INL/RPT-24-80481-Revision-0

Reactor System Demonstration with Cyber-Attack Scenarios Using CrowPis and Arduino Microcontrollers

September 2024

Karlie Purser, Michael Clegg, Quentin Kester, Dee Tucker, Palash Kumar Bhowmik, Kenneth Lee Fossum, Piyush Sabharwall

INL/RPT-24-80481-Revision-0

Reactor System Demonstration with Cyber-Attack Scenarios Using CrowPis and Arduino Microcontrollers

Karlie Purser, Michael Clegg, Quentin Kester, Dee Tucker, Palash Kumar Bhowmik, Kenneth Lee Fossum, Piyush Sabharwall

September 2024

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

High School Internship Project Report

[INL/RPT-24-80481-Rev000]

Nuclear Science and Technology

Reactor System Demonstration with Cyber-Attack Scenarios Using CrowPis and Arduino Microcontrollers

JULY 2024

Karlie Purser, Michael Clegg, Quentin Kester, and Dee Tucker

Interns, Thermal Fluid Systems Methods and Analysis, Idaho National Laboratory

Palash Kumar Bhowmik

Mentor, Thermal Fluid Systems Methods and Analysis, Idaho National Laboratory

Kenneth Lee Fossum

Co-Mentor, Thermal Fluid Systems Methods and Analysis, Idaho National Laboratory

Piyush Sabharwall

Manager, Thermal Fluid Systems Methods and Analysis, Idaho National Laboratory

Idaho National Laboratory

Battelle Energy Alliance manages INL for the U.S. Department of Energy's Office of Nuclear Energy

INL/RPT-24-80481-Rev000

High School Internship Project Report

Reactor System Demonstration with Cyber-Attack Scenarios Using CrowPis and Arduino Microcontrollers

Karlie Purser, Michael Clegg, Quentin Kester, and Dee Tucker
Interns, Thermal Fluid Systems Methods and Analysis,
Idaho National Laboratory

Palash Kumar Bhowmik and Kenneth Lee Fossum
Mentors, Thermal Fluid Systems Methods and Analysis,
Idaho National Laboratory

Piyush Sabharwall
Manager, Thermal Fluid Systems Methods and Analysis,
Idaho National Laboratory

July 2024

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Office of Nuclear Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

Page intentionally left blank

SUMMARY

This study covers developing and simulating nuclear reactor system using CrowPis and Arduino microcontrollers for demonstrating cyber-attack scenarios. The team was tasked with implementing more sensors and cybersecurity aspects to the reactor program that was created by last year's high school interns.

The team received the opportunity to collaborate and obtain advice from multiple university interns that helped us gain a better perspective of our project. Our mentor's background in nuclear science was pivotal to our understanding what we could add to the reactor program to make it as realistic as possible.

The first week of our internship was spent reading as much material as possible to gain an understanding of and the background for cyber-attacks and nuclear science. Nuclear science was a new horizon for each of the high school interns on the team, so spending this time in the beginning of our internship was crucial to our success. For the remaining portion of our internship, the team collectively did our best to implement as many sensors and use as much hardware as we could to make an accurate representation of a nuclear reactor in the program that was created.

This internship was a big learning experience for everyone on the team. We all gained so many insights into the nuclear world and how it can benefit our lives, as well as how so many moving pieces are needed for it to work properly.

Page intentionally left blank.

ACKNOWLEDGMENTS

The high school interns would like to collectively thank their mentors, Palash Bhowmik and Kenneth Fossum, for all of their dedication in helping us succeed. They played a vital role in helping us understand nuclear reactors and how we should implement and describe sensors and other additions to our summer internship project. We would also like to thank them for their patience with us, for their tips on our presentation, and for their taking the time to help us understand how we could improve our project as a whole.

We want to send a thank you to our university interns as well, especially Amon Ferrel. His knowledge of Arduino proved invaluable and we sincerely appreciated his help.

We also want to say thank you to our admin, Katrina Lott, for setting us up with everything we needed to be successful. She helped ensure our success because she took the time to make sure we had the proper tools and equipment to do what we set out to do. We would like to thank Piyush Sabharwall, C130 department manager for providing necessary resources and valuable guidance during our internship.

Finally, the high school interns want to express their appreciation to Jennifer Jackson for managing all of us and making this opportunity possible. She truly was the best resource to go to if we ever needed anything.

Page intentionally left blank

CONTENTS

FIGURES

ACRONYMS

2FA	two-factor authentication
AI	artificial intelligence
GUI	graphical user interface
INL	Idaho National Laboratory
IoT	Internet of Things
LCD	liquid-crystal display
LED	light-emitting diode
ML	machine-learning
NFC	near field communication
OLED	organic light-emitting diode
PCTRAN	Personal Computer Transient Analyzer
PIN	personal identification number
RAVEN	Risk Analysis and Virtual ENviroment
RFID	radio frequency identification
UDP	user datagram protocol

Page intentionally left blank

High School Internship Project Report

Reactor Demonstration with Cyber-Attack Scenarios Using CrowPis and Arduino Microcontrollers

1. THE REACTOR

1.1. Introduction and Background

A nuclear reactor operates on the principle of controlled nuclear fission, where the nucleus of an atom, typically uranium-235 (^{235}U), is split into smaller parts by being bombarded with a neutron. This process releases a significant amount of energy in the form of heat, as well as additional neutrons that can go on to split more atomic nuclei [1]. To sustain this reaction at a steady rate, control rods made of materials that absorb neutrons, like boron or cadmium, are inserted into the reactor core to regulate the number of free neutrons [2]. The heat generated by fission is carried away from the reactor core by a coolant, such as water, which can then be used to produce steam. This steam is transferred into a power plant where the steam drives turbines connected to generators, producing electricity. The entire process is contained within a reinforced structure to ensure safety and control, with multiple safety systems in place to prevent the release of radiation and to shut down the reactor in case of an emergency.

The high school interns from last summer's internship had programmed the initial code for a nuclear reactor on a CrowPi [3]. As shown in Figure 1, a CrowPi system typically is run using a Raspberry Pi, but it has 22 sensors, a programable liquid-crystal display (LCD) module, and multiple light-emitting diode (LED) modules. Along with the physical sensors on the machine, there is also a monitor attached so programming can be done directly on the CrowPi. The program developed by last year's interns was for a full reactor cycle, including a SCRAM procedure. This procedure was designed to shut-down the reactor due to xenon poisoning.

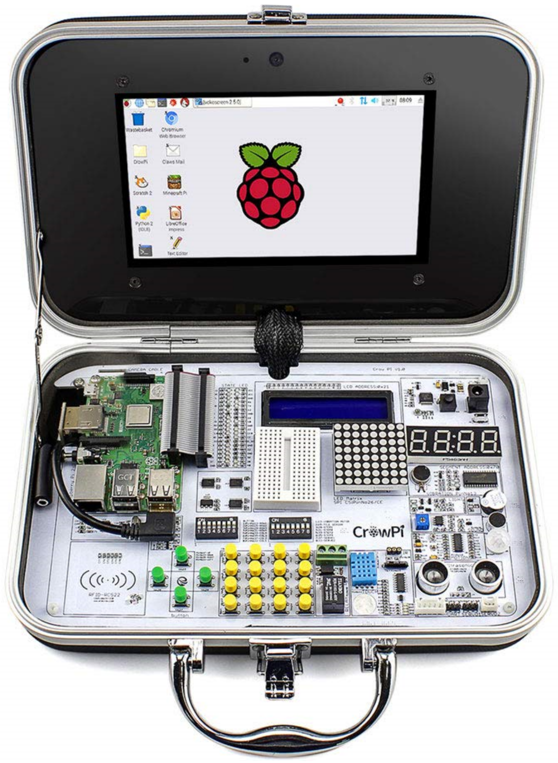


Figure 1. CrowPi system.

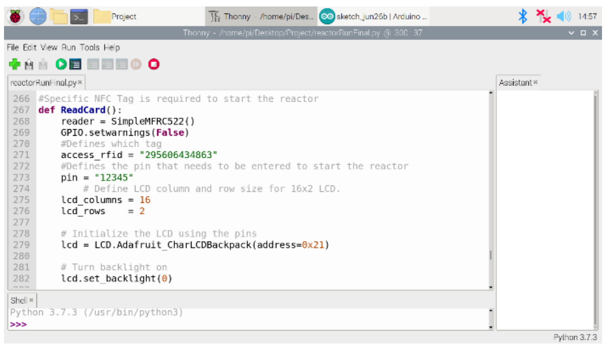
1.2. Objective

Our team in this year's internship was tasked with adding additional sensors to the reactor using various Arduino microcontrollers. The team also was asked to develop cyber-scenarios to demonstrate different levels of attacks that can occur while operating a reactor and a power plant side by side. Previous related studies were reviewed such as noise analysis for nuclear signals [4], Arduino microcontroller for control function implementation [5], cybersecurity for reactor system non-safety functions [6], defensive cyber security architecture [7], control system for malicious code detection [8], cyber security for digital instrument and control supply chain [9], and diagnostic algorithm for cyber-attack analysis in nuclear power plant [10]. Finally, our team was asked to develop a way to connect all of the hardware we would be using though a Wi-Fi access point.

