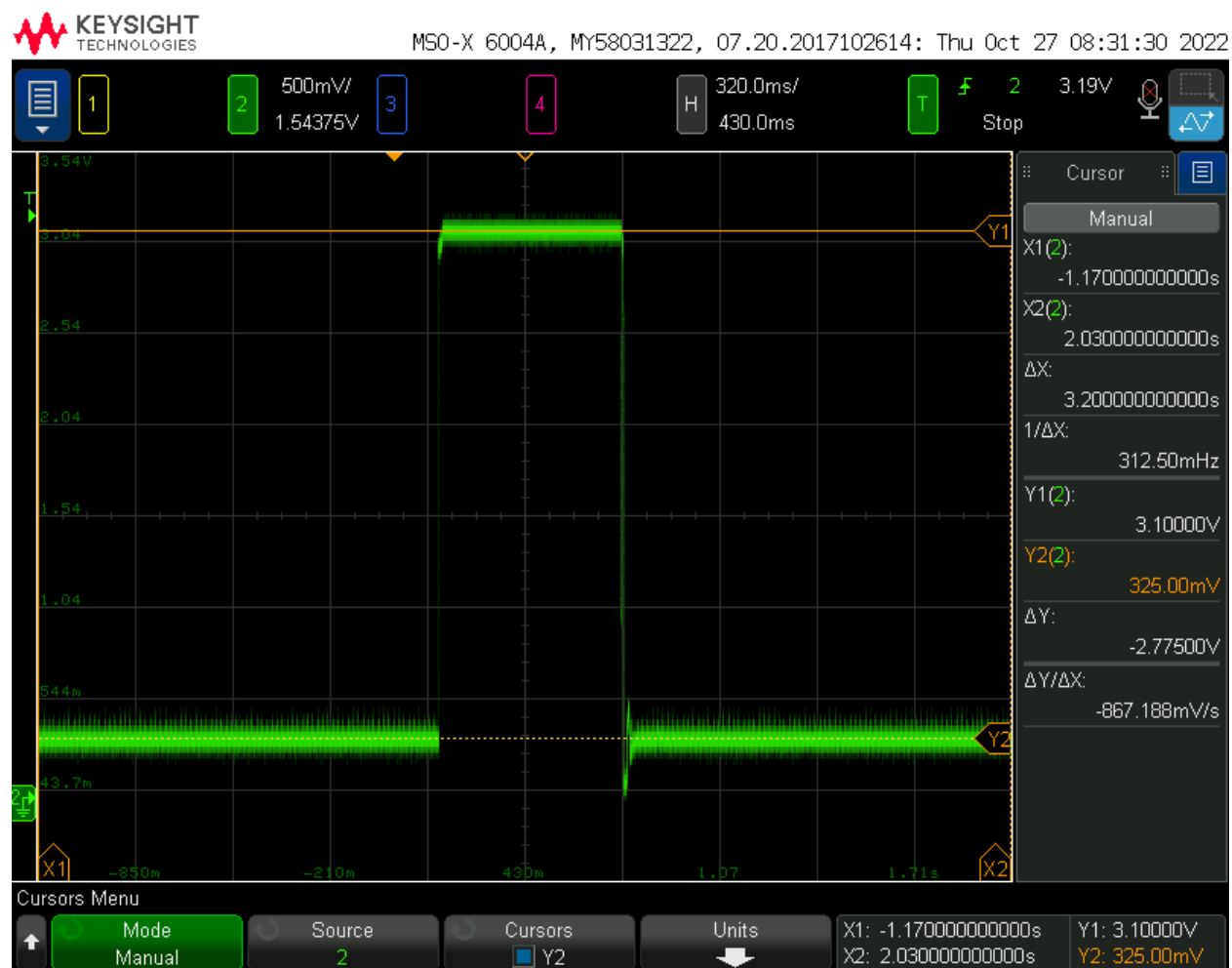


Figure 11. On/Off Hook Voltage and Hook Flash

### 3.4.5. Audio Conditioning

When measuring the AUDIOOUT (ADC) pin, the attenuation waveform had a DC bias of 1.6 V with around 1-2 kHz frequency.

