CHAMPLAIN COLLEGE | LCDi Leahy Center for Digital Investigation

175 Lakeside Ave, Room 300A
Burlington, Vermont 05401
Phone: (802)865-5744
Fax: (802)865-6446
http://www.lcdi.champlain.edu

4/16/2018

# Exploration Forensics

## Contents

## Introduction

Every year new devices are brought to market that claim radical things. Most of these devices are very situational leaving large areas of unmarked and undocumented forensic artifacts. This allow the LCDI to research and explore these devices based on what the manufacturer states. This semester, the Exploratory Forensics team is looking at more devices to check for credibility.

The exploration forensics team will gather evidence on how devices and applications, that claim to test for paranormal activity, operate. The Ovilus V device and the free mobile apps for the Paranormal Puck 2 will be examined to determine the legitimacy of what the manufacturer claims.

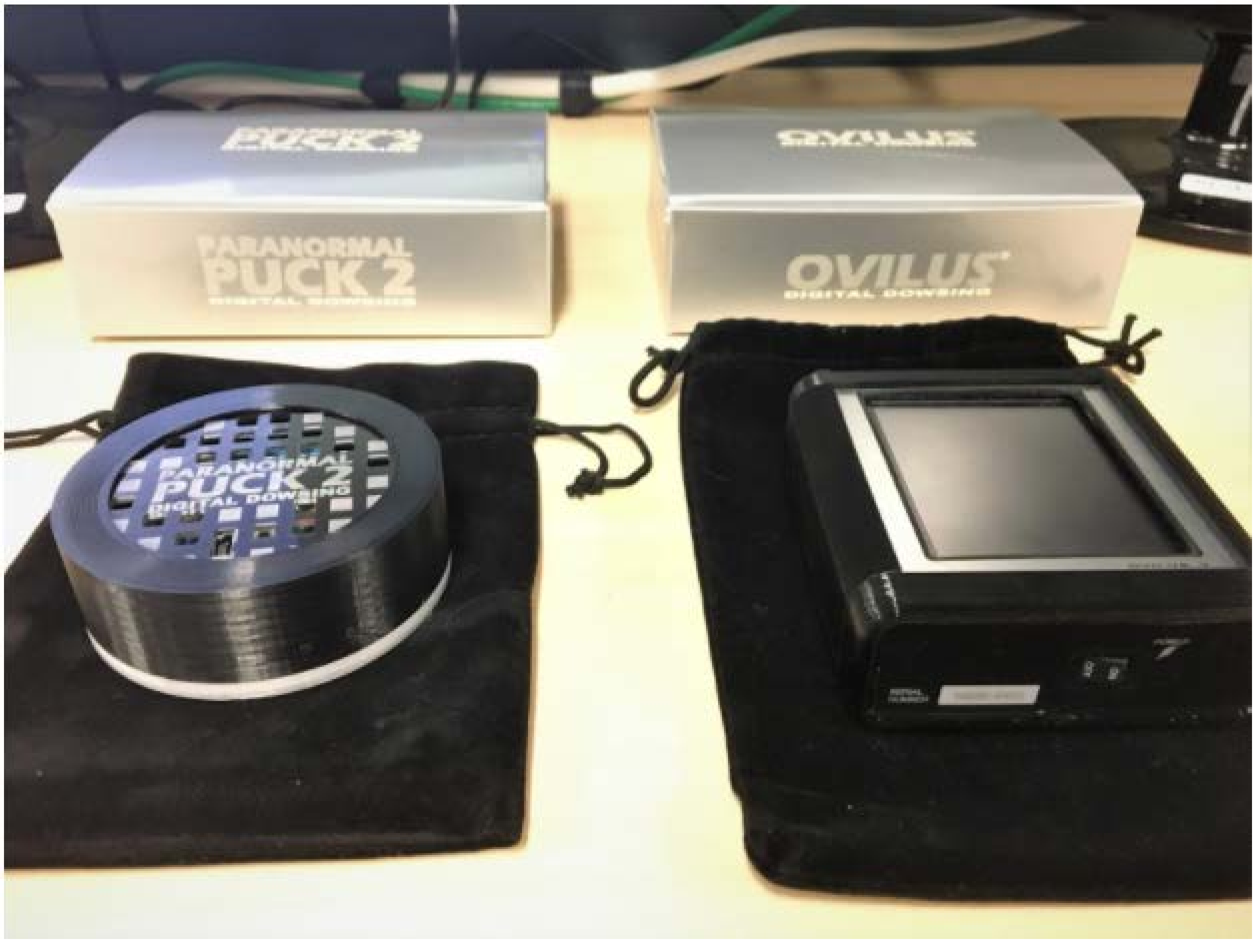


Figure 1

## Background

While there are no scholarly sources about most ghost-hunting devices, there are various online forum collections discussing the effectiveness of these devices. The Ovilus V and Paranormal Puck are products of Digital Dowsing, LLC. The company advertises the devices to be an effective way to communicate with ghosts. Digital Dowsing's website also outlines the ability of the devices to convert environmental factors such as temperature, humidity, barometric readings, electromagnetic fields, and movement into words. The team used information from the website and online forums to develop the research questions and research methods.

