GuardianLock

Caroline Rinks, Brandan Roachell, Reed Semmel, Samuel Thompson

Team Garden Guardians

Demo video: https://youtu.be/Tk5_R_vAkZc

Abstract

While there are many locks on the market, very few provide the user with access history or lock tampering detection. We propose a new lock system that consists of a physical padlock connected via WiFi to a web app. Each time the lock is locked or unlocked, it is recorded in the app for the user to view. Additionally, the app will send a notification to the user if tampering with the lock is detected as well as monitor online connection.

Introduction

When people buy padlocks, they wish to protect whatever lies behind the lock. While this can be achieved by physically monitoring the location where the items lie, people like to be able to leave the area while their items remain protected. One critical flaw in this situation is that the user of the padlock does not know the status of the integrity of the lock while unattended. An active, smart monitoring system can alert the user when security has been compromised in hopes that the user can react fast enough to the security intrusion.

The GuardianLock aims to be a simple device to meet this need. It behaves exactly as a padlock that can be open and closed with a key. In addition, the lock is IoT-connected, allowing unlocks, locks, and tampers of the lock to be logged and monitored.

The goals of this project include creating a product that:

1. Behaves as a normal padlock,
2. Maintains its security profile,
3. Provides live monitoring of locks and unlocks, and
4. Provides live monitoring of tampering.

In other words, our goal is to create a smart lock that does not compromise on security. Many smart locks have been shown to have a poor security profile [1, 2]. By having a limited set of smart features that do not actively interact with the lock, we can maintain the security profile of a standard padlock, and passive monitoring can provide the desired smart features.

Market Research and Existing Work

In our search for products in the same space, we found many products involving a fingerprint sensor to unlock the lock. These types of products have been shown to often be trivial to compromise [3].

Originally, we proposed a padlock with many more features, such as the ability to unlock and lock the padlock remotely, as well as a camera to allow the user to see what is happening near the padlock. We found many products with these two features. However, we did not find a single product offering tamper detection. Through this research, we reformed our idea to create a simpler product that only provides activity and tamper monitoring. This will reduce cost while providing a feature that does not exist on the market.

For example, one product [4] allows users to control the padlock with their phone and also has a fingerprint scanner. However, without tamper detection, the user cannot know if anything is happening at the site of the lock. Another product [5] also features a fingerprint reader for unlocking, which is a common feature in this market.

Additionally, while many of these devices use Bluetooth, our product will only use WiFi and an external server. This allows the user to be notified of locks, unlocks, and tampers from anywhere in the world as long as they have an internet connection.

Methodology

Originally, we wanted to build an automated plant maintenance system, but since two other teams had the same idea, we decided to pivot to a different idea from our brainstorming session. Our second favorite idea was a smart lock that could be accessed over the internet, and this is the idea we ultimately decided to pursue.

To begin, we wanted to see how locks work internally. One team member had a set of locks with plastics bodies which allows for easy access:

Figure 1: The opened lock with visible internals.

With this lock open, we could see the mechanism, and it enabled us to try creating a new body to make room for the electronics.

However, we quickly realized that this would not be a feasible solution with the time constraints at hand, so we decided to create an outer housing that goes around an existing lock. This has the advantage of not compromising the padlock in any capacity.

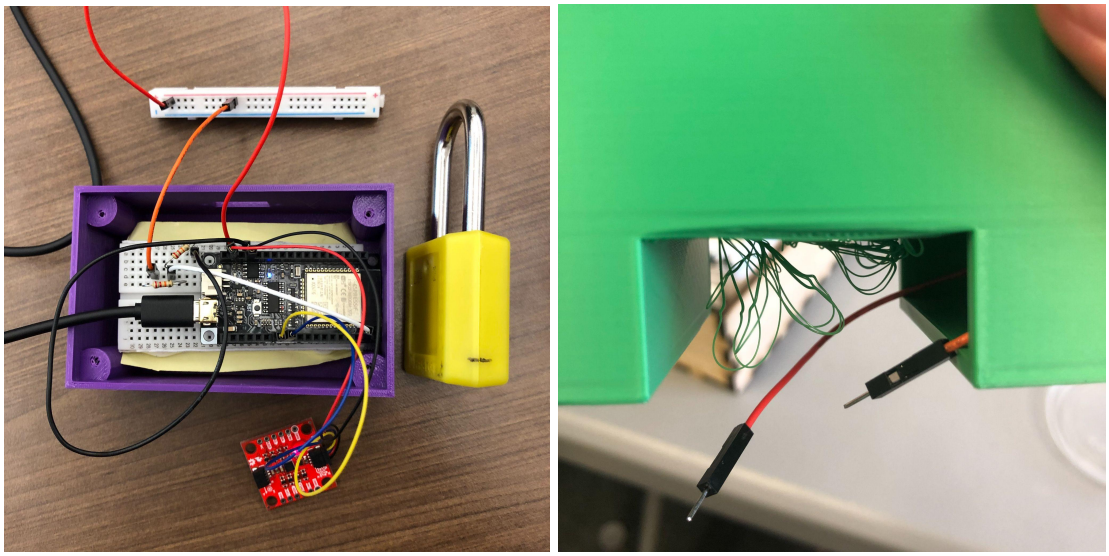

Figure 2: The original prototype (left) with plans for having the internals integrated into the design. The final prototype (right) has a cutout for the lock to fit in. The box will hold the electronics to monitor the lock without compromising the lock's security.