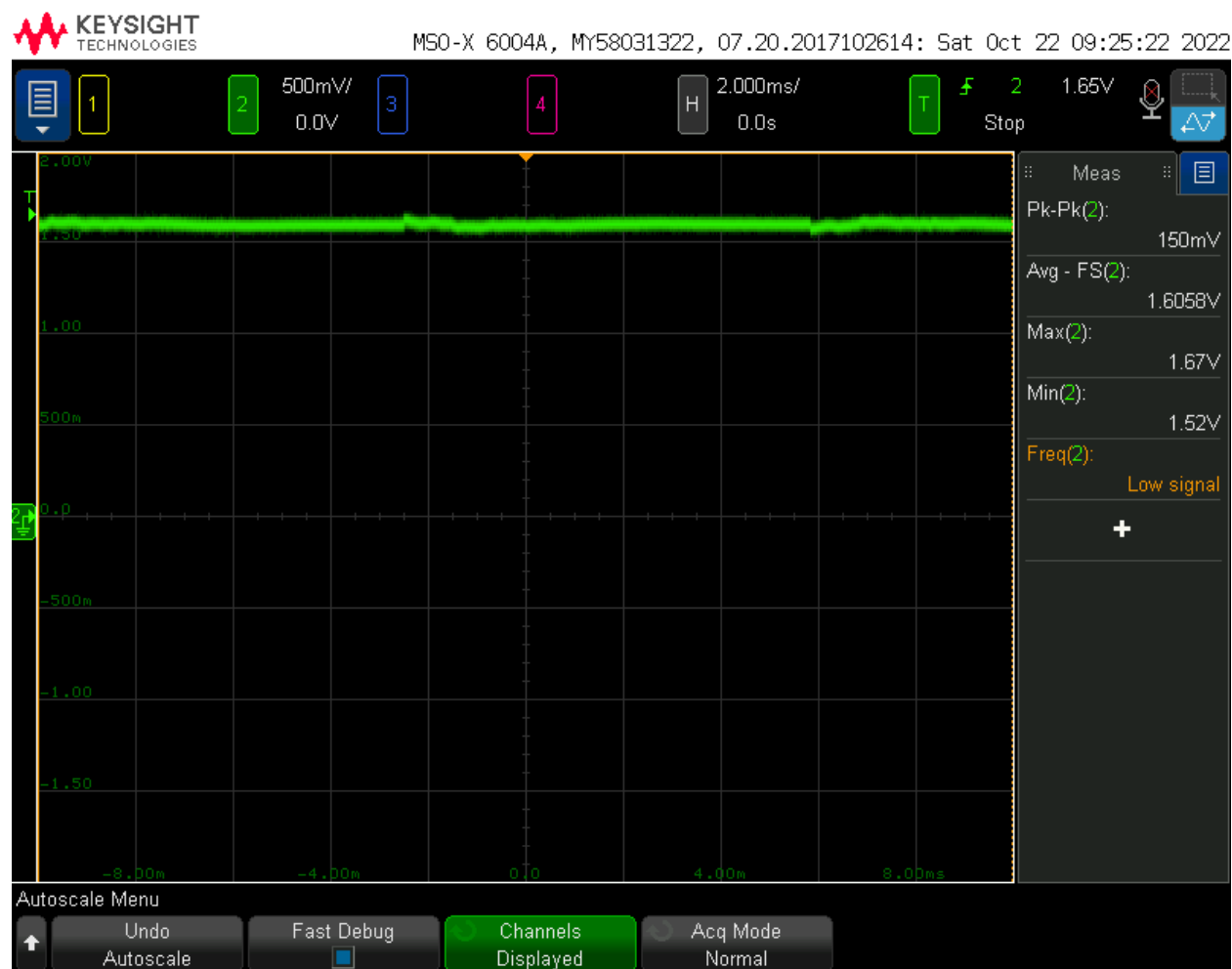


Figure 12. Audio Attenuation Waveform



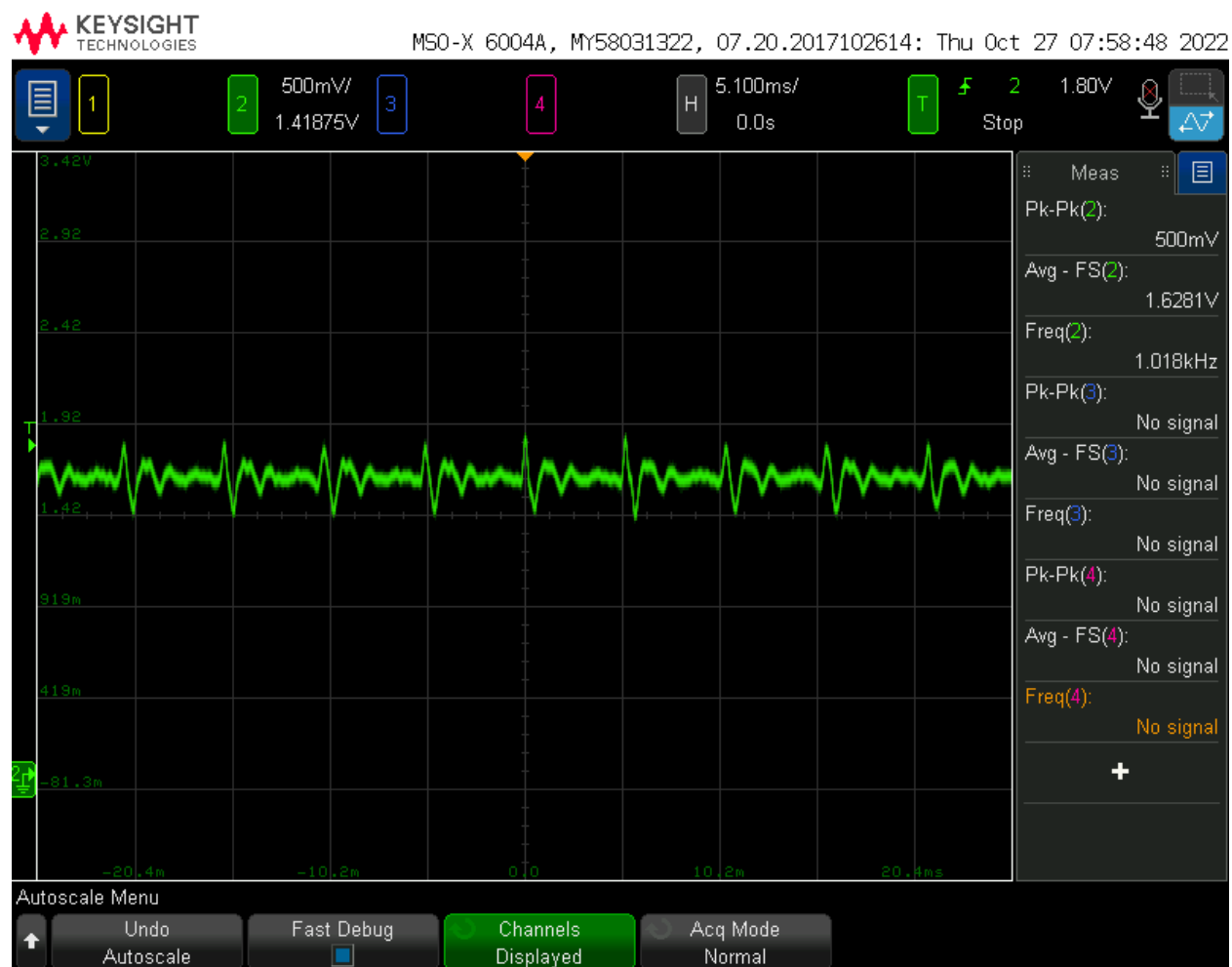


Figure 13. Speaking into Landline Phone Audio Waveform

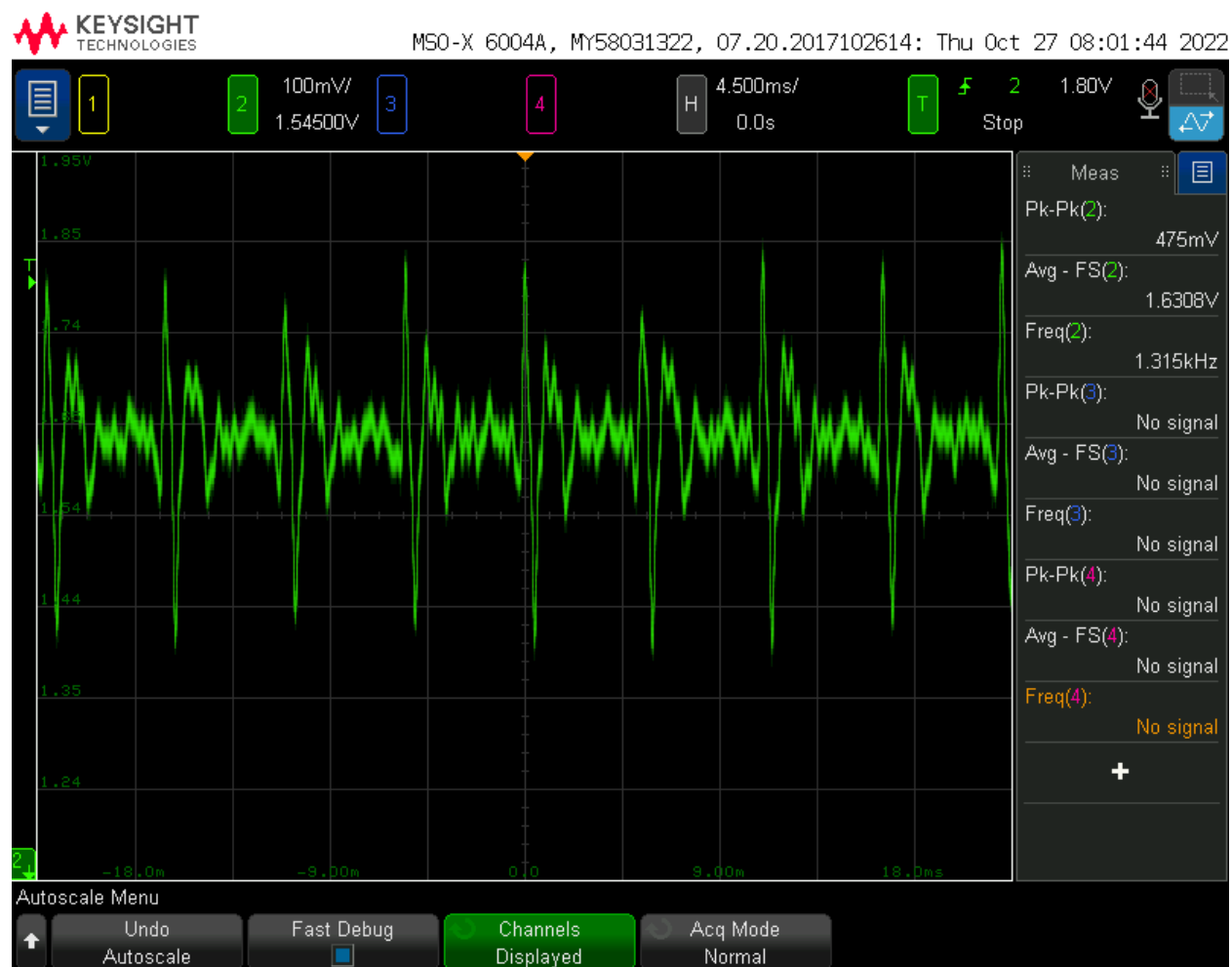


Figure 14. Speaking into Landline Phone Close-Up Audio Waveform

### 3.4.6. Displays

128x32 OLED display can currently show phone state and/or caller ID.

WS2818B LEDs will show red if caller is a scammer, green if caller is know, and nothing when waiting for the call.