1.3. Methodology and Design

1.3.1. Restructuring The Reactor

The reactor code developed by last year's interns was used as a baseline for what had to be done this year. The team first started off by implementing a cybersecurity measure. A two-factor authentication (2FA) mechanism was developed and employed, which required the user to use a near field communication (NFC) tag on a radio frequency identification (RFID) sensor set up on the CrowPI. As shown in Figure 2 and Figure 3, if the correct tag was not being used, the reactor could not be started. Once the correct tag was employed, however, the user still had to know the personal identification number (PIN) to gain access, as observed in Figure 4.

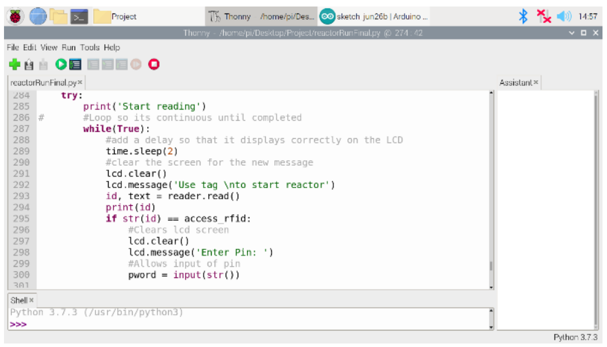


```
#Specific NFC Tag is required to start the reactor
def ReadCard():
    reader = SimpleMFRC522()
    GPIO.setwarnings(False)
    #Defines which tag
    access_rfid = "295606434863"
    #Defines the pin that needs to be entered to start the reactor
    pin = "12345"
        # Define LCD column and row size for 16x2 LCD.
    lcd_columns = 16
    lcd_rows    = 2

    # Initialize the LCD using the pins
    lcd = LCD.Adafruit_CharLCDBackpack(address=0x21)

    # Turn backlight on
    lcd.set_backlight(0)
```

```
Python 3.7.3 (/usr/bin/python3)
>>>
```

Project
Thonny - /home/pi/Des...
sketch_jun26b | Arduino ...
14:57
Thonny - /home/pi/Desktop/Project/reactorRunFinal.py @ 300 : 37
File Edit View Run Tools Help
reactorRunFinal.py
Assistant
Shell
Python 3.7.3

Figure 2. This code identifies the NFC tag and PIN.

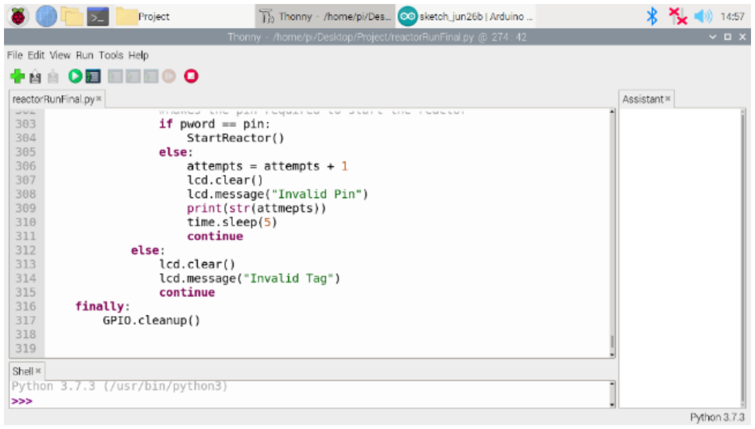


```
    try:
        print('Start reading')
#       #Loop so its continuous until completed
        while(True):
            #add a delay so that it displays correctly on the LCD
            time.sleep(2)
            #clear the screen for the new message
            lcd.clear()
            lcd.message('Use tag \nto start reactor')
            id, text = reader.read()
            print(id)
            if str(id) == access_rfid:
                #Clears lcd screen
                lcd.clear()
                lcd.message('Enter Pin: ')
                #Allows input of pin
                pword = input(str())
```

```
Python 3.7.3 (/usr/bin/python3)
>>>
```

Project
Thonny - /home/pi/Des...
sketch_jun26b | Arduino ...
14:57
Thonny - /home/pi/Desktop/Project/reactorRunFinal.py @ 274 : 42
File Edit View Run Tools Help
reactorRunFinal.py
Assistant
Shell
Python 3.7.3

Figure 3. This code reads the NFC tag and PIN.



```
                if pword == pin:
                    StartReactor()
                else:
                    attempts = attempts + 1
                    lcd.clear()
                    lcd.message("Invalid Pin")
                    print(str(attempts))
                    time.sleep(5)
                    continue
            else:
                lcd.clear()
                lcd.message("Invalid Tag")
                continue
    finally:
        GPIO.cleanup()
```

```
Python 3.7.3 (/usr/bin/python3)
>>>
```

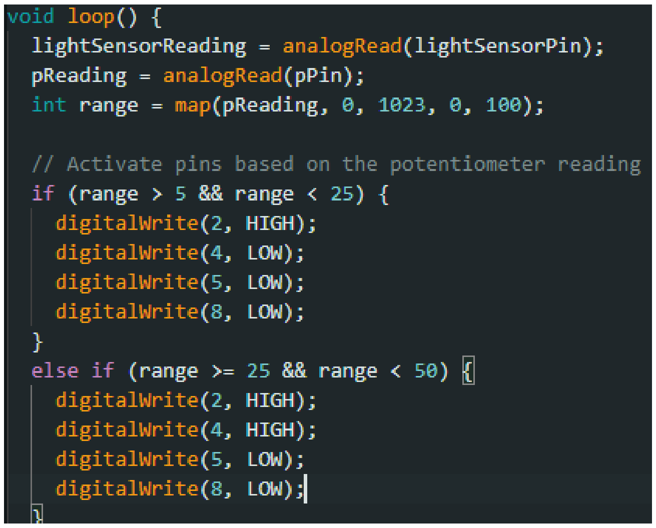
Project
Thonny - /home/pi/Des...
sketch_jun26b | Arduino ...
14:57
Thonny - /home/pi/Desktop/Project/reactorRunFinal.py @ 274 : 42
File Edit View Run Tools Help
reactorRunFinal.py
Assistant
Shell
Python 3.7.3

Figure 4. This code starts the reactor if the NFC tag and PIN match.

The team had the goal of making everything more realistic by using Arduino microcontrollers to implement more sensors. We had much more hardware to work with as a result. The goal was to make the reactor autonomous. But in order to make it autonomous, we had to make the CrowPI synonymous with the Arduinos microcontrollers. On CrowPI itself, a graphical user interface (GUI) had to be designed and developed so that it would take real-time data from the Arduinos and display it on the GUI. By making the reactor autonomous, it would be possible to simulate cyber-attacks because our data was being sent through a Wi-Fi signal. Once we spoofed our data, it was easy to see that code being displayed on the GUI vs. what was happening in the reactor. The idea with using false data is that any readings being seen would be either higher or lower so the reactor control officer would, in theory, have to either turn the power up or down since it is already operating.

1.3.1. Adding Sensors Using Arduino Microcontrollers

A variety of sensors were employed with the Arduino to help simulate the reactor. One of these was a phototransistor, which is a light sensor. This light sensor was used in conjunction with a few other components, including a potentiometer and four LEDs, as depicted in Figure 5 and Figure 6 to simulate a temperature sensor inside a nuclear reactor. The potentiometer is a small knob that is used to simulate input into the nuclear reactor to change the power levels. The four LEDs represent the amount of power inside the reactor. As the knob is turned, which represents the operator changing the power, the LEDs turn on in sequence so that when it is fully on, all LEDs are on, and when it is only halfway turned, only two of the LEDs are on. Then, it is possible to see the values of the potentiometer turning on the LEDs corresponding to the light sensor values in the Serial Plotter. After that, another Arduino can connect to it to spoof the data and represent a cyber-attack.



```
void loop() {
  lightSensorReading = analogRead(lightSensorPin);
  pReading = analogRead(pPin);
  int range = map(pReading, 0, 1023, 0, 100);

  // Activate pins based on the potentiometer reading
  if (range > 5 && range < 25) {
    digitalWrite(2, HIGH);
    digitalWrite(4, LOW);
    digitalWrite(5, LOW);
    digitalWrite(8, LOW);
  }
  else if (range >= 25 && range < 50) {
    digitalWrite(2, HIGH);
    digitalWrite(4, HIGH);
    digitalWrite(5, LOW);
    digitalWrite(8, LOW);
  }
```

Figure 5. Code that uses the potentiometer to turn on the LEDs and monitors with the light sensor.