## Purpose and Scope

The team will research how the devices work and conduct a forensic investigation into the software and hardware in order to establish the legitimacy of the claims by Digital Dowsing.

While we don't intend to investigate the presence of paranormal activity, there will be research conducted on how the device and corresponding mobile applications convert environmental readings into words. In addition, we will gain understanding in extracting application files from phones and tablets, and how to analyze the decompiled files.

### Research Questions

1. How accurate are the sensors on the devices?
2. What information can be gathered from monitoring the activity on the devices via a USB connection/USB sniffing?
3. How easily can the sensors be manipulated in order to produce specific desired results?
4. What information can be found from investigating the devices via data mining?
5. What are the security permissions of the mobile applications and how is that information used for the devices to operate?

## Terminology

**First** - Terms listed in alphabetical order.

*APK file* - the package archive file that stores information for the installation of an Android application and middleware (Montegriffo).

*Application Decompiling* - the act of using software to convert an executable file, like an .ipa file, into source code. This code should be able to be recompiled into the original file ("Decompiler").

*BlueJ* - a free integrated Java integrated development environment.

*Electromagnetic Interference* - xxx. This is reflected in the Ovilus V.

*IntelliJ* - a robust Java integrated development environment.

*.ipa file* - the archive file that stores an IOS application with the ARM architecture.

*JadX* - a java decompiler for use with APK files.

*Jailbreaking* - the modification of smartphones or other electronic devices to remove restrictions in order to allow unauthorized software to be installed (iOS jailbreaking).

*Ovilus V* - device that converts environmental factors into words and phrases, supposedly from the paranormal activity in the area.

*Magnetic fields* - electric charge associated with magnets.

*Notepad++* - this software available for  Windows is used by the team to analyze the decompiled APK file.

*PuTTY* - an extension to the command terminal that allows users to use SSH protocols to connect to serial ports of another device (Ylonen).

*Radare2* - framework used from the command line for reverse-engineering and analyzing binaries. The Exploration Forensics team used this

*Reverse engineering* -  the reproduction of a software from analysis of its components and operation.

*Rooting* - the modification of smartphones or other electronic devices to remove restrictions in order to allow unauthorized software to be installed. Rooting gives administrative controls to the user ("Rooting (Android)").

*Smali* - an android debugging plugin for use with android compilers to view code as it is executed.

*USBPcap* - an open-source USB sniffer

*WinSCP* - software used for file transfer between a local device and a computer (Martin).

*Wireshark* - program used to track and analyze packets sent over a network.

# Methodology and Methods

## Apple Application

First, the team will jailbreak the iPad by using the Phoenix software. After obtaining the .ipa file of the free application from the iPad through the use of the Cydia extension *ipainstaller*, the application file can be copied to a computer to analyze the contents. The assembly code will be analyzed with *Radare2* to investigate the credibility of the sensors, what sensors are used, and how the application operates. Knowledge of assembly language will be needed.

## Android Application

After obtaining the APK file from the Google tablet through the use of an APK extractor found on the Google Play store, the specific application file can be Uploaded to a Google Drive account. The team will then use software to decompile and analyze the APK file. Knowledge of object-oriented languages like Java will be needed.

## Paranormal Puck 2

The team originally started the project with the intent to investigate both the Ovilus V and the Paranormal Puck, which are both products of Digital Dowsing, LLC. The team used the first couple of weeks of experimentation on the Puck, testing it's capabilities and investigating the corresponding app. From the beginning, there was a loose screw in the device. In week 5, the device suddenly stopped working. The team and LCDI supervisors believe that the screw damaged a part of the circuitry, which made it stop working. The device was returned and a new device was ordered, but it was so back-ordered that the team didn't get the replacement in time to conduct further research on it. The team still continued work on the mobile app, though, in order to solely investigate the claims specifically of Digital Dowsing, LLC. In addition, the findings before the device stopped working are reported in this paper.

## Ovilus V

The team originally planned to utilize Wireshark to monitor how the device sent and received packets across a private network, but the newest version of the Ovilus has no WiFi capabilities. After receiving the device, the team found out that the network capabilities were removed in order to provide a larger speaker on the device. The team then shifted to wanting to sniff the USB connection when the device was plugged into a computer by using USBPcap with Wireshark. In addition, false positives were tested on the device's sensors. The team will attempt to manipulate the magnetic and vibration sensors to see if there is a difference in how the device interprets the false positives to normal readings (Figure 2). Through the utilization of magnets, we hope to affect the readings of the Ovilus.