Figure 15. OLED Display Phone State

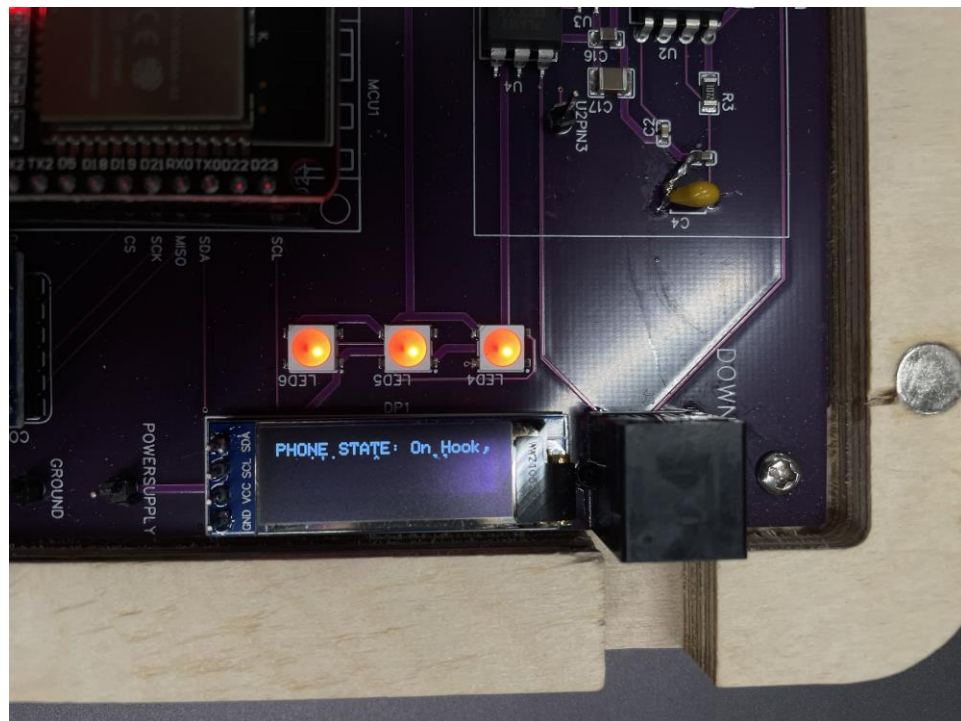


Figure 16. WS2818B LEDs Scam Call



**Figure 17. The NCID Defender Case When Scammer Calls**

## **4. Firmware Code (C++) for Espressif-32**

The firmware state machine is designed to decode caller information, detect on and off-hook, detect hook flash occurrences, detect DTMF tones, record calls, and communicate with the database subsystem.