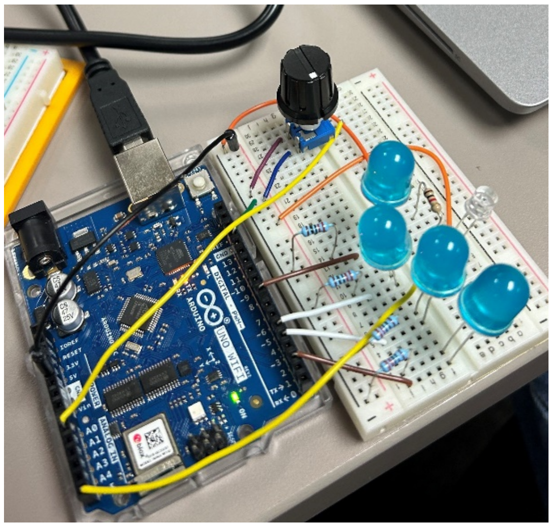


Figure 6. The LEDs, phototransistor, and potentiometer used in this setup.

1.3.2. Interface For The Plant

The plant is where the heat from the reactor gets turned into electricity. The team wanted to have an interface to display some of the numbers being provided from the sensors to help us see that process. That is where the Arduino MKR Wi-Fi 1010, as shown in Figure 7, and the MKR Internet of Things (IoT) Carrier REV2, as observed in Figure 8, came in. Both were used hand-in-hand to develop an interface. Really, the MKR Wi-Fi 1010 board is the brain behind the whole process, which is where the script is uploaded to and what stores the script, as well as providing the main power source. The IoT carrier has humidity and temperature sensors carried within it. This board also has five programable buttons and lights. These buttons were used to control the organic light-emitting diode (OLED) screen. Buttons one and three controlled the three menu options that were displayed on the OLED screen. The first option said, “Schaechner,” the second, “Weather,” while the third read, “Settings.” The Weather option can toggle though a display of temperature and humidity. The data being delivered is provided by the sensors and updates every minute. The setting screen controls the LEDs, which can dim them, turn them off, and then back on again using button zero.

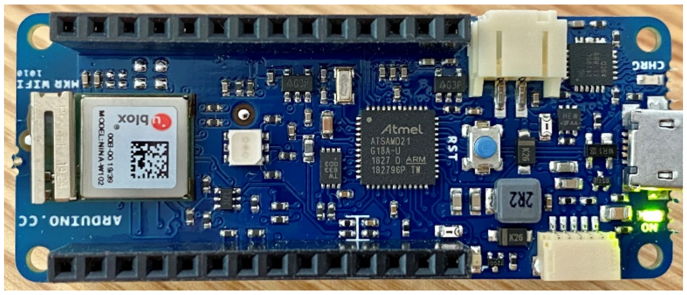


Figure 7. MKR Wi-Fi 1010.

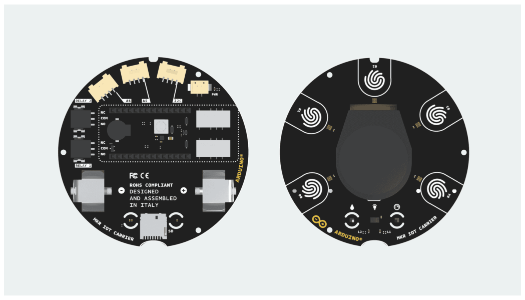
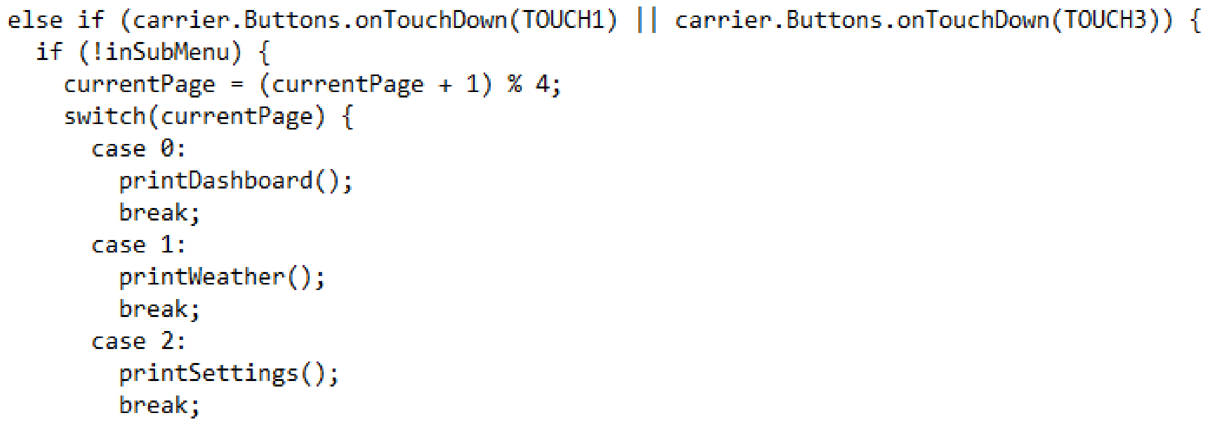


Figure 8. MKR IoT Carrier REV2.

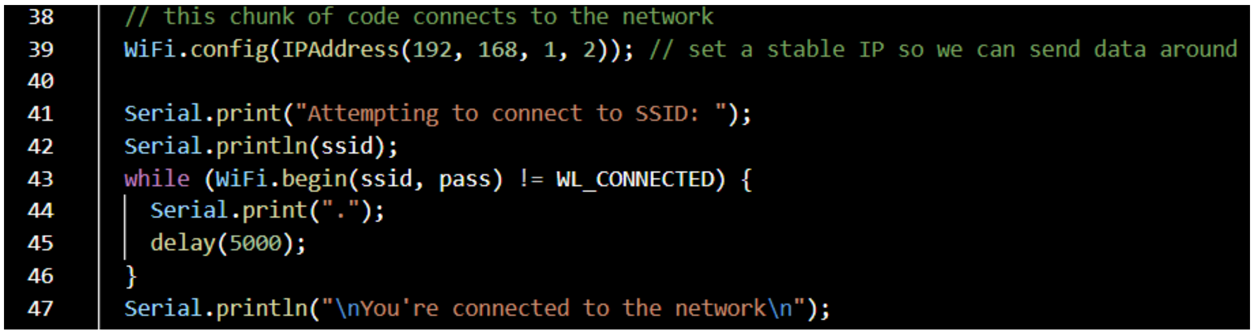
1.3.1. Developing a Wi-Fi Access Point

The team decided to develop a Wi-Fi access point to allow for easy sensor connectivity, as well as the easy expansion of additional sensors as this project grows. We used the Arduino Uno Wi-Fi Rev. 2, which includes a Wi-Fi chip built directly into the board. These Arduino boards use the WiFiNINA library to control network connectivity. Using this library was simple after a short learning period. WiFiNINA has modules to host and connect to Wi-Fi access points, both of which were used in this work. WiFiNINA also has modules that send and receive user datagram protocol (UDP) data over Wi-Fi, which would prove essential for communication between devices. The Raspberry Pi runs a version of Linux called Raspbian that includes a GUI, thus making connecting it to the network very simple but difficult to automate. Knowing this in advance, the team decided to stick to hosting the network on the Arduino and manually connecting the Raspberry Pi to the network. Making this manual connection just once meant that it would automatically connect to the Arduino network whenever it was hosted. Our current project uses two Arduinos and a Raspberry Pi. One Arduino, which also controls the cyber-attacks, hosted the network, while the other housed the light sensor and LEDs that were connected to the network. Figure 9 provides a sample of the script that was used for the plant interface. Figure 10, Figure 11, and Figure 12 show sample code that was used to connect to a WiFi network, to create a new WiFi network, and to send UDP messages, respectively. The actual code is provided in Appendix A.