For determining whether the lock is locked or unlocked, we can check if the shackle is in the closed position by placing copper tape in the shackle hole. Since the shackle is made of metal and the enclosure is made of plastic, we can use this conditional connection to create a voltage divider between the shackle and the lock body. This voltage divider is how the ESP32 determines whether or not the shackle is open.

For tamper detection, the external enclosure also contains an IMU, and its gyroscope movements are used to trigger a tamper notification after a threshold.

From here, the device itself is complete. The logic on the ESP32 handles cleaning up the data received from these two sources. Once an event has been detected, it sends the event to an external server over WiFi. This server handles receiving events from the ESP32 as well as providing the end user interface that allows the user to receive these events.

The web interface is a single web page using React. The web page connects to the server using a WebSocket to handle real time event notifications. The server serves this web application as well as streaming events to the WebSockets.

Results

We were successfully able to build a system that wirelessly communicates the state of the lock to a web app, including alerts for tampering and disconnection. Figure 3 shows images of the final physical product. The final prototype of our lock is a design which incorporates a padlock with integrated sensors. This connects our lock to the backend and frontend which handle all of the sensor data and format this into a comprehensive view of the entire lock history and its current state.

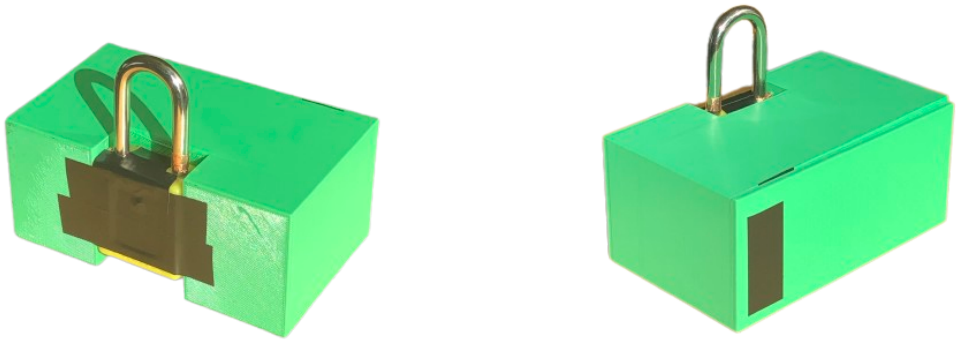

Figure 3: Images of the completed prototype.

Along with the physical lock interface, a custom user interface that is able to be accessed from any device, whether that be a desktop, laptop, or mobile device. This is what the user will see when they are using the app, in which they may view the various statuses of their physical lock. These statuses include two online indicators for the lock: connected and disconnected, as well as three event indicators which are as follows: locked, unlocked, and tamper. An example of the final user interface in action may be observed in Figure 4. For immediate feedback from the physical lock, the frontend was developed using React that connected to the backend that was developed in GO and is connected through the ESP32 using WiFi connection in the physical housing of the device. The code for this backend is available in Appendix A, while the code that supplied the frontend is available in Appendix C.

The code on the ESP32 is available in Appendix B.