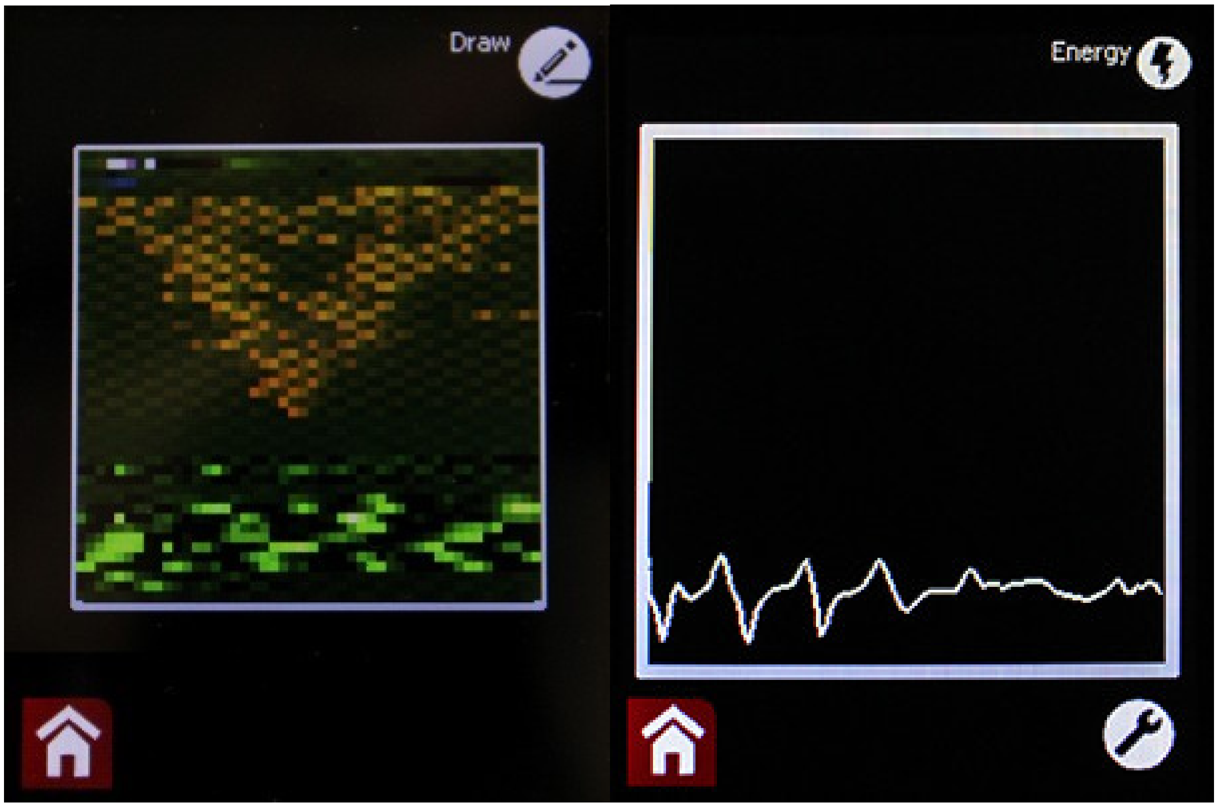


Figure 2 - Ovilus V Sample Readings

## Equipment Used

**Table 1: Devices used in Research**

| Devices | Purpose |
|---|---|
| Android Tablet | The Android tablet will be used to download and test the free applications. The APK files will be extracted from the device to be investigated on a computer. |
| Computer with Ubuntu VM | Using Radare2 to reverse engineer iOS app. An Ubuntu VM was loaded onto a Windows computer as the main interface used to analyze the contents of the applications. |
| iPad 2 | The iPad will be used to download and test the free applications. The .ipa files will be extracted from the device to be investigated on a computer. |

| | |
|---|---|
| Ovilus V | Stand alone device to be analyzed. This device converts environmental readings into words. |
| Paranormal Puck 2b | Device with accompanying apps that will be analyzed. This device converts environmental readings into words. |
| Windows Computer | A computer to analyze the decompiled Android application through Notepad++ |

# Analysis

## Apple .ipa File Data Collection and Analysis

In the beginning of the research into the free applications, the team knew the .ipa files would need to be extracted from the iPad in order to analyze the file on Radare2, which is software used to reverse-engineer and analyze binaries. The .ipa is the official application format for Apple, which contains all files needed for the execution of the app. From the tutorials the team reviewed during the preliminary research period, it used to be easy to download .ipa files from the backup archives of Apple devices on iTunes. Due to recent updates and a focus on device-oriented experiences, Apple has eradicated this feature, making it difficult for researchers and reverse-engineers to analyze Apple application files.

After attempts to access the file system of the device from a computer through an SSH process, the team continued researching new programs and ways to obtain the .ipa file. This was a major roadblock for the team, as they didn't foresee issues with getting a copy of the .ipa file, and instead prepared for roadblocks in understanding the assembly code. Eventually, a forum was found with information about the *ipainstaller* program, which proved to work ("How to extract ipa"). Below are the instructions for what worked:

1. Use Phoenix to jailbreak the iPad. The team had to do this a couple of times as the jailbreak would sometimes be removed when the iPad was plugged into the computer.
2. Search on the Cydia jailbreak application on the iPad for available extensions. Search for "*ipainstaller*".
3. Launched the new application *ipainstaller*.
4. From the computer, launch *PuTTY*, which is used to SSH into the device. This allows users to navigate into the file system of the device as long as the username, password, and IP address are known.
5. Log on as an administrator.
6. Type the command "*ipainstaller -l*" to list all of the apps installed on the jailbroken iPad.
7. To extract the specific app, type the command "*ipainstaller -b <app bundle name>*".
8. Using *WinSCP*, a file transfer software, copy the .ipa file to the desktop or another known folder.
9. Unzip the .ipa file.