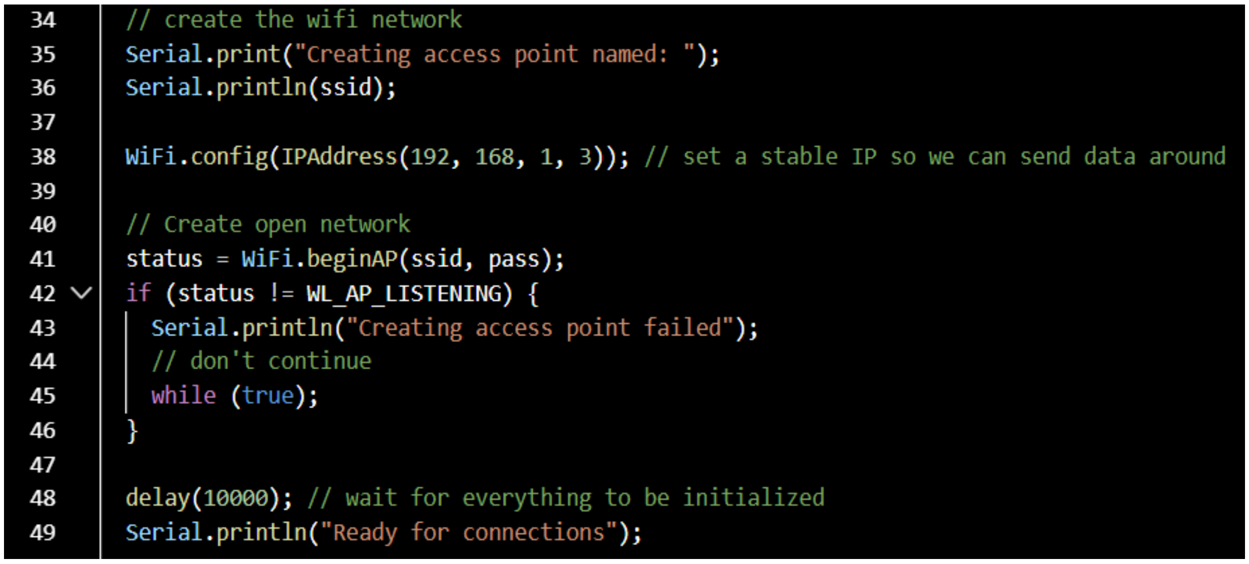
```
else if (carrier.Buttons.onTouchDown(TOUCH1) || carrier.Buttons.onTouchDown(TOUCH3)) {
  if (!inSubMenu) {
    currentPage = (currentPage + 1) % 4;
    switch(currentPage) {
      case 0:
        printDashboard();
        break;
      case 1:
        printWeather();
        break;
      case 2:
        printSettings();
        break;
```

Figure 9. Part of the script for the plant interface.



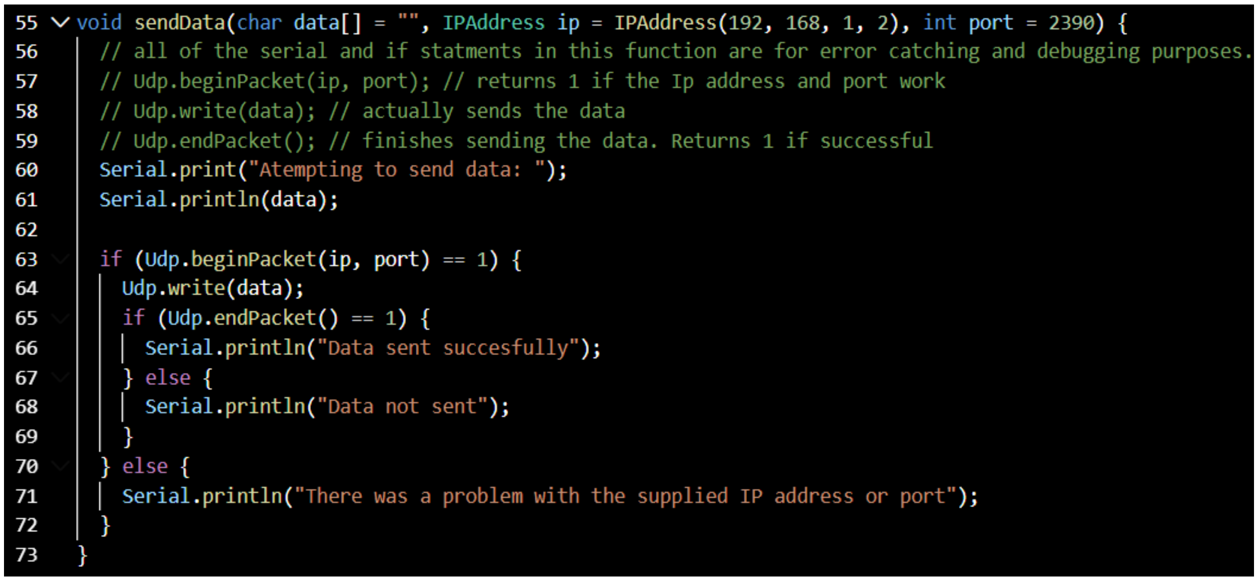
```
// this chunk of code connects to the network
WiFi.config(IPAddress(192, 168, 1, 2)); // set a stable IP so we can send data around

Serial.print("Attempting to connect to SSID: ");
Serial.println(ssid);
while (WiFi.begin(ssid, pass) != WL_CONNECTED) {
  Serial.print(".");
  delay(5000);
}
Serial.println("\nYou're connected to the network\n");
```

Figure 10. Arduino code to connect to a Wi-Fi network.



```
// create the wifi network
Serial.print("Creating access point named: ");
Serial.println(ssid);

WiFi.config(IPAddress(192, 168, 1, 3)); // set a stable IP so we can send data around

// Create open network
status = WiFi.beginAP(ssid, pass);
if (status != WL_AP_LISTENING) {
  Serial.println("Creating access point failed");
  // don't continue
  while (true);
}

delay(10000); // wait for everything to be initialized
Serial.println("Ready for connections");
```

Figure 11. Arduino code to create a Wi-Fi network.



```
void sendData(char data[] = "", IPAddress ip = IPAddress(192, 168, 1, 2), int port = 2390) {
  // all of the serial and if statments in this function are for error catching and debugging purposes.
  // Udp.beginPacket(ip, port); // returns 1 if the Ip address and port work
  // Udp.write(data); // actually sends the data
  // Udp.endPacket(); // finishes sending the data. Returns 1 if successful
  Serial.print("Atempting to send data: ");
  Serial.println(data);

  if (Udp.beginPacket(ip, port) == 1) {
    Udp.write(data);
    if (Udp.endPacket() == 1) {
      Serial.println("Data sent succesfully");
    } else {
      Serial.println("Data not sent");
    }
  } else {
    Serial.println("There was a problem with the supplied IP address or port");
  }
}
```

Figure 12. Arduino code to send UDP messages.

1.3.2. Cyber Security Set up and Scenarios

Several cyber-attack scenarios were created by the team to demonstrate how an attack would play out. Every attack is different but there are only so many components that an attacker can hack. This led to our development of four varying levels of attacks, as depicted in the block chart shown in Figure 13. Level 1 would occur when someone could infiltrate the operating system but cannot do anything beyond that. Only the information residing on the system would be vulnerable. For this level, our simulations would not show the hackers doing anything, but we would be able to monitor them and try to eventually boot them out of the system. Level 2 would happen when a hacker could mess with a plant subsystem, such as changing the amount of cooling water available for the plant. To counter this attack, it would become necessary to reduce power to the reactor. In a Level 3 attack, the hacker would be able to interfere with the main functions of the plant, such as a generator or turbine. For this type of attack, it was determined that the plant would have to be shut down and power to the reactor would need to be reduced to compensate. The most serious type of attack is Level 4, which is where a hacker would be able to compromise the actual reactor, such as influencing the control rods or the control drum. This would call for a complete reactor shutdown called a SCRAM.