The caller ID state machine code incorporates code designed by Mark Qvist and 6v6gt. To detect DTMF tones, refer to Adrianotiger and the Goertzel method by jacobrosenthal.

### **4.1. AFSK.cpp / AFSK.h**

AFSK.cpp and AFSK.h are adapted code files developed by Mark Qvist and 6v6gt. Since 6v6gt wrote these files using an ESP-8266 and the ESP-8266 does not have an internal ADC pin, he had to use an external ADC module. I tried to implement Evan Krall's way of taking ADC samples through the ESP-32's I2S (section 4.1.1).

The ADC ISR drives the AFSK demodulator; it decodes the input signal into 0s and 1s, strips the start and stop bits, drops bit stuffing, and handles the endian conversion. It recognizes the channel seizure and mark block, which comes before the data stream, and then strips these out. The remaining data stream is then passed to the byte assembler.

#### **4.1.1. void AFSK\_hw\_init(void)**

AFSK\_hw\_init is a function written by Mark Qvist to interface I2S and the ESP32's ADC channel.

### **4.2. Config.cpp / Config.h**

Config.cpp and Config.h was adapted from 6v6gt's code from his "Arduino Esp8266 Based Telephone Caller ID System With Anti-Spam Function" project.

Config.h also has a configuration setting to record a call on an SD Card.

### **4.3. constants.h**

This file is used to store constants of the ESP32 (adapted from Mark Qvist).

### **4.4. device.h**

This file was written to configure the ESP32 (adapted from Mark Qvist) for interfacing I2S with the ADC channel.

### **4.5. FIFO.h**

Mark Qvist wrote FIFO to organize the incoming bit data by First In, First Out.

### **4.6. Globals.h**

Globals file was written to assist mainly with debug modes for this project (adapted from 6v6gt).

## **4.7. main.cpp**

The main file is where the phone and caller identification state machine are written.

### **4.7.1. void cidSM()**

The caller ID state machine is code adapted from 6v6gt. This state machine includes the following functions:

1. void AFSKReady(DemodState\_t lastAFSKState, uint32\_t mstime)
2. void AFSKAltMark(DemodState\_t lastAFSKState)
3. void AFSKAllMarks(DemodState\_t lastAFSKState)
4. void AFSKData(DemodState\_t lastAFSKState)
5. void AFSKCleanUp(DemodState\_t lastAFSKState)
6. void AFSKErrorx(DemodState\_t lastAFSKState)

Caller ID is acquired from the AFSKCleanUp function. This data is sent to the database and, at the same time, displayed on the 128x32 OLED display.

### **4.7.2. void record(I2SSampler \*input, const char \*fname)**

The database subsystem wrote the record function. This function integrates the hardware and firmware with the database. Its purpose is to write voice data into an SD card for as long as the phone is off-hook.

My subsystem has modified this function to detect hook-flash occurrences of the phone as long as the phone is off-hook..

### **4.7.3. void setup()**

The setup function sets up input and output pins, the SD card, the OLED display, and WS2818b LEDs. Then, the AFSK and DTMF libraries so that their functions are ready to be called at any time. Lastly, the phone and AFSK states are set to their initial states.

### **4.7.4. void loop()**

The loop function holds the phone state machine. The four states are onhook\_nocall, offhook\_callconn, offhook\_outgoing, and onhook\_callend.

#### **4.7.4.1. void onHookNoCall()**

The first phone state is when the phone is on-hook and is waiting for a call. It detects when the RING pin is triggered (which means the phone is ringing) and then suppresses the first ring for downstream phones. While the ring is suppressed, cidSM() is called to decode caller ID. If the caller ID matches with a scammer, the ESP-32 will receive a JSON packet from the database to play a fax tone or do an NCID hang-up. The LEDs will also indicate red when this happens. If the caller ID matches with a known or unknown person, the LEDs will indicate green, and The NCID Defender device will let the call keep ringing until the user picks up. However, if the ringing times out after some time and the phone is not picked up, the phone is automatically hung up, and the next state is onhook\_callend.

If the user or the ESP-32 receives a JSON packet from making an outgoing call from NCID, the next state is offhook\_outgoing.

#### **4.7.4.2. void offHookCallConn()**

The second phone state is when the call is connected. Immediately, a JSON packet is sent to NCID to indicate the call start, and recording of the call begins. If the user hangs up, the next state is onhook\_callend. If a howler tone is detected, the ESP-32 contacts NCID to alert the user; when on-hook is detected, the next state is onhook\_callend.

While the first call is ongoing, this function should look out for any DTMF key tones and incoming calls. These have not yet been implemented at the time of writing this report.

#### **4.7.4.3. void offHookOutgoing()**

The third phone state is when the user is making an outgoing call. As soon as the phone goes off-hook, NCID will receive a JSON packet indicating the start of the call, and recording will begin.

This function should also be looking for a solid or stutter tone. If DTMF pressed was a valid phone number, the next state is offhook\_callconn; if not, the next state is onhook\_callend. These have not yet been implemented at the time of writing this report.

#### **4.7.4.4. void onHookCallEnd()**

The last phone state is when the call has ended, and the user puts the phone back on-hook. The recording is ended and sent to the database. A JSON packet is also sent to NCID with an indication of the end time of the call. The LEDs are reset to no color, and the next state is onhook\_nocall.

### **4.8. *PhoneDTMF.cpp / PhoneDTMF.h***

This library was adapted from Adrianotiger to detect DTMF key pad tones easily. This library has not been used at the time of writing this report.

### **4.9. *XMDF.cpp / XMDF.h***

XMDF files were adapted from 6v6gt to parse incoming bits. parseDemodBitQueue(bool inbit) collects bits from the demod queue and assembles the bytes. The bytes are then handed over to the byte parser parseMdmf(). This function returns either a 0 (no state change), 1 (task completed), or 2 (error). ParseMdmf(byte index, byte value) is called with every byte found in the data stream, including xDMF header byte and except the checksum byte.