After extracting the .ipa file from the Apple device, the team was able to analyze the decompiled application file and see the contents of the .ipa file folder, which includes the files used by the app, from .jpegs to folders to .txt files. This also includes the backgrounds on the app, various icons for the app, the word list, etc. This is found under *Payload*, then the *.app directory*. In addition, the Info.plist file is an XML file that includes pertinent information about how the app runs. A guide found on github provided valuable information about how to decrypt the plist file (SnakeNinny, 2015).

The unpacked .ipa file contained an executable file that can be opened with *Radare2* to be analyzed. For the Puck application, the executable was named “Puck2 1.36”. The executable was navigated to in the Linux terminal, and the command “*r2 “Puck2 1.36”*” was performed in order to open the executable with radare. Next the executable was analyzed for use in radare via the “*aaa*” command. In order to facilitate an easier understanding of the disassembled executable, the command “*e asm.pseudo=true*” was executed, which allows *Radare2* to display simplified assembly code. Rather than displaying the individual movements of values across registers to represent addition, for example, pseudocode mode just displays the values being added with the addition operator. For example, in assembly, adding 2+5 would consist of the following commands: “*MOV R1, 2*”, which means “move the value ‘2’ to the first register”. Then, “*ADD R1, 3*”, meaning “add three to the value contained in the first register”. Thus, even simple addition problems consist of multiple unintuitive instructions in assembly. Pseudocode reduces the above instructions to “*R1=2+3*”. Using the “*afl*” command, the functions of the app were displayed, but the main function could not be located due to the obfuscation of function names as a result of disassembly.

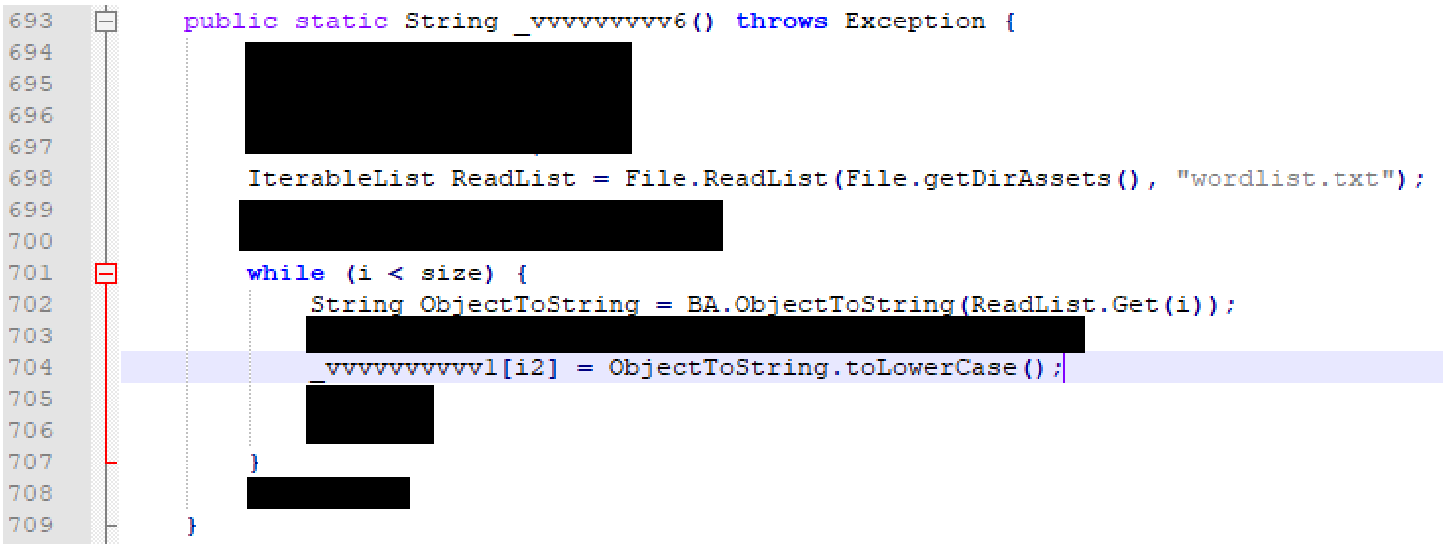
## Android APK File Data Collection and Analysis

In addition to decompiling the Paranormal Puck IOS application, the team also investigated the same app on the Google tablet, which runs through the Google Play Store.

1. Use *Nexus Root Toolkit*, which is specifically designed for Nexus devices, to root the tablet. Run the program after enabling USB debugging unlocked the tablet.
2. From the Google Play store, download the software called Apk Extractor, which will be used to extract the APK files and copy them to the phone’s file system/SD card (“Apk Extractor”)
3. Once the Apk Extractor application is downloaded, each specific application downloaded to the device can be searched for from the Apk Extractor application. Search for the application to be extracted.
4. Click the three dots to the right of the name of the specific app. Click share, then Share to Drive.
5. The APK file will now be available in the desired Google Drive.
6. This file will then need to be decompiled in software of your choice. The exploration forensics team used *jadx decompiler* for Android to decompile the APK file into java code.