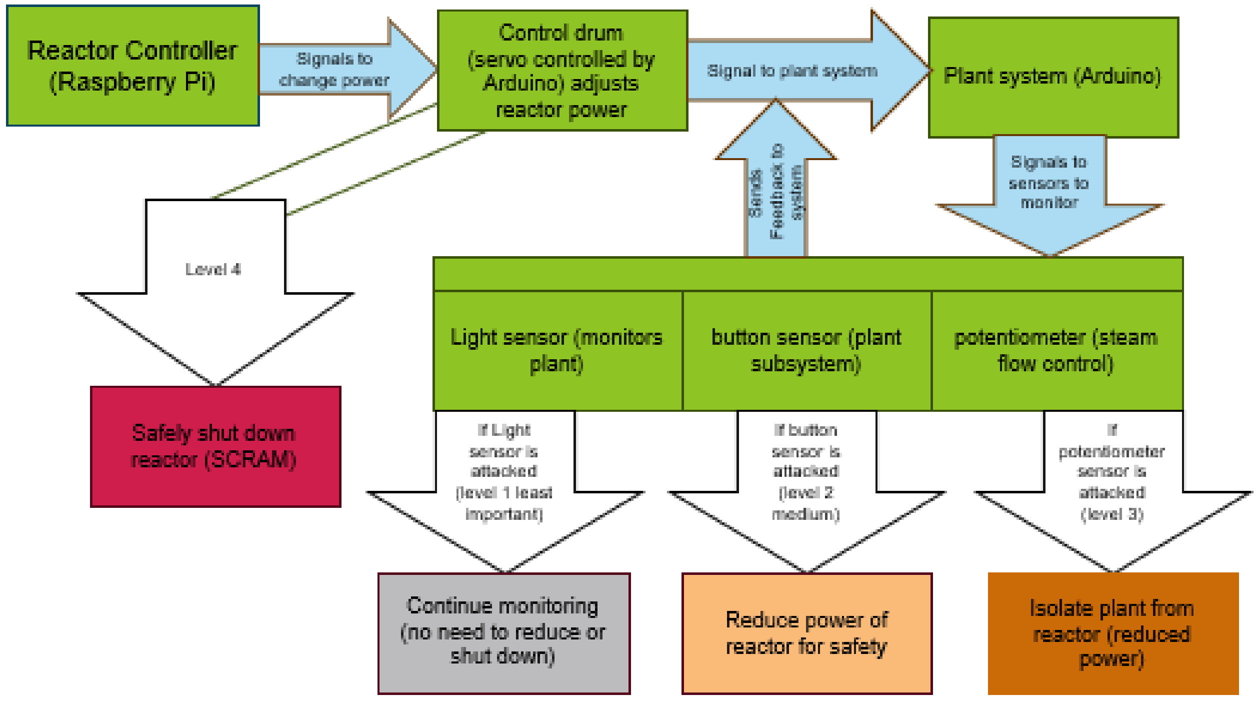

Reactor Controller (Raspberry Pi)
Signals to change power
Control drum (servo controlled by Arduino) adjusts reactor power
Signal to plant system
Plant system (Arduino)
Sends Feedback to system
Signals to sensors to monitor
Level 4
Safely shut down reactor (SCRAM)
Light sensor (monitors plant)
button sensor (plant subsystem)
potentiometer (steam flow control)
If Light sensor is attacked (level 1 least important)
If button sensor is attacked (level 2 medium)
If potentiometer sensor is attacked (level 3)
Continue monitoring (no need to reduce or shut down)
Reduce power of reactor for safety
Isolate plant from reactor (reduced power)

Figure 13. Block chart of the reactor and the cyber-levels.

1.4. Discussion

The overall goal for the project this summer was to improve upon what last year's interns had completed with a details cyber-attack demonstration. It was also important for us to provide next year's interns the opportunity to improve upon what we were able to do. Our improvements included implementing 2FA, adding Arduino microcontrollers to use as sensors, and using another Raspberry Pi as a web server. Project scop extended to integrate programable logic controller (PLC) units. The team also started developing a GUI to display information. An interface is already in place via the CrowPI LCD screen, but we wanted to develop a different one that read data directly from the Arduinos and allows the ability to simulate cyber-attacks.

1.5. Conclusion

This project has been an amazing learning experience for each member of the intern team. We each learned how to use an Arduino and Raspberry Pi. Furthermore, this has been a real mind-bending project. It allowed us to turn on our creativity regarding how we can use different sensors to represent different things. It was interesting to put our cybersecurity and computer science backgrounds towards the nuclear field. This nuclear reactor will continue to progress through other internship groups throughout the years. However, we believe that we have set a great foundation for future interns to build from.

2. INTERNSHIP TAKEAWAYS

Although our project was our primary focus as interns, we had more interaction with Idaho National Laboratory (INL) personnel than just from our mentors. Our primary mentor gave us a multitude of opportunities to see all that nuclear has to offer and to explore some of the groundbreaking software INL is using in nuclear engineering research. We took every opportunity we could to network and learn what INL and nuclear energy is all about.

Our objective for this internship was to learn as much as possible. We Each of us learned everything we could about nuclear, the INL itself, and all of the skills we felt would be needed to pull off this project. We all had to learn things we had little to no experience in. We took the opportunity of learning to better accumulate our skills and further spark our interests.

The methodology for us was to not only keep an open mind to learning as much as possible but to build valuable connections as well. Networking and connections are nearly as important as learning for building opportunities. The following sections describe the additional learning opportunities we received during our internships.

2.1. Health Fair and AI/ML Tech Expo

One of the best learning and networking opportunities for us was at the Health Fair and Tech Expo. We had the opportunity to network with so many different types of people from across INL. One of the booths at the Tech Expo used heavy encryption to give data to people that did not have authority to see it. The way this was done was by sending data through a picture, such as one of a litter of kittens. If the kittens were gray, the data was good; whereas if they were orange, it would be bad. It was cool to see cybersecurity in action and the creativity that is allowed at the lab.

Another thing that was very interesting at the artificial intelligence (AI) Expo was a booth about fire recognition on camera feed with AI. The program had been trained through machine-learning (ML) so that it could monitor cameras and look for flames to alert if there was a fire. This could greatly increase safety in places where security cameras are already in place and this software could then just be implemented.

2.2. Hardware and Programming Experiences

This internship introduced all of us to a lot of new things, such as the opportunity to explore nuclear reactors and Arduino microcontrollers. The connections of Arduinos and Raspberry Pis also was an invaluable experience. Learning and exploring new things like this was a box that got checked over and over again during our internship.

2.3. Relevant Trainings and Invited Research Talks

Over the course of our internship, we had the opportunity to attend various training courses. These were great opportunities to learn more about various subjects. One of these was a Risk Analysis and Virtual ENviroment (RAVEN) training by Congjian Wang, who taught us more about how RAVEN works and how it can be used for our various project applications. Our project did not necessitate the use of RAVEN; however, it was very interesting to learn about. Another training we were able to attend was about the Personal Computer Transient Analyzer (PCTRAN) by Hossam Abdellatif. PCTRAN is a nuclear reactor simulation software where values can be adjusted that lead to results on different things. Another meeting we got to attend was a talk about nuclear reactor cybersecurity by Lon Dawson.

2.4. Summary

We had tremendous opportunities to learn and interact with faculty all over the laboratory. Everything that we have been introduced to and have been exposed to has been crucial to our futures. The skills we have learned to network and to advocate for ourselves during this internship are some of our greatest takeaways. It has been so rewarding to hear everyone's stories about how they made their careers. It has given us insight into how we are going to take our own paths into our own careers. We have been given opportunities to do multiple training courses and hear from multiple experts about everything from nuclear to cybersecurity. To conclude, this has been an amazing learning experience with everything we have been able to touch and learn from.

3. REFERENCES

1. Nuclear 101: How does a nuclear reactor work? The United State Department of Energy. (Accessed on July 2024). https://wwwenergy.gov/ne/articles/nuclear-101-how-does-nuclear-reactor-work.
2. Stanley, C. J., & Marshall, F. M. (2008, January). Advanced test reactor: A national scientific user facility. In *International Conference on Nuclear Engineering* (Vol. 48175, pp. 367-372).
3. Allegood, Brecken, and Arianne Fels. "High School Internship Research Report." *Idaho National Laboratory*. Idaho National Laboratory, n.d. http://www.inl.gov.
4. "An Online FDI Detection Scheme Based on Noise Analysis for Nuclear Signals." *School of Nuclear Engineering, Purdue University*, 2021. https://doi.org/10.13182/T130-44813.
5. Arduino Forum. "A Code for MKR IoT Carrier Rev2 - Please Help!," March 31, 2023. https://forum.arduino.cc/t/a-code-for-mkr-iot-carrier-rev2-please-help/1109215.
6. Kim, Taejin, Young-Jun Lee, Inhye Hahm, and Korea Atomic Energy Research Institute. "A Dataset to Support Cybersecurity Research for Non-Safety Controller of APR1400." *Dataset Collection Results*, n.d. https://doi.org/10.13182/T130-44120.
7. Maccarone, Lee T., Andrew S. Hahn, Michael T. Rowland, and Sandia National Laboratories. "Design of Defensive Cyber Security Architectures Using Event Trees." *Sandia National Laboratories*. Sandia National Laboratories, 2021. https://doi.org/10.13182/T130-44645.
8. Park, Poe Il, Kookheui Kwon, and Korea Institute of Nuclear Non-proliferation And Control (KINAC). "Control System Targeted Malicious Codes Detection through Configuration Management," 2021.
9. Park, Seunghoon, Jr., Kookheui Kwon, Poe Il Park, and Korea Institute of Nuclear nonproliferation and Control. "Requirement for Security Verification and Supply Chain Control of Digital Commercial Off-The-Shelf for Cybersecurity of Nuclear Facilities." *Korea Institute of Nuclear Nonproliferation and Control*, 2021. https://doi.org/10.13182/T130-44697.
10. Patel, Japan K., Athi Varuttamaseni, Youngblood Iii Robert W, Junjie Guo, Steven Wacker, Rafael Pires Barbosa, and John C. Lee. "Diagnostics Algorithms in Nuclear Plant Cyber Attack Analysis Toolkit." *arXiv (Cornell University)*, 2021. https://doi.org/10.48550/arxiv.2311.17936.