## **5. Conclusion**

Near the end of this project, the team realized many properties of a POTS landline phone that we had overlooked earlier. The ADC pin on the ESP-32 created a thick layer of noise that could not be fixed in time for the demonstration. A Butterworth filter was created to help the noise, but the results were different from what was expected. In addition, when the audio files were opened in the Audacity program, the voices were unexpectedly too quiet. Before the final initial version of the board is released, we mentioned to our sponsor to add an op-amp before the ADC channel to amplify the voice audio. However, in conclusion, The NCID Defender device has taught the team many lessons even though it is not fully complete.



## 6. Validation Plan

Depicted in Table 1 is my validation of each subtask within my subsystem.

**Table 1. Hardware and Firmware Validation Tasks List**

Test	Detail	Data	Status	Responsible Student
Device powers on	Turns on Raspberry Pi and ESP32	Turns on	Complete	Amy Chen
Display powers on	Displays caller ID information		Complete	Amy Chen
Ring detect	LED1 lights up when detection occurs	48 V DC to sine wave	Complete	Amy Chen
Ring suppress	Initial ring is suppressed		Complete	Amy Chen
Phone pick up	LED2 lights up	LED lights up when CAL* is grounded	Complete	Amy Chen
Audio Conditioning Out	Phone audio to ADC1 pin		Complete	Amy Chen
Audio Conditioning In	DAC1 pin to audio		Complete	Amy Chen
Detect off-hook/on-hook	LED3 lights up when detection occurs		Complete	Amy Chen
Detect hook flash on ESP32	Detect hook flash in firmware	Code written, not tested	Incomplete	Amy Chen
Decode CID/CWID on ESP32	Decode CID/CWID information in firmware	Code written, not tested	Incomplete	Amy Chen
Decode DTMF and FSK on ESP32	Decode DTMF and FSK in firmware	Code written, not tested	Incomplete	Amy Chen
OLED program	Code for OLED display		Complete	Amy Chen
WS2812B program	Code for LED light		Complete	Amy Chen
Control WS2812B	Test code on LED light		Complete	Amy Chen

## 7. Additional References

[1] LibAPRS an Arduino soft modem library which provided the basis for the AFSK demodulator code.

<https://github.com/markqvist/LibAPRS>

[2] “Arduino Esp8266 Based Telephone Caller ID System With Anti-Spam Function” project by 6v6gt.

<https://forum.arduino.cc/t/esp8266-arduino-telephone-caller-id-system-with-anti-spam-feature/507603>

[3] E2210 Telephone Line Interface Datasheet by Locus Engineering

<https://locus-engineering.com/wp-content/uploads/2018/08/E2210-Telephone-Line-Interface-DataSheet.pdf>

[4] PhoneDTMF library by Adrianotiger

<https://github.com/Adrianotiger/phoneDTMF>

[5] Goertzel method by jacobrosenthal

<https://github.com/jacobrosenthal/Goertzel/blob/master/Goertzel.cpp>

The NCID Defender  
Scott Kevil-Yeager

# **DATABASE AND DATA PROCESSING SUBSYSTEM**

REVISION 1  
2 December 2022

# DATABASE AND DATA PROCESSING DOCUMENT FOR The NCID Defender

TEAM <28>

PREPARED BY:

---

Scott Kevil-Yeager                      Date

APPROVED BY:

---

Project Leader                      Date

---

Dr. Wonhyeok Jang                      Date

---

Rohith Kumar                      Date

## Change Record

Rev.	Date	Originator	Approvals	Description
0	4/30/2022	Scott Kevil-Yeager		Final Report 403 Submission
1	12/2/2022	Scott Kevil-Yeager		Revision 1: changes for Final Report 404 Submission

## Table of Contents

<b>Table of Contents .....</b>	<b>III</b>
<b>List of Tables .....</b>	<b>IV</b>
<b>List of Figures .....</b>	<b>V</b>
<b>1. Subsystem Introduction .....</b>	<b>1</b>
<b>2. Python Files and Functions .....</b>	<b>2</b>
2.1. Calls.py .....	2
2.2. Incoming_callers.py .....	2
2.3. Mongoengine_setup.py .....	2
2.4. Receiving_callers.py .....	2
2.5. Obtain_call_length.py .....	3
2.5.1. Audio_duration .....	3
2.5.2. Creation Date .....	3
2.5.3. Obtain_length_of_call .....	3
2.6. Remove_silence.py .....	3
2.7. Upload_data.py .....	4
2.7.1. Retrieve_recording .....	4
2.7.2. Delete_local_recording .....	4
2.7.3. Delete_database_recording .....	4
2.7.4. Error_check .....	4
2.7.5. Error_check_file .....	5
2.7.6. Num_files .....	5
2.7.7. Upload_folder .....	5
2.8. User_interface.py .....	5
2.9. Main.py .....	5
2.10. JSON_data_transfer.py .....	6
2.10.1. record_audio .....	6
2.10.2. ncid_msg_print .....	6
<b>3. C++ Files and Functions .....</b>	<b>7</b>
3.1. main.cpp .....	7
3.1.1. printHex .....	7
3.1.2. serialize .....	7
3.1.3. wait_for_offhook .....	7
3.1.4. record .....	7
3.1.5. main_task .....	7
3.1.6. setup .....	7
<b>4. Validation .....</b>	<b>8</b>

## **List of Tables**

<b>Table 1. Database &amp; Data Processing Validation Checklist .....</b>	<b>8</b>
---	----------

## List of Figures

Figure 1. Database Containing Uploaded Data .....	9
Figure 2. Pre Silence Removal Audio.....	9
Figure 3. Post Silence Removal.....	9
Figure 4. Audio Recorded From Landline Handset.....	10
Figure 5. Audio Recorded From Landline Handset Zoomed In.....	10
Figure 6. Data Sent Through Serial Port As Hex .....	10



## 1. Subsystem Introduction

The database and data processing subsystem is the subsystem that acts as a bridge between the hardware and machine learning subsystems. The database is implemented in MongoDB using python as the language and mongoengine as the MongoDB API interpreter. MongoDB was chosen as this project's database because of its lightweight and document-based architecture. MongoDB's document architecture allows for easy storage of large or complex file types such as audio, as well as nested storage of documents within documents, i.e., allowing for a "voice signature" document alongside a "call recording" document to be stored in a "call information." This simplifies the database implementation for our project as the files we store are large and complex audio files. Our database will have many nested documents within documents to organize voice signature classification data for easy access later.