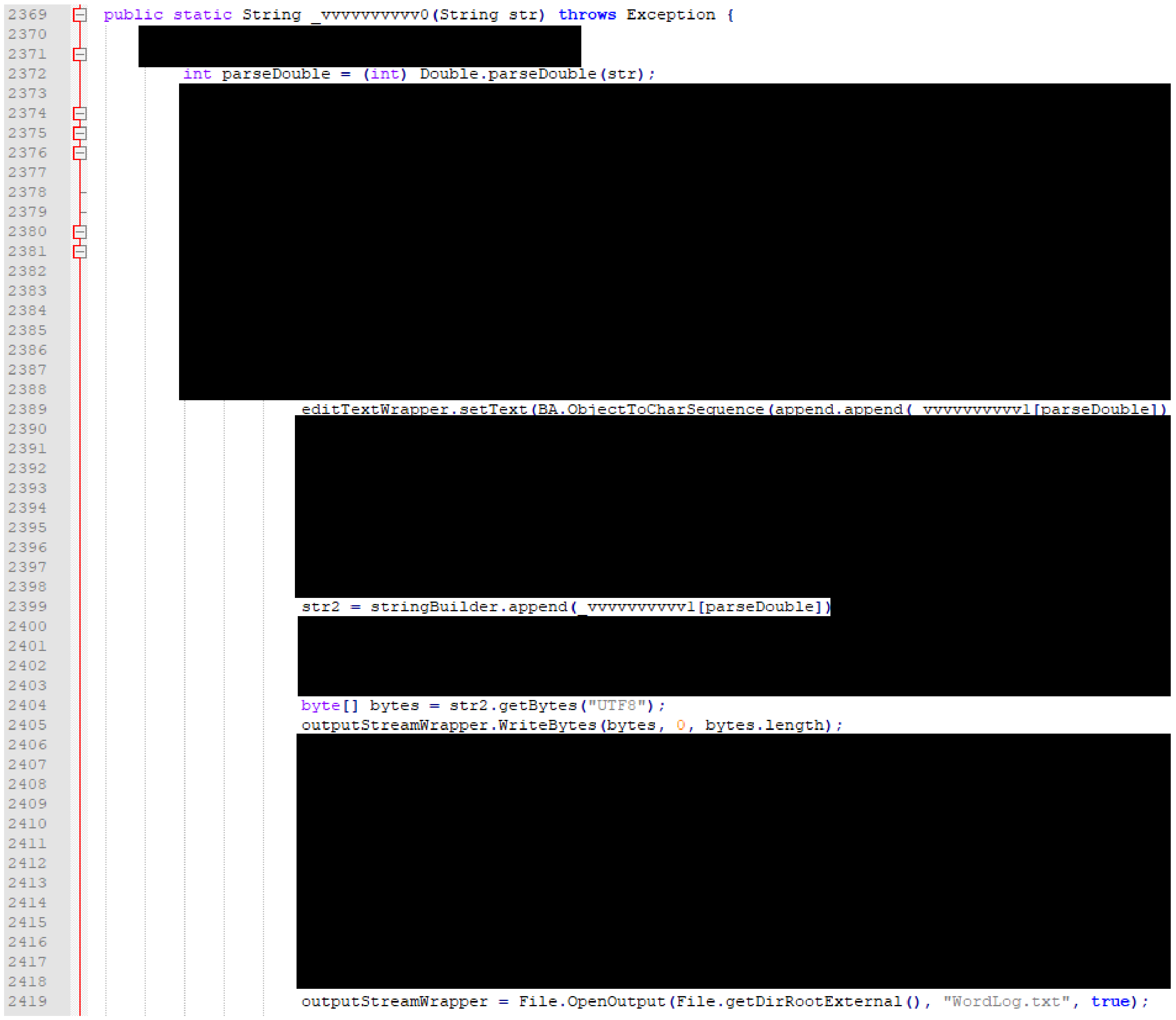
After the APK file from the Android device was decompiled, the team analyzed the decompiled file in various programs that would display the decompiled java code. These included IntelliJ, BlueJ, and Notepad++. In addition to the manual analysis of the decompiled code, the team tried using debuggers like Smali in order to see how applications execute instructions as it runs. Unfortunately, because of the nature of the Smali debugger, the Puck app was not able to be downloaded on virtualized android devices, and was not able to be debugged on the physical android device within the time constraints.

In the manual analysis of the decompiled code, the team searched the packages that make up the app, and found a java class titled “*main.java*”. This main class appeared to be the foremost file that dealt with the puck itself, rather than the visual interface of the app or its interface with the android operating system and user inputs. The team searched the class file for any mention of the wordlist. The wordlist stores all possible words that the Ovilus and Puck can output, and as such, any method that contains operations relating to the worldlist would most likely relate to the selection of words. The search found one instance of a reference to "*wordlist.txt*" in the method named "*_vvvvvvvvv6*" by the Jadx decompiler. After analysis of the method, it was concluded that the method read from the wordlist.txt and wrote the words contained therein into an array of strings, named "*_vvvvvvvvvv1*". Pertinent lines of the method are displayed below:



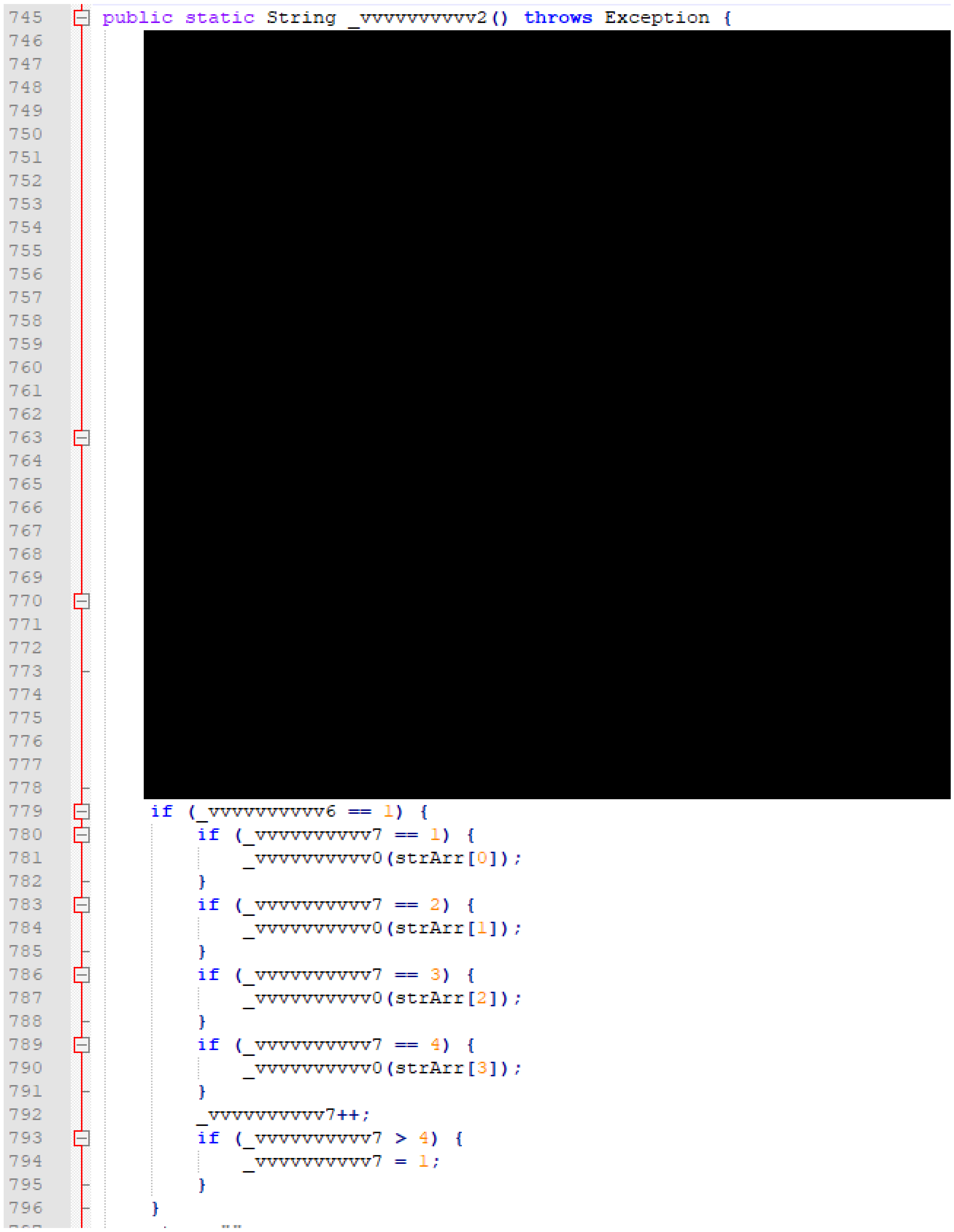
```
    public static String _vvvvvvvvv6() throws Exception {
        [redacted]
        IterableList ReadList = File.ReadList(File.getDirAssets(), "wordlist.txt");
        [redacted]
        while (i < size) {
            String ObjectToString = BA.ObjectToString(ReadList.Get(i));
            [redacted]
            _vvvvvvvvvv1[i2] = ObjectToString.toLowerCase();
            [redacted]
        }
        [redacted]
    }
```

This array of strings was searched for elsewhere in the class, and was found to be used in the method "*_vvvvvvvvvv0*", which is passed a string, designated "*str*" by the decompiler. The string is parsed into a numerical double value, "*parseDouble*". This double is referenced as the index of "*_vvvvvvvvvv1*", and the value at that index in "*_vvvvvvvvvv1*" is outputted as the word. This means that whatever the string passed to "*_vvvvvvvvvv0*" is the zero-based line number of the word that will be selected from the wordlist.txt. For example, "*HUMAN*" is the word on line 10 of the wordlist.txt, so if the string passed to "*_vvvvvvvvvv0*" is "*11*" (10+1), "*HUMAN*" is chosen as the outputted word. This functionality is displayed in the below screenshot of the method:



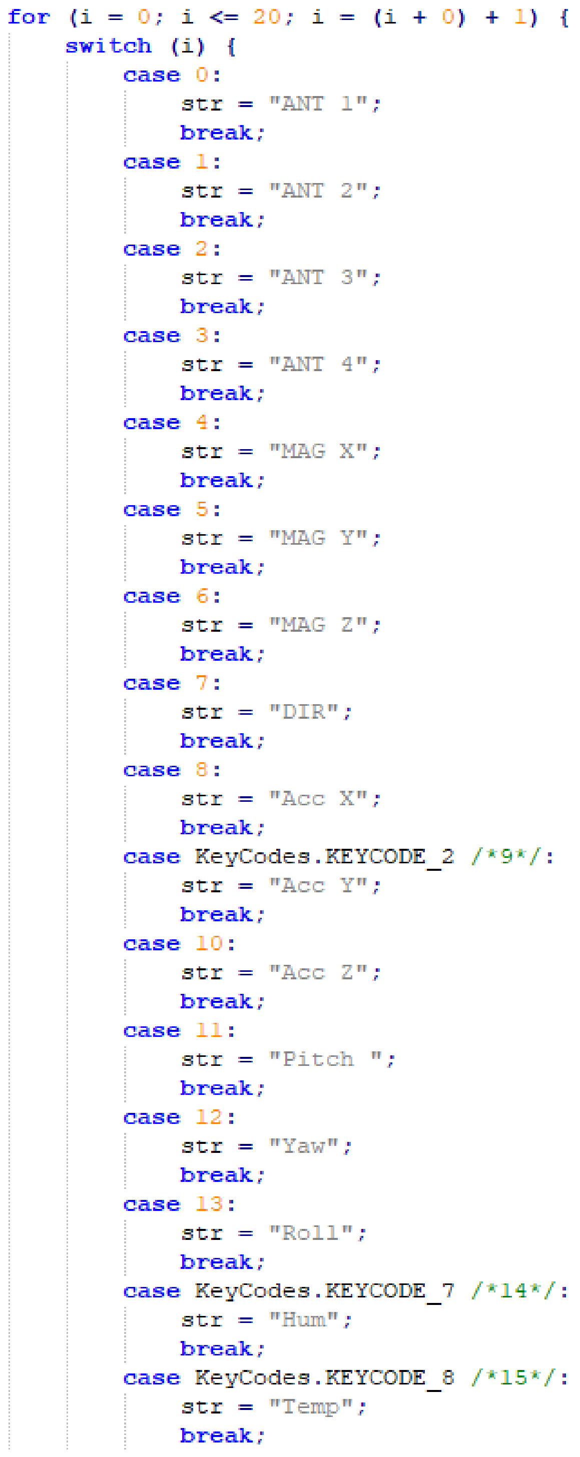
```
public static String _vvvvvvvvvv0(String str) throws Exception {
    [redacted]
        int parseDouble = (int) Double.parseDouble(str);
        [redacted]
                    editTextWrapper.setText(BA.ObjectToCharSequence(append.append(_vvvvvvvvvv1[parseDouble])
                    [redacted]
                    str2 = stringBuilder.append(_vvvvvvvvvv1[parseDouble])
                    [redacted]
                    byte[] bytes = str2.getBytes("UTF8");
                    outputStreamWrapper.WriteBytes(bytes, 0, bytes.length);
                    [redacted]
                    outputStreamWrapper = File.OpenOutput(File.getDirRootExternal(), "WordLog.txt", true);
```

After the analysis of this method, lines of code that called this method were searched for, as any call to this method would pass it the "*str*" value. The method "*_vvvvvvvvvv2*" includes four calls to the method, and, depending on the values of certain global variables, the index string of the chosen word is passed to "*_vvvvvvvvvv0*", as displayed below:



```
public static String _vvvvvvvvvvv2() throws Exception {
[redacted]
        if (_vvvvvvvvvvv6 == 1) {
            if (_vvvvvvvvvvv7 == 1) {
                _vvvvvvvvvvv0(strArr[0]);
            }
            if (_vvvvvvvvvvv7 == 2) {
                _vvvvvvvvvvv0(strArr[1]);
            }
            if (_vvvvvvvvvvv7 == 3) {
                _vvvvvvvvvvv0(strArr[2]);
            }
            if (_vvvvvvvvvvv7 == 4) {
                _vvvvvvvvvvv0(strArr[3]);
            }
            _vvvvvvvvvvv7++;
            if (_vvvvvvvvvvv7 > 4) {
                _vvvvvvvvvvv7 = 1;
            }
        }
```

Because of the length and complexity of the method, which spans 359 lines of decompiled code, the way in which the passed value is determined could not be elucidated within the time constraints. However, this method did contain references to sensors on the Puck, such as four ANT cases, three magnetometer and accelerometer cases for x, y, and z coordinates, pitch, yaw, and roll cases, as well as humidity, temperature, and pressure cases, as displayed below:



```
for (i = 0; i <= 20; i = (i + 0) + 1) {
    switch (i) {
        case 0:
            str = "ANT 1";
            break;
        case 1:
            str = "ANT 2";
            break;
        case 2:
            str = "ANT 3";
            break;
        case 3:
            str = "ANT 4";
            break;
        case 4:
            str = "MAG X";
            break;
        case 5:
            str = "MAG Y";
            break;
        case 6:
            str = "MAG Z";
            break;
        case 7:
            str = "DIR";
            break;
        case 8:
            str = "Acc X";
            break;
        case KeyCodes.KEYCODE_2 /*9*/:
            str = "Acc Y";
            break;
        case 10:
            str = "Acc Z";
            break;
        case 11:
            str = "Pitch ";
            break;
        case 12:
            str = "Yaw";
            break;
        case 13:
            str = "Roll";
            break;
        case KeyCodes.KEYCODE_7 /*14*/:
            str = "Hum";
            break;
        case KeyCodes.KEYCODE_8 /*15*/:
            str = "Temp";
            break;
```