Appendix A

Reactor and Plant Codes

A-1. Plant Interface

```
#include <Arduino_MKRIoTCarrier.h>
#include <FastLED.h>
MKRIoTCarrier carrier;

float temperature = 0;
float humidity = 0;
unsigned long lastTempUpdate = 0;
int currentPage = 0;
bool inSubMenu = false;
int subMenuPage = 0;
bool ledState = false;
int brightness = 0;

void printDashboard() {
carrier.display.fillScreen(ST77XX_BLACK);
carrier.display.setTextColor(ST77XX_WHITE);
carrier.display.setTextSize(2);
int16_t x, y;
uint16_t width, height;
carrier.display.getTextBounds("SCHAECHNER", 0, 0, &x, &y, &width, &height);
carrier.display.setCursor((carrier.display.width() - width) / 2, (carrier.display.height() - height) / 2);
carrier.display.print("SCHAECHNER");
}

void printTemperature() {
carrier.display.fillScreen(ST77XX_BLUE);
carrier.display.setTextColor(ST77XX_WHITE);
carrier.display.setTextSize(2);
int16_t x, y;
uint16_t width, height;
```

```
String tempStr = "Temp: " + String(temperature) + " C";
carrier.display.getTextBounds(tempStr, 0, 0, &x, &y, &width, &height);
carrier.display.setCursor((carrier.display.width() - width) / 2, (carrier.display.height() - height) / 2);
carrier.display.print(tempStr);
}

void printHumidity() {
carrier.display.fillScreen(ST77XX_GREEN);
carrier.display.setTextColor(ST77XX_WHITE);
carrier.display.setTextSize(2);
int16_t x, y;
uint16_t width, height;
String humidityStr = "Humidity: " + String(humidity) + " %";
carrier.display.getTextBounds(humidityStr, 0, 0, &x, &y, &width, &height);
carrier.display.setCursor((carrier.display.width() - width) / 2, (carrier.display.height() - height) / 2);
carrier.display.print(humidityStr);
}

void printWeather() {
carrier.display.fillScreen(ST77XX_YELLOW);
carrier.display.setTextColor(ST77XX_BLACK);
carrier.display.setTextSize(2);
int16_t x, y;
uint16_t width, height;
carrier.display.getTextBounds("Weather", 0, 0, &x, &y, &width, &height);
carrier.display.setCursor((carrier.display.width() - width) / 2, (carrier.display.height() - height) / 2);
carrier.display.print("Weather");
}


void printSettings() {
carrier.display.fillScreen(ST77XX_MAGENTA);
carrier.display.setTextColor(ST77XX_BLACK);
carrier.display.setTextSize(2);
int16_t x, y;
```

```
uint16_t width, height;
carrier.display.getTextBounds("Settings", 0, 0, &x, &y, &width, &height);
carrier.display.setCursor((carrier.display.width() - width) / 2, (carrier.display.height() - height) / 2);
carrier.display.print("Settings");
}

void printLight() {
  carrier.display.fillScreen(ST77XX_ORANGE);
  carrier.display.setTextColor(ST77XX_BLACK);
  carrier.display.setTextSize(2);
  int16_t x, y;
  uint16_t width, height;
  String text = "LED: ";

  // Check if button 0 is pressed
  if (digitalRead(0) == HIGH) {
    // Increase brightness by 10% until it reaches 100%
    brightness += 10;
    if (brightness > 100) {
      // Reset brightness to 0%
      brightness = 0;
      ledState = false;
    } else {
      ledState = true;
    }
  } else {
    ledState = false; // Turn off the LED if button 0 is not pressed
  }

  if (ledState) {
    text += "ON";
  } else {
    text += "OFF";
  }
  carrier.display.getTextBounds(text, 0, 0, &x, &y, &width, &height);

  carrier.display.setCursor((carrier.display.width() - width) / 2, (carrier.display.height() - height) / 2);

  carrier.display.print(text);

  if (ledState) {
    // Set the LED brightness according to the global variable
    carrier.leds.fill(CRGB::White);
    carrier.leds.setBrightness(brightness);
    carrier.leds.show();

  } else {
    // Set the LED color to black
    carrier.leds.fill(CRGB::Black);
    carrier.leds.show();

  }
}

void setup() {
  CARRIER_CASE = false;
  carrier.begin();
  carrier.display.setRotation(0);
  temperature = carrier.Env.readTemperature();
  humidity = carrier.Env.readHumidity();
}

void loop() {

if (carrier.Buttons.onTouchDown(TOUCH0)) {
    if (inSubMenu && currentPage == 2 && subMenuPage == 0) {
      ledState = !ledState;
      printLight();
    }
    else if (currentPage == 2) {
      inSubMenu = true;
      subMenuPage = 0;
      printLight();
    }
  }

  carrier.Buttons.update();
  unsigned long now = millis();

  if(now - lastTempUpdate > 60000){
    temperature = carrier.Env.readTemperature();
    humidity = carrier.Env.readHumidity();
    lastTempUpdate = now;
  }

  if (carrier.Buttons.onTouchDown(TOUCH2)) {
    currentPage = 0;
    inSubMenu = false;
    printDashboard();
  }
  else if (carrier.Buttons.onTouchDown(TOUCH1) || carrier.Buttons.onTouchDown(TOUCH3)) {
    if (!inSubMenu) {
      currentPage = (currentPage + 1) % 4;
      switch(currentPage) {
        case 0:
          printDashboard();
          break;
        case 1:
          printWeather();
          break;
        case 2:
          printSettings();
          break;
      }
    }
    else {
      subMenuPage = (subMenuPage + 1) % 2;
      switch(currentPage) {
        case 1:
          switch(subMenuPage) {
            case 0:
              printTemperature();
              break;
            case 1:
              printHumidity();
              break;
          }
          break;
        case 2:
          switch(subMenuPage) {
            case 0:
              printLight();
              break;
            case 1:
              printTemperature();
              break;
          }
          break;
      }
    }
  }
  else if (carrier.Buttons.onTouchDown(TOUCH4)) {
    if ((currentPage == 1 || currentPage ==2) && !inSubMenu) {
      inSubMenu = true;
      subMenuPage = (subMenuPage + 1) %2;
      switch(currentPage) {
        case 1:
          switch(subMenuPage) {
            case 0:
              printTemperature();
              break;
            case 1:
              printHumidity();
              break;
          }
          break;
        case 2:
          switch(subMenuPage) {
            case 0:
              printLight();
              break;
            case 1:
              printTemperature();
              break;
          }
          break;
      }
    }
    else if (inSubMenu) {
      inSubMenu = false;
      switch(currentPage) {
        case 1:
          printWeather();
          break;
        case 2:
          printSettings();
          break;
      }
    }
  }}
```