This subsystem also includes a pre-processing portion where audio has silent portions of the call removed before storage. This has two-fold importance for this project. Not only must we be mindful of the end-users storage limitations by removing unnecessary data from the call to lower file size, but the machine learning algorithm provides more accurate predictions when the speech is continuous. Therefore, by removing silence, we improve the project along the data management axis and analysis of the voice signatures axis.

The subsystem also encompasses recording the analog audio of the callers. It accomplishes this by sampling the incoming analog audio through the ESP32's ADC1. As it samples the analog values, it writes them to an array, removes the dc bias, and multiplies the values to increase the voices' volume.

## 2. Python Files and Functions

This section is documentation of every function made during this project. Each function has an explanation as to what it does.

### 2.1. *Calls.py*

Calls.py contains the class in which all other classes will be nested. This class will then be the object queried by the database for important information that the end user may wish to access. I chose to contain all other classes within this nested class because otherwise, it would be difficult to follow where information is stored. This allows for easier identification and modification of the code by anyone who may wish to do so. As this is an open-source project, I aimed to be mindful of future coders while creating classes.

Within calls.py the class Call() contains:

Variables:

- Date\_of\_call: the date the call was recorded
- Date\_of\_upload: the date the call was uploaded to the database
- Original\_length\_of\_call: the length of the call before silence was removed
- New\_length\_of\_call: the length of the call after silence is removed
- Call\_recording: the embedded document that contains the recording
- File\_name: the name of the file as specified by the user

Functions:

- Upload\_recording: this function takes a folder path and file name and uploads the document at that location to the database.

### 2.2. *Incoming\_callers.py*

Incoming\_callers.py was written this semester as a placeholder for ECEN 404 where I will be integrating this subsystem with the machine learning algorithm. This class will contain the voice signature of the incoming caller. It will allow for easy matching of voice signatures and look up of calls containing the voice signature of a specific person who may have perpetrated a scam.

### 2.3. *Mongoengine\_setup.py*

This file contains setup information for the database. This was implemented as a function for future use; if the database needs to be configured to a different location, it can be called with the new host IP and name of the database. This will cut down on integration time in ECEN 404.

### 2.4. *Receiving\_callers.py*

Receiving\_callers.py was written this semester as a placeholder for ECEN 404 where I will be integrating this subsystem with the machine learning algorithm. This class will contain the voice signature of the receiving caller, allow for easy matching of voice signatures, and look up calls containing the voice signature of a specific person who may have been a victim of a scam.

## ***2.5. Obtain\_call\_length.py***

This file contains three functions, all related to the length of the call from a specific point in time; they are `audio_duration`, `creation_date`, and `obtain_length_of_call`.

### **2.5.1. Audio\_duration**

`Audio_duration` takes one argument, `length`, and converts that given length to a time format that is easily read. This is accomplished by dividing the given time by 3600 and assigning that to hours. Then, divide the remainder of hours by 60 and assign that to minutes. Finally, keep the remainder of that division as the seconds. This function was necessary because when the length of an audio file is found, it is given in seconds, which could be more intuitive for the end user to read.

### **2.5.2. Creation Date**

`Creation_date` takes two arguments, the folder path of the file you wish to find the creation date and the name of the file you wish to find the creation date. The python `os` functions wrapped in the `pathlib` library take the folder path and file name and find a file creation date in seconds since the Unix epoch date was January 1st, 1970 at 00:00:0000 UTC. The function then computes the creation date from the seconds given and converts it into a time in the Central Standard Time Zone to give an accurate creation date for our time zone. The use of `pathlib` also ensures that no matter the end-user's operating system, the proper path will be returned to the function.

### **2.5.3. Obtain\_length\_of\_call**

This function takes one argument, the file location, and obtains the length of the call in seconds before returning the length in a readable format by calling the `audio_duration` function.

## ***2.6. Remove\_silence.py***

This file contains a single function called `remove_silence`. This function takes two arguments, the folder path that contains the required file where silence will be removed and the file name. The function will first read the file in the location given into a variable. Then the file will be segmented into one-second portions of audio that will be analyzed for their 'energy.' Energy in this context is a measure of the amount of speaking in the clip and helps determine whether the section of audio is silent or contains speech that may be important. After determining the energy of every one-second clip, the clips with energy below the threshold will be removed before concatenating the one-second segments together again.

After the segments are concatenated, the function checks if there is a folder for the processed clips to be uploaded. If there is a folder containing the proper naming convention, then the function will upload the processed clips to the folder using the naming convention provided by the user. If there is no folder for the processed clips to be sent to, a folder will be created using the name provided by the user in the same file location as the previous folder. Then the processed clips are uploaded to that folder.

This process ensures that the original audio is only deleted once we are ready to do so.

## **2.7. Upload\_data.py**

Upload\_data.py contains seven functions related to data manipulation and error checking within the database.

### **2.7.1. Retrieve\_recording**

Retrieve\_recording takes four arguments, the file location, the file name in the database, the database name, and the database host ip. This function has been implemented to streamline data retrieval should the project move in the direction of a cloud database. The two variables, database name and database host ip, are not required and default to the local database.

This function returns the first entry in the database that matches the name given in the function call to the location given by the file location in the function call.

### **2.7.2. Delete\_local\_recording**

Delete\_local\_recording takes two arguments, the folder path and the file name. Given these two variables, the function will find the file specified, error-check the user input to make sure that the file exists, warn the user that they are deleting a file and prompt the user for confirmation, and then delete the local file..