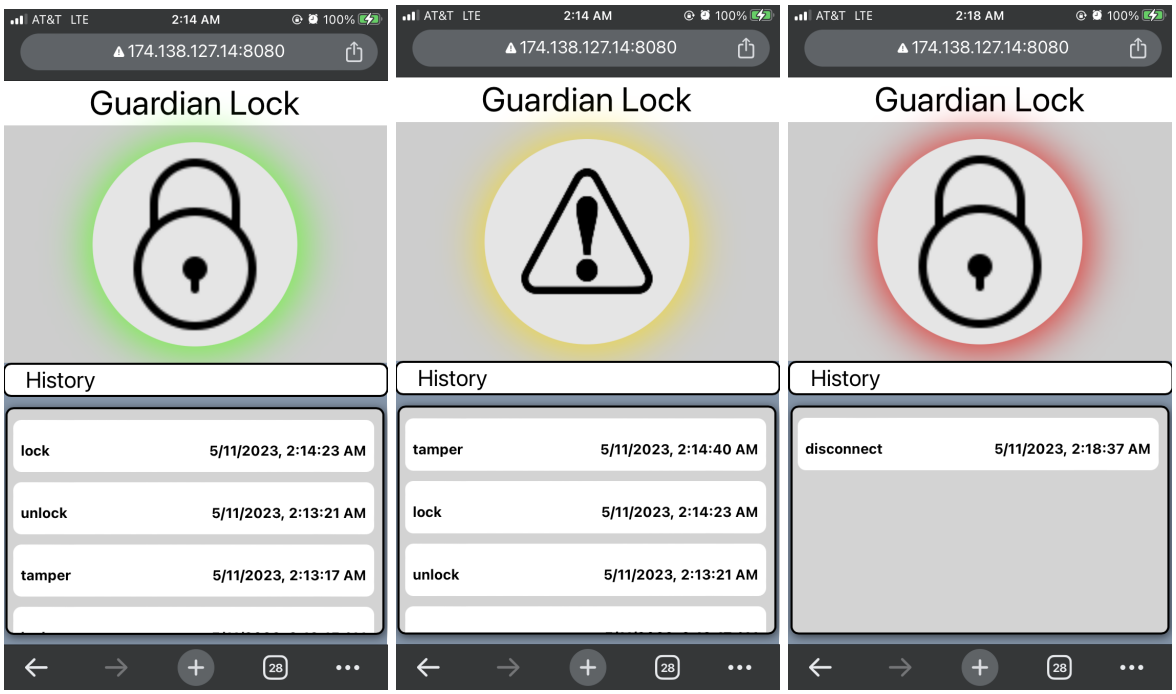

Figure 4: Screenshots of the web application showing the variety of states of the lock.

As for the internals of the Guardian Lock system, there are three main components. These components consist of an IMU, an ESP32, and a power system. Both the IMU and the ESP32 were provided by the course kit which came in handy since we already had these on hand as well as had some experience with them. The power source that we are currently using in the working prototype is a portable charger that we found when looking for power sources. This does end up taking more space in the physical housing due to the casing around the battery but allows the lock to be active for multiple hours on end. We initially considered AAA batteries and in the future still may want to explore this idea or others that will save space in the physical lock housing. A full breakdown of the physical working model of our lock may be found in Figure 5, which includes all main components in the case and laid out for easy comprehension.

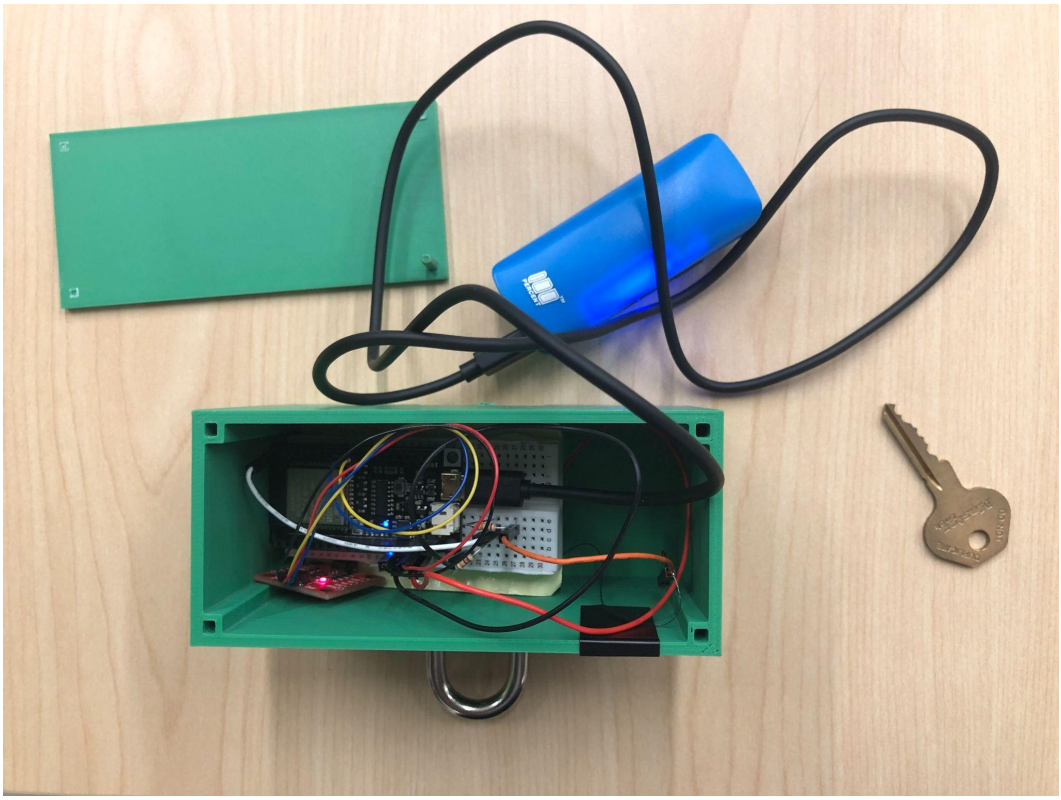

Figure 5: The internals of the exterior inclosure.

Notable challenges that we faced were the housing for the lock and UI being accessible on any device. The main challenge that came with the housing for the lock was the fact that we did not know how to minimize the space that the housing took up while not compromising the integrity or security of the padlock itself. In order to overcome this challenge, we actually made the innovation of a system that simply includes all needed electronics that forms around a pre-existing locking mechanism. Our other challenge was due to the fact that the customers of our lock may want to access their lock from any device. Originally, we were only supporting full screen systems on a laptop or desktop form, which we realized was not going to fit all uses of the lock. In order to overcome this, we created a frontend that adapted dynamically to each screen that it was rendered on.

Discussion

Overall, the GuardianLock was successful, and it does exactly what we set out to do, can be useful in practice, and has great potential for improvement into a market-ready product.