A-2. Reactor

```
from Adafruit_LED_Backpack import SevenSegment
import Adafruit_CharLCD as LCD
import re
import time
import RPi.GPIO as GPIO
from luma.led_matrix.device import max7219
from luma.core.interface.serial import spi, noop
from luma.core.render import canvas
from luma.core.virtual import viewport
from luma.core.legacy import text, show_message
from luma.core.legacy.font import proportional, CP437_FONT, TINY_FONT, SINCLAIR_FONT,
LCD_FONT
# from mfrc522 import SimpleMFRC522
import subprocess


#Set up gpio mode once
GPIO.setmode(GPIO.BCM)

# declare variables to store WiFi information
ssid = "MyAccessPoint"
password = "1234567890"

# Subroutine for the initial print LCD statement
def InitializeReactor():

  # Define LCD column and row size for 16x2 LCD.
  lcd_columns = 16
  lcd_rows    = 2

  # Initialize the LCD using the pins
  lcd = LCD.Adafruit_CharLCDBackpack(address=0x21)

  # Turn backlight on
  lcd.set_backlight(0)

  # Print a two line message
  lcd.message('Press top button \nto start reactor')
```

```
    # Creating variables by their BCM number to call later on
    button_pin = 26

    # Setting command mode to BCM
    GPIO.setmode(GPIO.BCM)

    # Setup button pin as INPUt (last command turns digital code into analog output)
    GPIO.setup(button_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)

    # Allows button to only have to pressed and then released to run loop
    not_push = True

    while(not_push):

        if(GPIO.input(button_pin) == 0):
            not_push = False

    # Turn off back light and clear message once button pressed
    lcd.clear()
    lcd.set_backlight(1)

# Subroutine that counts down the the reactor start
# Has a parameter count which allows for the user to pick how long the countdown is (max 99s)
def CountdownToReactor(count):

    buzzer_pin = 18
    vibration_pin = 27

    # Setting command mode to BCM
    GPIO.setmode(GPIO.BCM)

    # Setup vibration pin to OUTPUT
    GPIO.setup(vibration_pin, GPIO.OUT)

   # Creates a variable defining the segment display
```

```
segment = SevenSegment.SevenSegment(address=0x70)

# Initialize the display. Must be called once before using the display.
segment.begin()

# The following commands only occur when count >=0
while(count >= 0):
  # Set up must be repeated or else buzzing won't occur every iteration
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(buzzer_pin, GPIO.OUT)
    # Buzzer turns on
    GPIO.output(buzzer_pin, GPIO.HIGH)
    # Buzzer goes off for 0.1 seconds
    time.sleep(0.1)
    # Buzzer turns off
    GPIO.output(buzzer_pin, GPIO.LOW)
    GPIO.cleanup()

    # Sets 10s place and 1s place
    segment.clear()
    segment.set_digit(2, int(count/10))
    segment.set_digit(3, count%10)

    # Write the display buffer to the hardware.  This must be called to
    # update the actual display LEDs.
    segment.write_display()

    # One second for each count
    time.sleep(1)
    # Count decreasing everytime the loop is completed
    count = count - 1

segment.clear()
segment.write_display()
# When the loop is finished (count < 0) a vibration will occur
```

```
    GPIO.setmode(GPIO.BCM)
    # Setup vibration pin to OUTPUT
    GPIO.setup(vibration_pin, GPIO.OUT)
    # Turn on vibration
    GPIO.output(vibration_pin, GPIO.HIGH)
    # Wait half a second
    time.sleep(1)
    # Turn off vibration
    GPIO.output(vibration_pin, GPIO.LOW)
    # Cleaup GPIO
    GPIO.cleanup()



# Subroutine that contains the ReactorPwrSCRAM code
# Has a parameter count which allows for the user to pick how long the countdown is (max 99s)
def ReactorPwrMatrix(count):
   # Define LCD column and row size for 16x2 LCD.
    lcd_columns = 16
    lcd_rows    = 2

    # Initialize the LCD using the pins
    lcd = LCD.Adafruit_CharLCDBackpack(address=0x21)

    # Turn backlight on
    lcd.set_backlight(0)

    # Print a two line message
    lcd.message('Reactor Power \nPercent:')

    # Create matrix device
    serial = spi(port=0, device=1, gpio=noop())
    device = max7219(serial, cascaded= 1, block_orientation=90, rotate= 0)

    # Defines variables as their BCM number to be called later
```

```
buttonUP_pin = 26
buttonDOWN_pin = 13
buzzer_pin = 18

# Setting command mode to BCM
GPIO.setmode(GPIO.BCM)

# Setup button pins as input (last command is to turn digital code into analog output)
GPIO.setup(buttonUP_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(buttonDOWN_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)

# Setup buzzer pin to OUTPUT
GPIO.setup(buzzer_pin, GPIO.OUT)

# Setting variables to be called in the loop
not_push = True
percent = 0
seconds = time.time()
scram = False

# Adresses the mode in the main function
GPIO.setmode(GPIO.BCM)

# Commands buttons to  increase and decrease percent
while not_push:
  # Commands percent to increase when up button is pressed

  if GPIO.input(buttonUP_pin) == 0 and not scram:

    # Stops percent from going above 110
    if(percent <= 100):
      percent = percent + 10

    # Print percent on matrix
      msg = str(percent) + "%"
```

```
                show_message(device, msg, fill="white", font=proportional(CP437_FONT),
scroll_delay=0.05)

            # Commands percent to decrease when down button pressed
            if GPIO.input(buttonDOWN_pin) == 0 and not scram:
                 # Stops percent from going below 0
                GPIO.output(buzzer_pin, GPIO.LOW)


                if percent > 10:
                    percent = percent - 10
                # Prints percent
                msg = str(percent) + "%"
                # Debugging
                show_message(device, msg, fill="white", font=proportional(CP437_FONT),
scroll_delay=0.05)

        # Has a beep go off every 1 second between 10% and 70%
            if percent <= 70 and time.time() - seconds > 1 and not scram:
                GPIO.output(buzzer_pin, GPIO.HIGH)
                time.sleep(0.1)
                GPIO.output(buzzer_pin, GPIO.LOW)
                seconds = time.time()

            # Starts SCRAM once percent reaches 110
            if percent == 110 or scram:
                # Creates a new message on the LED board
                lcd.clear()
                lcd.message('Caution: SCRAM')

                # Creates a delay to start SCRAM
                if percent == 110:
                    time.sleep(3)
                # Prints the 25% decrease
                scram = True
                percent = percent - 100
                msg = str(percent) + "%"
                show_message(device, msg, fill="white", font=proportional(CP437_FONT),
scroll_delay=0.1)
                time.sleep(3)
                percent = percent - 2
                msg = str(percent) + "%"
                show_message(device, msg, fill="white", font=proportional(CP437_FONT),
scroll_delay=0.1)
                time.sleep(5)
                percent = percent -1
                msg = str(percent) + "%"
                show_message(device, msg, fill="white", font=proportional(CP437_FONT),
scroll_delay=1.5)
                time.sleep(15)
                percent = percent - 1
                msg = str(percent) + "%"


            # Once the percent reaches 10%, scram stops
            if percent == 6 and scram:
                scram = False

                # Prints on LCD that there is xenon poisoning
                lcd.clear()
                lcd.message('Xenon poisoning \n recovery time:')

                # Creates a variable defining the segment display
                segment = SevenSegment.SevenSegment(address=0x70)

                # Initialize the display. Must be called once before using the display.
                segment.begin()

                # Runs xenon poisoning countdown
                while percent == 6 and scram == False:
                    # A minute and a half countdown begins
                    segment.clear()
                    segment.set_digit(2, int(count/10))
                segment.set_digit(3, count%10)

              # Write the display buffer to the hardware.  This must be called to
              # update the actual display LEDs.
                segment.write_display()

              # One second for each count
                time.sleep(1)
              # Count decreasing everytime the loop is completed
                count = count -1

                # count = 0, due to the order of operations we must say -1.
                if count == - 1:
                    # Prevents beeping from starting
                    GPIO.cleanup()
                    # Prints that the reactor is now ready
                    lcd.clear()
                    lcd.message('Reactor ready\n for restart')
                    time.sleep(10)
                    StartReactor()
                    # Stops the countdown from repeating forever
                    break


def StartReactor():
    try:
            InitializeReactor()
            # Pick your count variable, in this case, it is 10 seconds
            CountdownToReactor(3)
             # Pick your count variable, in this case, it is 90 seconds
            ReactorPwrMatrix(5)
    # if CTRL C is pressed, program stops
    except KeyboardInterrupt:
            pass
```

```
#2fa cyber informed decision
#Specific NFC Tag is required to start the reactor
def ReadCard():
    reader = SimpleMFRC522()
    GPIO.setwarnings(False)
    #Defines which tag
    access_rfid = "295606434863"
    #Defines the pin that needs to be entered to start the reactor
    pin = "12345"
        # Define LCD column and row size for 16x2 LCD.
    lcd_columns = 16
    lcd_rows    = 2

    # Initialize the LCD using the pins
    lcd = LCD.Adafruit_CharLCDBackpack(address=0x21)

    # Turn backlight on
    lcd.set_backlight(0)

    try:
        print('Start reading')
#       #Loop so its continuous until completed
        attempts = 0
        while(True):
            #add a delay so that it displays correctly on the LCD
            time.sleep(2)
            #clear the screen for the new message
            lcd.clear()
            lcd.message('Use tag \nto start reactor')
            id, text = reader.read()
            print(id)
            if str(id) == access_rfid:
                #Clears lcd screen
                lcd.clear()
                lcd.message('Enter Pin: ')
                #Allows input of pin
                pword = input(str())
                #Gives a max of 3 attempts before adding a delay
                if attempts >= 3:
                    lcd.clear()
                    lcd.message('To many attempts\nTry in 1 minute')
                    #Makes it so nothing happens for 60 seconds
                    time.sleep(60)
                    #Resets the variable back to 0
                    attempts = 0
                #Makes it so pin is required to start the reactor
                if pword == pin:
                    StartReactor()
                #Adds attempts if pin was incorrect
                elif pword != pin:
                    attempts = attempts + 1
                    #Debugging
                    print(attempts)
                else:
                    lcd.clear()
                    lcd.message("Invalid Pin")
                    continue
            else:
                lcd.clear()
                lcd.message("Invalid Tag")
                continue
    finally:
        GPIO.cleanup()

def connectToWifi(ssid:str, password:str) -> bool:
    """This function connects to a wifi network. It returns true if connected and false otherwise"""
    # Edit wpa_supplicant configuration
    with open('/etc/wpa_supplicant/wpa_supplicant.conf', 'a') as f:
        f.write(f'\nnetwork={{\n    ssid="{ssid}"\n    psk="{password}"\n}}\n')
    # Restart wpa_supplicant
    subprocess.run(['sudo', 'systemctl', 'restart', 'wpa_supplicant'])

    # Wait for connection
    time.sleep(5)

    # Check if interface has obtained IP address
    result = subprocess.run(['ifconfig', 'wlan0'], capture_output=True, text=True)
    if 'inet ' in result.stdout:
        print(f"Successfully connected to WiFi network '{ssid}'")
        return True
    else:
        print(f"Failed to connect to WiFi network '{ssid}'")
        print("Error:", result.stderr)
        return False

print(f"Attempting to connect to WiFi network: '{ssid}'")
while connectToWifi(ssid, password):
    print(".")
    time.sleep(1)



# try:
#     # ReadCard()
#     StartReactor()
# except KeyboardInterrupt:
#     pass
```