### **2.7.3. Delete\_database\_recording**

Delete\_database\_recording takes one argument, the file name in the database. Given the file name in the database, this function will find the file name, error-check the user input against the database to ensure the file exists, warn the user they are deleting a file from the database and prompt the user for confirmation, and then delete the file from the database.

### **2.7.4. Error\_check**

Error\_check is the first of two error-checking functions; this function takes two arguments, folder path and file name. This function was implemented to error-check various naming conventions from our dataset. The dataset had many different naming conventions for recordings, and the error-checking function was designed around checking the most significant number of these folders in our dataset to ensure that files existed within them.

Given a folder path and file name, this function will check to ensure that a file exists within the folder path by using the python os functionality wrapped in the pathlib library. This ensures that you are not trying to upload data from an empty folder and allows the user to be prompted for a different folder directory without exiting the Demo UI used for testing. The use of pathlib ensures that these os path checking functions will be compatible across all platforms despite changes in path naming conventions.

If the file path does not exist, then the function returns a boolean false.

### **2.7.5. Error\_check\_file**

This error-checking function was designed to check if a single file existed. This function will be the error-checking function of choice during integration as we will have complete control over the naming conventions of our files and, therefore, will not need a specialized error-checking function as explained above. The use of pathlib ensures that these os path checking functions will be compatible across all platforms despite changes in path naming conventions.

This function takes two arguments, folder path and file name. The function will then check the folder path to see if the file given exists within the path. If it does not, then the function returns a boolean false.

### **2.7.6. Num\_files**

Num\_files takes one argument, the folder path where the files are stored, and returns the number of files in that folder path. This function uses the pathlib library to ensure that the function will work regardless of the end user's operating system..

### **2.7.7. Upload\_folder**

Upload\_folder is the primary function of this file. Upload\_folder takes three inputs, folder location, write location, and naming convention of the files (i.e., recording1, recording2, recording3, would be input as simply recording). Given the path to the folder you wish to upload and the naming convention of the files within the folder, this function will automatically detect the number of files in the folder and upload them to the database while updating their Call() class variables.

The function starts by error-checking the user's inputs to ensure that the folder contains files and exists. Then the function counts the number of files within the folder path given. After determining the number of files, the function goes through a for loop where it executes the creation\_date, obtain\_length\_of\_call, silence removal, a second obtain\_length\_of\_call, and updates the file name before uploading the file to the database. Each function updates a corresponding variable in the call () class before the upload, ensuring that all the call information is up to date.

The function then uploads a silence-removed version of the audio named by the file's creation date to the filesToTest directory for the machine learning to process.

## **2.8. User\_interface.py**

This file contains the functions that switch between a user's desired actions. When a user inputs a character, the corresponding character in the switch case will be executed. Each of these functions executes a function from the database functions and classes section.

## **2.9. Main.py**

Executes the mongoengine\_setup() function and the user\_interface, run(), which connects the database to the program and start the Demo UI program so that the user can test the functionality of the database classes and functions.

## ***2.10. JSON\_data\_transfer.py***

This file contains the `record_audio` function and the `ncid_message_print` function. This file's objective is to capture the incoming serial data from the serial port and write it to a file with a WAV header so that the machine learning algorithm can use it. Then, it communicates with the NCID platform whether or not a file matches or does not match with a known voice in the database.

### **2.10.1.record\_audio**

This function takes a serial COM port, file location (to be written to), and the audio file's sample rate and writes a wav file at the file location.

### **2.10.2.ncid\_msg\_print**

`Ncid_msg_print` takes a float value as its input and returns a message to the terminal with the confidence value of the machine learning algorithm. This function was originally intended to integrate with the NCID platform through a TCP IP port, however, time constraints and bugs in other functions delayed that integration.

## 3. C++ Files and Functions

### 3.1. *main.cpp*

Main.cpp contains most of the code that makes audio sampling through the ADC possible. It sets up the SD card, calls the ADCSampler code from the lib directory, and then pins the ADCSampler task to an ESP32 CPU core so it is not interrupted. The code then waits until the phone goes off-hook (is picked up and answered) to begin sampling the values from the ADC and writing them to the SD card..

#### 3.1.1. **printHex**

PrintHex takes an array and the length of that array as inputs and then prints out a converted hex value to the serial port.

#### 3.1.2. **serialize**

Serialize takes a file name as text input, opens that file, and reads the data in the file to the serial port as hex data. The .eof() function was not working correctly with the SD card library during testing, so the function waits to read a '/' character to signify that it has reached the end of the file and should stop printing to the serial port.

#### 3.1.3. **wait\_for\_offhook**

Wait\_for\_offhook turns the hardware subsystems off-hook voltage signal into a button to start and stop sampling through the ADC.

#### 3.1.4. **record**

Record allocates memory for the samples array, then starts the ADCSampler task. Following this, it opens a file on the SD card, and then when the signal for ONOFFHOOK\_PIN goes HIGH, it begins sampling the ADC and writing the samples to a file in the .raw audio format. When the ONOFFHOOK\_PIN goes LOW, the function stops recording, closes the file on the SD card, and deallocates the memory on the ESP32 for the array.

#### 3.1.5. **main\_task**

The main\_task function initializes the SD card, the ADCSampler, and the while loop that will 'listen' for the wait\_for\_offhook function to trigger the recording to start. It then calls the function to serialize following the recording ends.

#### 3.1.6. **setup**

Setup initializes the serial port, the ONOFFHOOK\_PIN pinMode, and pins the main\_task function to an ESP32 CPU core to constantly run on one of the cores.

## 4. Validation

The following is this subsystems ECEN 404 validation checklist, along with whether they were complete, or incomplete at time of demo.