As a class project, there were many limitations, one of the biggest being limited time and resources. As previously discussed, the scope of this project was initially very large due to the many features we had brainstormed for our device, such as a built-in camera and remote locking and unlocking, but we later realized that not only was it infeasible within the given timeframe, they were actually undesirable features. Once we focused the efforts into the parts that mattered, we were able to make significant progress.

We were also financially limited in this project. Consequently, our final prototype was built with cheaper materials and used less refined electronics such as a whole breadboard. Ideally, this could be redesigned with a smaller—possibly custom—circuit board that has all the necessary functionality to massively reduce the size of the system, as well as print a smaller housing and integrate the lock's mechanisms directly into the electronics. This could also be better executed with a CNC machine out of metal in order to maintain full durability throughout the entire product. In its current state, more thorough and accurate testing of our product is needed to evaluate its security. Due to the nature of the use case for our product, it is difficult to create testing scenario conditions that are realistic to an actual security breach, which is likely to be more destructive. However, given the alerting feature, any kind of action taken against the lock can and will be reported, and the strength of protection still lies within the in-tact lock itself. Users of our product have nothing to lose when upgrading to the GuardianLock from a traditional padlock.

For future work, it may also be interesting to use AI to better detect tampering. Our implementation of tampering detection uses gyroscope measurements alone, but perhaps there are other parameters and patterns that can be used to identify tampering instead. A binary classification model could be trained with machine

learning methods and take advantage of these additional parameters to more accurately predict whether the lock is being tampered with. We also would like to have mobile push notifications to the user instead of using a web app interface that must be manually checked or monitored, as well as housing that allows for easier recharging.

Conclusion

The GuardianLock is a smart padlock that addresses the limitations of traditional padlocks by providing access history and tamper detection. Our research and development efforts have successfully resulted in a functional prototype that wirelessly communicates the status of the padlock to a web app, offering real-time monitoring.

The key findings of our project demonstrate the feasibility of creating a smart padlock that maintains high security standards while providing monitoring features. By offering access history and tamper detection, users can have peace of mind and promptly respond to any security threats.

Looking ahead, future work could involve further refining the design, incorporating additional security features, and enhancing the user interface. With its unique combination of security and smart features, the GuardianLock is poised to make a significant impact in the market and provide a safer and more convenient locking solution for users worldwide.

References

[1]: https://www.youtube.com/watch?v=Os1oyIfhM-o
[2]: https://www.youtube.com/watch?v=m_MX96MVD00
[3]: https://www.youtube.com/watch?v=0bCEiRSTMMM
[4]: https://www.amazon.com/Anweller-Fingerprint-Padlock-Weatherproof-Biometric/dp/B0BK8Z53L7/
[5]: https://www.amazon.com/Fingerprint-eLinkSmart-Remotely-Authorized-Bluetooth/dp/B07HFZWQ1R

Appendices

Appendix A: Backend Code

The backend code is responsible for receiving events from the ESP32 and broadcasting said events to the web application.

```
package main

import (
     "encoding/json"
     "net/http"
     "sync"
     "time"

     "golang.org/x/net/websocket"
)

type Event struct {
     Event string    `json:"event"`
     Time  time.Time `json:"time"`
}

var events = []Event{}
var eventLock = sync.Mutex{}

var connected = map[chan string]struct{}{}
var connectedLock = sync.Mutex{}

var timer = time.AfterFunc(30*time.Second, func() {
     eventLock.Lock()
     events = append(events, Event{"disconnect", time.Now()})
     eventLock.Unlock()
     writeToAll("disconnect")
})

func writeToAll(msg string) {
     connectedLock.Lock()
     defer connectedLock.Unlock()
     for c := range connected {
```

```
                c <- msg
     }
}

func heartbeatHandle(w http.ResponseWriter, r *http.Request) {
     res := timer.Reset(30 * time.Second)
     if !res {
                eventLock.Lock()
                events = append(events, Event{"connect",
time.Now()})
                eventLock.Unlock()
                writeToAll("connect")
     }
}

// Path /api/get-history returns all events as an http handler
func historyHandle(w http.ResponseWriter, r *http.Request) {

     w.Header().Set("Access-Control-Allow-Origin", "*")
     w.Header().Set("Content-Type", "application/json")

     eventLock.Lock()
     defer eventLock.Unlock()
     json.NewEncoder(w).Encode(events)
}

func deleteHistory(w http.ResponseWriter, r *http.Request) {
     w.Header().Set("Access-Control-Allow-Origin", "*")
     eventLock.Lock()
     defer eventLock.Unlock()

     events = events[:0]
     w.WriteHeader(http.StatusOK)
}

// Path /api/tamper adds a tamper event to the history
func tamperHandle(w http.ResponseWriter, r *http.Request) {
     eventLock.Lock()
     events = append(events, Event{"tamper", time.Now()})
     eventLock.Unlock()
```

```
	// Send tamper event to all connected clients
	writeToAll("tamper")
}

// Path /api/lock adds a lock event to the history
func lockHandle(w http.ResponseWriter, r *http.Request) {
	eventLock.Lock()
	events = append(events, Event{"lock", time.Now()})
	eventLock.Unlock()

	// Send lock event to all connected clients
	writeToAll("lock")
}

// Path /api/unlock adds an unlock event to the history
func unlockHandle(w http.ResponseWriter, r *http.Request) {
	eventLock.Lock()
	events = append(events, Event{"unlock", time.Now()})
	eventLock.Unlock()

	// Send unlock event to all connected clients
	writeToAll("unlock")
}

func socketHandle(ws *websocket.Conn) {
	ch := make(chan string)
	// Add channel to connected map
	connectedLock.Lock()
	connected[ch] = struct{}{}
	connectedLock.Unlock()

	// Remove channel from connected map when this function
returns
	defer func() {
		connectedLock.Lock()
		delete(connected, ch)
		connectedLock.Unlock()
	}()

	// Send events to clients as we receive them
	for {
```

```
                msg := <-ch
                err := websocket.Message.Send(ws, msg)
                if err != nil {
                        return
                }
        }
}

func main() {
        http.HandleFunc("/api/get-history", historyHandle)
        http.HandleFunc("/api/tamper", tamperHandle)
        http.HandleFunc("/api/lock", lockHandle)
        http.HandleFunc("/api/unlock", unlockHandle)
        http.HandleFunc("/api/heartbeat", heartbeatHandle)
        http.HandleFunc("/api/delete-history", deleteHistory)

        websocketServer := websocket.Server{
                Handshake: func(ws *websocket.Config, req
*http.Request) error {
                        return nil
                },
                Handler: websocket.Handler(socketHandle),
        }

        http.Handle("/api/ws", websocketServer)

        // Path / serves the static files from the /static
directory
        http.Handle("/", http.FileServer(http.Dir("./static")))
        http.ListenAndServe("0.0.0.0:8080", nil)
}
```