A-3. LED and Potentiometer Code

```
#include <WiFiNINA.h>

const int lightSensorPin = A5;
const int pPin = A0;
const char ssid[] = "hostArduino";
const char pass[] = "1234567890";
const int ledPin1 = 1;
const int ledPin2 = 2;
const int ledPin3 = 3;
const int ledPin4 = 4;

int pReading;
int lightSensorReading;
int range;

// these handle UDP
WiFiUDP Udp; // creates a udp object to handle sending and recieving UDP data
char packetBuffer[255]; // Stores UDP messages

int securityState = 0; // used to compare what level of attack we are in

int securityStates[5][3] = {
  {0, lightSensorReading*2, range},
  {1, 20, 0},
  {2, -20, 0},
  {3, 0, 20},
  {4, random(-140, 141), random(-140, 141)}
};

void setup() {
  Serial.begin(9600);
  while (!Serial) {
    ; // Wait for serial to connect
  }

  // Set out pins (This is only for digital pins)
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(ledPin3, OUTPUT);
  pinMode(ledPin4, OUTPUT);
  // this chunk of code connects to the network
  WiFi.config(IPAddress(192, 168, 1, 2)); // set a stable IP so we can send data around

  Serial.print("Attempting to connect to SSID: ");
  Serial.println(ssid);
  while (WiFi.begin(ssid, pass) != WL_CONNECTED) {
    Serial.print(".");
    delay(5000);
  }
  Serial.println("\nYou're connected to the network\n");

  // Begin udp so we can send and recieve messages
  Udp.begin(2390);
}

void printValues(int val1, int val2) {
// Print values in CSV format for the Serial Plotter
  Serial.print(100);
  Serial.print(",");
  Serial.print(val1);
  Serial.print(",");
  Serial.print(val2);
  Serial.print(",");
  Serial.println(0); // The last value should be followed by println to move to the next line

  // Note: The Serial Plotter does not support titles for each value.
  // The order of the values will determine which plot corresponds to each sensor.
  // Light Sensor Value, Potentiometer Percentage, Range/25
}

void lightLeds(bool led1 = false, bool led2 = false, bool led3 = false, bool led4 = false) {
  digitalWrite(ledPin1, led1);
  digitalWrite(ledPin2, led2);
  digitalWrite(ledPin3, led3);
  digitalWrite(ledPin4, led4);
}

void loop() {
  lightSensorReading = analogRead(lightSensorPin);
  pReading = analogRead(pPin);
  int range = map(pReading, 0, 1023, 0, 100);

  // Activate pins based on the potentiometer reading
  lightLeds();

  if (range > 5) {
    lightLeds(true);

    if (range > 25) {
      lightLeds(true, true);

      if (range > 50) {
        lightLeds(true, true, true);

        if (range > 75) {
          lightLeds(true, true, true, true);
        }
      }
    }
  }

  // this section recieves UDP packets
  int packetSize = Udp.parsePacket();

  if (packetSize) {
    Udp.read(packetBuffer, 255);

    // This makes sure the data we recieved is actually a Security level and not some other data
    if (strstr(packetBuffer, "Security Level: ") != NULL) {
      securityState = atoi(strstr(packetBuffer, ": ") + 2); // This pulls the actual security level from the string. It assumes that the security level follows ": " for example ": 1" would get 1 but ":1" would not
    }

    // This code is used to send a message back to say that we recieved the packet
    Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
    Udp.print("recieved packet: ");
    Udp.print(packetBuffer);
    Udp.print("\nReturn packet sent from: ");
    Udp.print(WiFi.localIP());
    Udp.endPacket();
  }

  // these control the random number in security state 4. They're just here to update the value every loop
  securityStates[4][1] = random(-lightSensorReading*2, 141);
  securityStates[4][2] = random(-range, 141);

  // This section controls the Security state we are in
  for (int i = 0; i < 5; i++) {
    if (securityState == securityStates[i][0]) {
      printValues(lightSensorReading*2.5 + securityStates[i][1], range + securityStates[i][2]);
      break;
    }
  }
}
```

A-4. Wifi Spoofing

```
#include <WiFiNINA.h>

const int button1Pin = 2;
const int button2Pin = 3;
const int button3Pin = 4;
const int button4Pin = 5;
const int button5Pin = 6;
const char ssid[] = "hostArduino";
const char pass[] = "1234567890";

int status = WL_IDLE_STATUS;

int buttonState = 0; // this is used to compare what level of threat we are in
int reseter = 0;

WiFiUDP Udp; // create a udp object to handle sending UDP data
char packetBuffer[255]; // store UDP messages

void setup() {
  Serial.begin(19200);
  // Can be removed just here for easier debugging purposes
  // while (!Serial) {
  //   ; // Wait for serial to connect
  // }
  // end

  // setup our pins
  pinMode(button1Pin, INPUT);
  pinMode(button2Pin, INPUT);
  pinMode(button3Pin, INPUT);
  pinMode(button4Pin, INPUT);
  pinMode(button5Pin, INPUT);

  // create the wifi network
  Serial.print("Creating access point named: ");
  Serial.println(ssid);

  WiFi.config(IPAddress(192, 168, 1, 3)); // set a stable IP so we can send data around

  // Create open network
  status = WiFi.beginAP(ssid, pass);
  if (status != WL_AP_LISTENING) {
    Serial.println("Creating access point failed");
    // don't continue
    while (true);
  }

  delay(10000); // wait for everything to be initialized
  Serial.println("Ready for connections");

  Udp.begin(2390);
}

// this function handles sending data
void sendData(char data[] = "", IPAddress ip = IPAddress(192, 168, 1, 2), int port = 2390) {
  // all of the serial and if statments in this function are for error catching and debugging purposes. The
only required code is in the coment below
  // Udp.beginPacket(ip, port); // returns 1 if the Ip address and port work
  // Udp.write(data); // actually sends the data
  // Udp.endPacket(); // finishes sending the data. Returns 1 if successful
  Serial.print("Atempting to send data: ");
  Serial.println(data);

  if (Udp.beginPacket(ip, port) == 1) {
    Udp.write(data);
    if (Udp.endPacket() == 1) {
      Serial.println("Data sent succesfully");
    } else {
      Serial.println("Data not sent");
    }
  } else {
    Serial.println("There was a problem with the supplied IP address or port");
  }
}

void loop() {
  // this displays when a device connets or disconnects from the access point
```

```
    if (status != WiFi.status()) {
      status = WiFi.status();

      if (status == WL_AP_CONNECTED) {
        Serial.println("Device connected to AP!");
      } else {
        Serial.println("Device disconnected from AP :(");
      }
    }

    // TODO this feels ineffecient
    if(digitalRead(button5Pin) == HIGH) { // Reset all security conditions
      Serial.println("Button 5 Pressed");
      buttonState = 0;
      sendData("Security Level: 0");
      reseter = 1;
      delay(1000);

    } else if(digitalRead(button1Pin) == HIGH  && (buttonState != 1 && buttonState != 2 &&
buttonState != 3 && buttonState != 4)) { // security level 1
      Serial.println("Button 1 Pressed");
      buttonState = 1;
      sendData("Security Level: 1");
      reseter = 0;
      delay(1000);

    } else if(digitalRead(button2Pin) == HIGH && (buttonState != 2 && buttonState != 3 &&
buttonState != 4)) { // security level 2
      Serial.println("Button 2 Pressed");
      buttonState = 2;
      sendData("Security Level: 2");
      delay(1000);
      reseter = 0;

    } else if(digitalRead(button3Pin) == HIGH  && (buttonState != 3 && buttonState != 4)) { //
security level 3
```

```
    Serial.println("Button 3 Pressed");
    buttonState = 3;
    sendData("Security Level: 3");
    delay(1000);
    reseter = 0;

  } else if(digitalRead(button4Pin) == HIGH && buttonState != 4) { // security level 4
    Serial.println("Button 4 Pressed");
    buttonState = 4;
    sendData("Security Level: 4");
    delay(1000);
    reseter = 0;

  }

  // this section recieves UDP packets
  int packetSize = Udp.parsePacket();

  if (packetSize) {
    Udp.read(packetBuffer, 255);
    Serial.println(packetBuffer);
  }
}
```