**Table 1. Database & Data Processing Validation Checklist**

Retrieve file from database	The file will be in the given or created directory that the user has input	Complete	Scott Kevil-Yeager
UI works as expected, allowing users to input test folder directories	UI works as expected, allowing users to input test folder directories	Complete	Scott Kevil-Yeager
Upload folder	Files in given directory will be counted, processed, named, and uploaded to the database automatically	Complete	Scott Kevil-Yeager
Listen to recording	Properly allows the playback of recording audio through the host machine, this assumes that the host machine will have a speaker	Complete	Scott Kevil-Yeager
Error checking	If a folder directory or file directory is incorrectly given then a message is given and the user is prompted for another input	Complete	Scott Kevil-Yeager
Delete recording in database	Given a valid name the function removes a single entry from the database	Complete	Scott Kevil-Yeager
Delete local recording	If a folder path and file name are given then the function will delete the local file	Complete	Scott Kevil-Yeager
pyAudioAnalysis	Removes periods of silence in recordings to reduce file size	Complete	Scott Kevil-Yeager
Local storage receives recordings	A file can be written to the SD card	Complete	Scott Kevil-Yeager
ESP32 Captures incoming analog audio signal	Samples audio signal using the ESP32 ADC1	Complete	Scott Kevil-Yeager
Handset properly records through ESP32	Audio is sampled and then written to the SD card as a .raw format file	Complete	Scott Kevil-Yeager
Integrate with ML subsystem	Audio files with silence removed are written to filesToTest, and the database is accessible by the ML code	Complete	Scott Kevil-Yeager
Integrate with hardware subsystem	Audio files are recorded when the phone goes off hook, and data is sent to the serial port when the phone is put back on hook	Complete	Scott Kevil-Yeager
Integrate with NCID	Connect using TCPIP to the NCID platform and send messages to it	Incomplete	Scott Kevil-Yeager
Serial communication between microcontroller and database	Receive serial port data in python and write it to a file with a wav file header	Incomplete	Scott Kevil-Yeager



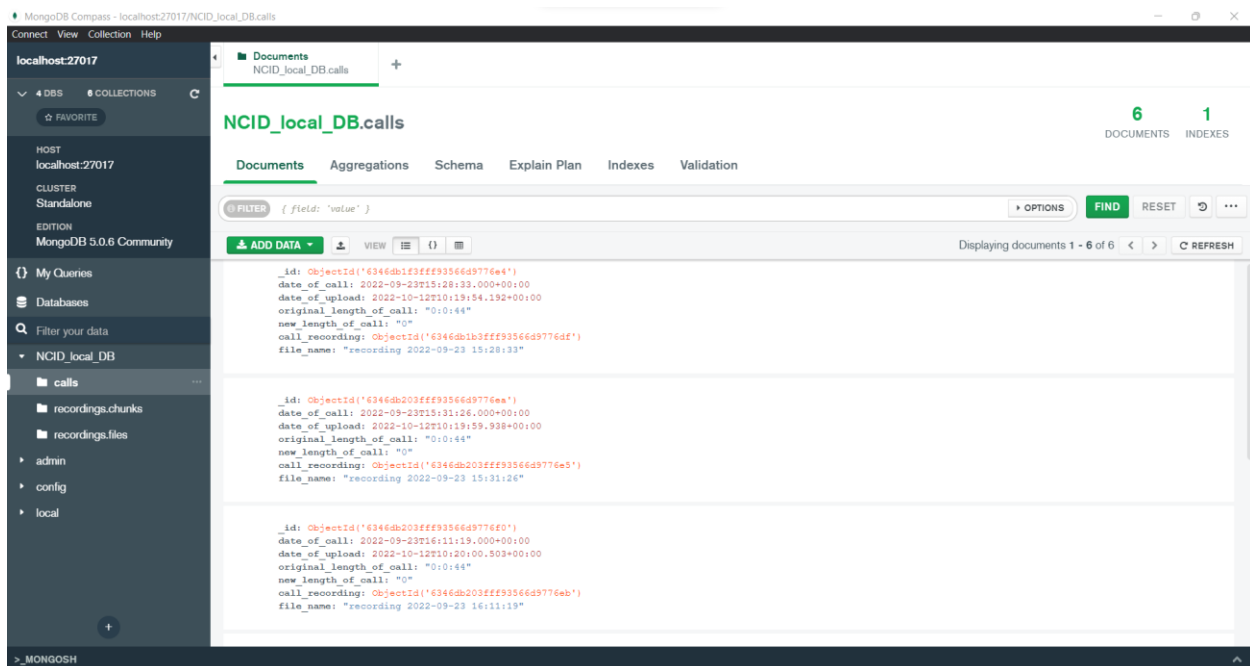


Figure 1. Database Containing Uploaded Data

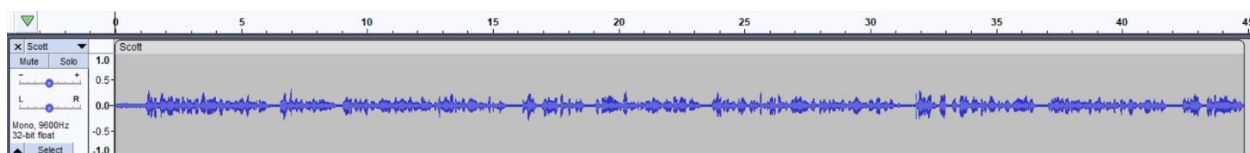


Figure 2. Pre Silence Removal Audio

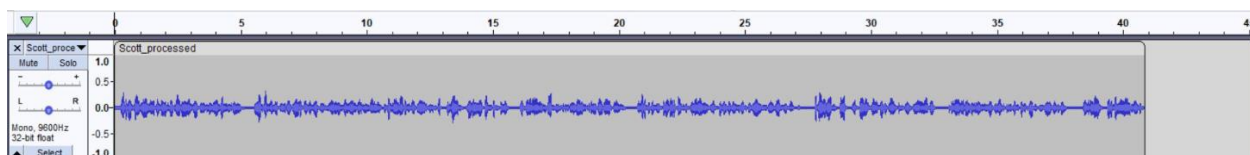


Figure 3. Post Silence Removal



### Figure 6. Data Sent Through Serial Port As Hex

## 5. Additional References

[1] ESP-32 Audio Library from atomic14  
[https://github.com/atomic14/esp32\\_audio](https://github.com/atomic14/esp32_audio)

[2] ESP-32 SD Card Audio from atomic14  
[https://github.com/atomic14/esp32\\_sdcard\\_audio](https://github.com/atomic14/esp32_sdcard_audio)