Appendix B: ESP32 Code

The code on the ESP32 is responsible for sending events to the backend when it detects changes to the lock's status.

```
//
https://wiki.dfrobot.com/FireBeetle_ESP32_IOT_Microcontroller(V3
.0)__Supports_Wi-Fi_&_Bluetooth__SKU__DFR0478
```

```
#define LOCK A0
#define THRESHOLD 1000

#include <WiFi.h>
#include <HTTPClient.h>
#include <Arduino_JSON.h>
#include "ICM_20948.h"
#include <MadgwickAHRS.h>

ICM_20948_I2C imu;
Madgwick filter;

unsigned long microsPerReading, microsPrevious, microsNow;
float roll, pitch, yaw, error, actual = 90;

const char *ssid = "ut-open";
const char *password = "";
const char *serverIp = "174.138.127.14"; // cloud server IP
const int serverPort = 8080;

String serverName = "http://" + String(serverIp) + ":" +
String(serverPort);

// the following variables are unsigned longs because the time,
measured in
// milliseconds, will quickly become a bigger number than can be
stored in an int.
unsigned long lastTime = 0;
unsigned long lastTimeHeartbeat = 0;
unsigned long lastTimeUnlock = 0;
unsigned long tamperTimeBegin = 0;
unsigned long timerDelay = 2000;
unsigned long heartbeatDelay = 10000;
unsigned long tamperDelay = 30000;

void initWiFi() {
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi ..");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print('.');
```

```
    delay(1000);
  }
  Serial.println(WiFi.localIP());
}

void setup() {
  Serial.begin(115200);

  pinMode(LOCK, INPUT);

  // WiFi setup

  Serial.println();
  Serial.print("ESP Board MAC Address: ");
  Serial.println(WiFi.macAddress());

  // Set WiFi to station mode and disconnect from an AP if it
was previously connected
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);

  Serial.println("Setup done");

  initWiFi();

  while (!Serial) {}

  Wire.begin();
  Wire.setClock(400000);

  // IMU setup

  while (1) {
    imu.begin(Wire, 1);
    Serial.print("IMU Status: ");
    Serial.println(imu.statusString());
    if (imu.status == ICM_20948_Stat_Ok) {
      break;
    }
  }
```

```
  filter.begin(200); // This many samples per second
  // Initialize variables to pace updates to correct rate
  microsPerReading = 1000000 / 200;
  microsPrevious = micros();
}

bool isHigh = false;
bool isLocked = true;
bool isTampered = false;
bool changed = false;
int tamperCounter = 0;

void loop() {
  // Check if it's time to read data and update the filter
  microsNow = micros();
  if (imu.dataReady() && microsNow - microsPrevious >=
microsPerReading) {
    imu.getAGMT();

    // Only count tampering while locked
    if(isLocked && (imu.gyrX() > 100 || imu.gyrY() > 100 ||
imu.gyrZ() > 100)) tamperCounter++;
    if(tamperCounter > 300) {
      isTampered = true;
      tamperCounter = 0;
      tamperTimeBegin = millis();
    }

    // Tell the filter to just ignore the accelerometer
entirely,
    // since it was causing readings to incorrectly drift
    filter.updateIMU(imu.gyrX(), imu.gyrY(), imu.gyrZ(), 0, 0,
0);

    roll = filter.getRoll();
    pitch = filter.getPitch();
    yaw = filter.getYaw();

    // Serial.print("Roll: ");
    // printFormattedFloat(roll, 3, 3);
```

```
    // Serial.print("  Pitch: ");
    // printFormattedFloat(pitch, 3, 3);
    // Serial.print("  Yaw: ");
    // printFormattedFloat(yaw, 3, 3);
    // Serial.print("  Angle: ");
    // printFormattedFloat(actual, 3, 3);
    // Serial.print("  Error: ");
    // printFormattedFloat(error, 3, 3);
    // Serial.println();

    // Increment previous time, so we keep proper pace
    microsPrevious = microsPrevious + microsPerReading;
  } else {
    //Serial.println("No data");
  }

  // Serial.println("scan start");

  // // WiFi.scanNetworks will return the number of networks
found
  // int n = WiFi.scanNetworks();
  // Serial.println("scan done");
  // if (n == 0) {
  //     Serial.println("no networks found");
  // } else {
  //   Serial.print(n);
  //   Serial.println(" networks found");
  //   for (int i = 0; i < n; ++i) {
  //     // Print SSID and RSSI for each network found
  //     Serial.print(i + 1);
  //     Serial.print(": ");
  //     Serial.print(WiFi.SSID(i));
  //     Serial.print(" (");
  //     Serial.print(WiFi.RSSI(i));
  //     Serial.print(")");
  //     Serial.println((WiFi.encryptionType(i) ==
WIFI_AUTH_OPEN)?" ":"*");
  //     delay(10);
  //   }
  // }
  // Serial.println("");
```

```
  // // Wait a bit before scanning again
  // delay(5000);

  // Check if it needs to send an HTTP GET request every time
delay
  if((millis() - lastTime) > timerDelay) {
    // Lock status
    int lockRead = analogRead(LOCK);

    // Serial.println(lockRead);

    bool newValue = lockRead > THRESHOLD;
    if(newValue && !isHigh) {
      isLocked = true;
      changed = true;
    } else if(!newValue && isHigh) {
      isLocked = false;
      changed = true;
    }
    isHigh = newValue;

    // Check WiFi connection status
    if(WiFi.status() == WL_CONNECTED) {
      HTTPClient http;

      String endpointLock = serverName + "/api/lock";
      String endpointUnlock = serverName + "/api/unlock";
      String endpointTamper = serverName + "/api/tamper";
      String endpointHeartbeat = serverName + "/api/heartbeat";

      // Send heartbeats to the server to detect disconnect
      if((millis() - lastTimeHeartbeat) > heartbeatDelay) {
          http.begin(endpointHeartbeat);

          // Send HTTP GET request
          int httpResponseCode = http.GET();

          if(httpResponseCode > 0) {
            Serial.print("HTTP response code from
/api/heartbeat: ");
```

```
      Serial.println(httpResponseCode);
      String payload = http.getString();
      Serial.println(payload);
    }
    else {
      Serial.print("Error code: ");
      Serial.println(httpResponseCode);
    }
    // Free resources
    http.end();
    lastTimeHeartbeat = millis();
}

if(changed) {
  changed = false;
  tamperCounter = 0;

  if(isLocked) {
    http.begin(endpointLock);

    // Send HTTP GET request
    int httpResponseCode = http.GET();

    if(httpResponseCode > 0) {
      Serial.print("HTTP response code from /api/lock: ");
      Serial.println(httpResponseCode);
      String payload = http.getString();
      Serial.println(payload);
    }
    else {
      Serial.print("Error code: ");
      Serial.println(httpResponseCode);
    }
    // Free resources
    http.end();
  } else if(!isLocked) {
    http.begin(endpointUnlock);

    // Send HTTP GET request
    int httpResponseCode = http.GET();
```

```
      if(httpResponseCode > 0) {
        Serial.print("HTTP response code /api/unlock: ");
        Serial.println(httpResponseCode);
        String payload = http.getString();
        Serial.println(payload);
      }
      else {
        Serial.print("Error code: ");
        Serial.println(httpResponseCode);
      }
      // Free resources
      http.end();
      lastTimeUnlock = millis();
    }
  } else if(isTampered) {
    if((millis() - tamperTimeBegin) > tamperDelay) {
      if((millis() - lastTimeUnlock) > tamperDelay) {
        http.begin(endpointTamper);

        // Send HTTP GET request
        int httpResponseCode = http.GET();

        if(httpResponseCode > 0) {
          Serial.print("HTTP response code /api/tamper: ");
          Serial.println(httpResponseCode);
          String payload = http.getString();
          Serial.println(payload);
        }
        else {
          Serial.print("Error code: ");
          Serial.println(httpResponseCode);
        }
        // Free resources
        http.end();
      }
      // Reset isTampered after every tamperDelay
      isTampered = false;
    }
  }
}
else {
```

```
      Serial.println("WiFi Disconnected");
    }
    lastTime = millis();
  }
}

void printFormattedFloat(float val, uint8_t leading, uint8_t
decimals)
{
  float aval = abs(val);
  if (val < 0)
    Serial.print("-");
  else
    Serial.print(" ");
  for (uint8_t indi = 0; indi < leading; indi++) {
    uint32_t tenpow = 0;
    if (indi < (leading - 1))
      tenpow = 1;
    for (uint8_t c = 0; c < (leading - 1 - indi); c++)
      tenpow *= 10;
    if (aval < tenpow)
      Serial.print("0");
    else
      break;
  }
  if (val < 0)
    Serial.print(-val, decimals);
  else
    Serial.print(val, decimals);
}
```

Appendix C: Frontend Code

The code that is responsible for generating the frontend and connecting to the backend to display usable information to the customer.

```
import { useEffect, useRef, useState } from 'react';
import { Text } from 'react-native'
import lockLogo from './assets/lock.png'
import unlockLogo from './assets/unlock.png'
import tamperLogo from './assets/tamper.png'
```

```
import { useWebSocket } from
'react-use-websocket/dist/lib/use-websocket';

function TestingFunc({ width, height, title, message, time,
date, landscape}) {

  if (landscape == true) {
     return(
     <div style={{ width: width * 0.48, marginTop: 5,
marginBottom: 5, alignSelf: 'center', alignContent: 'center',
backgroundColor: 'white', borderWidth: 5, borderColor: 'black',
borderRadius: 8 }}>
     <div style={{ display: 'flex', flexDirection: 'column',
padding: 10, alignItems: 'center' }}>

          <div style={{ height: 20, width: width * 0.44,
display: 'flex', flexDirection: 'row', justifyContent:
'space-between', marginTop: 10, marginBottom: 10}}>
          <div>
               <Text style={{ fontWeight: 'bold', fontSize: 25
}}>
               {title}
               </Text>
          </div>

          <div>
               <Text style={{ fontWeight: 'bold', fontSize: 25
}}>
               {(new Date(time)).toLocaleString()}
               </Text>
          </div>
          </div>
     </div>
     </div>
     );
  } else {
     return(
     <div style={{ width: width * 0.94, marginTop: 5,
marginBottom: 5, backgroundColor: 'white', borderWidth: 5,
borderColor: 'black', borderRadius: 8 }}>
```

```
     <div style={{ display: 'flex', flexDirection: 'column',
padding: 10, alignItems: 'center' }}>

          <div style={{ height: height * 0.02, width: width *
0.90, display: 'flex', justifyContent: 'space-between',
alignContent: 'center', marginTop: 10, marginBottom: 10}}>
          <div>
               <Text style={{ fontWeight: 'bold', fontSize:
height * 0.025 }}>
               {title}
               </Text>
          </div>

          <div>
               <Text style={{ fontWeight: 'bold', fontSize:
height * 0.025 }}>
               {(new Date(time)).toLocaleString()}
               </Text>
          </div>
          </div>
     </div>
     </div>
     );
  }

}

export default function Game() {
  const [currentLock, setCurentLock] = useState([]);
  const [lockHistory, updateLockHistory] = useState([]);
  const width = window.innerWidth;
  const height = window.innerHeight;

  const { lastMessage } =
useWebSocket('ws://174.138.127.14:8080/api/ws');

  // () ? True : False

  // const landscape = ((width <= height) ? true : false)
  const landscape = (width >= height)
```

```
  useEffect(() => {
     fetch('http://174.138.127.14:8080/api/get-history')
     .then(response => response.json())
     .then((data) => {
     updateLockHistory(data.reverse());
     })
  }, [lastMessage]);

  function LockButton({ height, width, lockStatus, landscape })
{
     let shadow = "";
     let lock = "";

     // yellow is FCD800
     console.log(lockStatus)

     if (lockStatus[0] == "disconnect") {
     shadow = '1px 2px 50px #FF0000';
     } else {
     if (lockStatus[1] == "tamper") {
     shadow = '1px 2px 50px #FCD800';
     } else {
     shadow = '1px 2px 50px #28FF00';
     }
     }

     console.log("The lock status is: " + lockStatus[1])
     if (lockStatus[1] == "lock") {
     lock = <img src={lockLogo} style={{ height: 200, width: 200
}}/>
     } else if (lockStatus[1] == "unlock"){
     lock = <img src={unlockLogo} style={{ height: 200, width:
200 }}/>
     } else {
     lock = <img src={tamperLogo} style={{ height: 200, width:
200 }}/>
     }

     if (landscape == true) {
     return (
```

```
     <div style={{ height: height * 0.9, width: width * 0.5,
backgroundColor: '#CDCDCD', display: 'flex', alignItems:
'center', justifyContent: 'center' }}>
          <div onClick={async () => {
          fetch('http://174.138.127.14:8080/api/delete-history')
               .then(updateLockHistory([]))
          }} style={{ boxShadow: shadow, backgroundColor:
'#E5E5E5', height: 200, width: 200, borderRadius: 200,
alignContent: 'center', justifyContent: 'center', alignItems:
'center' }}>
          {lock}
          </div>
     </div>
     );
     } else {
     return (
     <div style={{ height: height * 0.5, width: width,
backgroundColor: '#CDCDCD', display: 'flex', alignItems:
'center', justifyContent: 'center' }}>
          <div onClick={async () => {
          fetch('http://174.138.127.14:8080/api/delete-history')
               .then(updateLockHistory([]))
          }} style={{ boxShadow: shadow, backgroundColor:
'#E5E5E5', height: 200, width: 200, borderRadius: 200,
alignContent: 'center', justifyContent: 'center', alignItems:
'center' }}>
          {lock}
          </div>
     </div>
     );
     }
  }

  // console.log(test_api[0]["event"])

  useEffect(() => {
     console.log("first time setup")
     fetch('http://174.138.127.14:8080/api/get-history')
     .then(response => response.json())
     .then((data) => {
     updateLockHistory(data.reverse());
```

```
    })
  }, []);

  useEffect(() => {

    // Find connected or disconnected
    let connection_status = "connect"
    for (let i = 0; i < lockHistory.length; i++) {
    if ((lockHistory[i]["event"] == "disconnect") ||
(lockHistory[i]["event"] == "connect")) {
    connection_status = lockHistory[i]["event"];
    break;
    }
    }

    let lock_status = "lock"
    for (let i = 0; i < lockHistory.length; i++) {
    if ((lockHistory[i]["event"] == "lock") ||
(lockHistory[i]["event"] == "unlock") ||
(lockHistory[i]["event"] == "tamper")) {
    lock_status = lockHistory[i]["event"];
    break;
    }
    }

    setCurentLock([connection_status, lock_status])

  }, [lockHistory]);

  if (landscape == true) {
    return (
    <div>
    <div style={{ height: height * 0.1, width: width,
backgroundColor: 'white', borderBottom: 5, borderBottomColor:
'black', display: 'flex', alignItems: 'center' }}>
          <Text style={{ width: width, fontSize: height * 0.05,
alignSelf: 'center', textAlign: 'center' }}>Guardian Lock</Text>
    </div>

    <div style={{ height: height * 0.9, width: width, display:
'flex', flexDirection: 'row' }}>
```

```
          <LockButton height={height} width={width}
lockStatus={currentLock} landscape={landscape}/>

          <div style={{ height: height * 0.9, width: width *
0.5, display: 'flex', flexDirection: 'column', backgroundColor:
'#8095AA' }}>
          <div style={{ height: height * 0.07, width: width *
0.497, backgroundColor: 'white', display: 'flex', alignSelf:
'center', border: '2px solid black', borderRadius: 8 }}>
               <Text style={{ fontSize: height * 0.05,
alignSelf: 'center', paddingLeft: 20 }}>
               History
               </Text>
          </div>

          <div style={{ height: height * 0.80, width: width *
0.48, maxHeight: height * 0.80, overflowY: 'scroll',
backgroundColor: '#D3D3D3', scrollBehavior: 'smooth', display:
'flex', flexDirection: 'column', padding: 2, scrollbarColor:
'black', alignSelf: 'center', marginTop: 10, marginBottom: 10,
border: '2px solid black', borderRadius: 8, boxShadow: '1px 2px
9px #333B44' }}>
               {lockHistory.map((alert, i) => (
               <TestingFunc key={i} width={width}
height={height} title={alert["event"]}
time={Date.parse(alert["time"])} date="Today" message="This is
the message." landscape={landscape}/> // Make sure to add a
unique key value.
               ))}
          </div>
          </div>
     </div>
     </div>
     );
  } else {
     return (
     <div>
     <div style={{ height: height * 0.08, width: width,
backgroundColor: 'white', borderBottom: 5, borderBottomColor:
'black', display: 'flex', alignItems: 'center' }}>
```

```
          <Text style={{ width: width, fontSize: height * 0.06,
alignSelf: 'center', textAlign: 'center' }}>Guardian Lock</Text>
     </div>

     <div style={{ height: height * 0.92, width: width, display:
'flex', flexDirection: 'column' }}>
          <LockButton height={height} width={width}
lockStatus={currentLock}/>

          <div style={{ height: height * 0.5, width: width,
display: 'flex', flexDirection: 'column', backgroundColor:
'#8095AA' }}>
          <div style={{ height: height * 0.07, width: width *
0.993, backgroundColor: 'white', display: 'flex', alignSelf:
'flex-start', border: '2px solid black', borderRadius: 8 }}>
               <Text style={{ fontSize: height * 0.04,
alignSelf: 'center', paddingLeft: 20 }}>
               History
               </Text>
          </div>

          <div style={{ height: height * 0.5, width: width *
0.95, maxHeight: height * 0.80, overflowY: 'scroll',
backgroundColor: '#D3D3D3', scrollBehavior: 'smooth', display:
'flex', flexDirection: 'column', padding: 5, scrollbarColor:
'black', alignSelf: 'center', marginTop: 10, marginBottom: 10,
border: '2px solid black', borderRadius: 8, boxShadow: '1px 2px
9px #333B44' }}>
               {lockHistory.map((alert, i) => (
               <TestingFunc key={i} width={width}
height={height} title={alert["event"]}
time={Date.parse(alert["time"])} date="Today" message="This is
the message." landscape={landscape}/> // Make sure to add a
unique key value.
               ))}
          </div>
          </div>
     </div>
     </div>
     );
  }
```

```
}
```