## Paranormal Puck 2 Physical Device Analysis

The team was able to do a preliminary investigation into the Paranormal Puck device before it stopped working. We observed how the Apple application operates in conjunction with the device. The app would output responses, somewhat randomly, it seemed. There were differences in energy readings when the device was a few feet from any other device, and when a team member would move a phone near it. In addition, the Puck required access to the tablet's camera in order to visualize information on the application. When the team disabled permissions for the camera, the app would crash each time it was booted, until camera access was allowed again. The app is set to take pictures, very randomly it seemed, whenever the environmental readings hit a certain point, supposedly to capture pictures of paranormal activity. The team wasn't able to investigate why photos were taken at these times.

The team was not able to conduct an official experiment to test these outputs before the device fell apart. There had been a loose screw from the first time the team used the device, which could be heard rattling inside the device. When it stopped working, the casing easily popped off without that screw in tact. The team took the screw out and put the device back together after examining the circuitry. Various sensors were found in the circuitry, which confirmed Digital Dowsings claims of the device using environmental readings. Refer to the appendix to see pictures of the circuitry in the Puck. With the Puck, the team found:

- A Bluetooth chip,
- An AU1716 chip. With further investigation online, this chip looks like a processing chip with read and write capabilities, with combined processing and flash memory. This is the black chip on Figure 4(D) labeled ATMEL mega 32 8 p AU1716
- A temperature and humidity sensor. This is the light blue mesh chip on Figure 3(B) labeled DHT-11.
- A gyroscope chip, or motion sensor. This is the dark blue chip on Figure 3(A) labelled INVENSENSE MPU 6050 \n 03G242-a1 \n 1411-E.

## The Ovilus V Device Analysis

The project surrounding the Ovilus, as mentioned earlier, was road blocked by the change in the newest Ovilus. Older Ovilus versions had WiFi capabilities, and the team hoped to use Wireshark to monitor how the device interacted with the internet network. While the Puck had its problems, the team had to improvise with the Ovilus after finding out about some hardware modifications. After having to scrap the original plan of sniffing packets through a network, the team tried to sniff USB traffic using USBPcap with Wireshark. By doing that we found another hiccup that kept us from continuing, which was that the device needed to be mounted to be examined with USBPcap, which kept us from examining the device while it was in use.

The device doesn't actually produce readings with units, and rather just displays unlabeled, visual, relative indicators, as can be seen in Figure 3. The team hoped to test the validity of the measurements, but this is not possible since the device doesn't produce concrete readings and instead is based off a combination of several different data sources. The team instead tested false positives, hoping to create false readings from the interference of magnets. A magnet was taken from a broken hard drive, but various testing moving the magnet near the Ovilus V minorly affected the readings, but simply tapping the antenna provided a greater change than the magnets did.

## Results

As mentioned previously, the team's focus shifted throughout the project's timespan due to the roadblocks we encountered. When the Paranormal Puck 2 broke, the team was unable to conduct the main forensic research they had planned to. Before sending the device back to Digital Dowsing for a replacement, the team was able to look at the hardware of the Puck and found that there are many sensors to record environmental readings, like temperature and motion. This confirms that there are sensors in the device to sense the environment, but we weren't able to test how accurate these sensors are, like we planned to.

We instead focused on investigating the application files and conducting research into how the applications operate as we waited on a replacement Puck. We focused on the Paranormal Puck 2 app for both Apple and Android to continue the research, despite not having the physical device anymore. From this investigation, the team found leads within the Android app, but were unable to decipher the Apple assembly code. While the code that made up the Puck's executable for Apple was explored using the "*s*" command to seek functions, and the "*pdf*" command in order to read through the assembly code, due to the complexity of the app's operation, the team failed to interpret the assembly code in a meaningful way. For the Android app, the team was able to elucidate the process leading up to the selection of the outputted word, but the ways in which the value that is parsed into the outputted word was unable to be determined. While evidence does exist that indicates that sensor reading from the Puck's sensors are used to determine the outputted word, a concrete connection could not be discerned within the time constraints.

The team also investigated the Ovilus V device, without much success. When the project proposal for this device was submitted, the newest generation of the Ovilus hadn't been released. This newest generation of the Ovilus, which the team received, had no wifi capabilities, which is what the team originally planned to investigate. The previous Ovilus generation had a network card, which the team planned to set up to its own wifi network and investigate the packets the device sent over this network. Instead, the team tried to sniff the USB connection to see how the device interacted with a computer through a wired connection. This didn't yield any results. In addition, the false positive testing didn't provide much of a variance in readings on the Ovilus, which indicates the sensors aren't heavily affected by the interference.

## Conclusion

Due to the elimination of networking capabilities of the most recent generation of the Ovilus V, the original research plan for the device was not conducted. Further research indicated that false positives and intentionally manipulating the sensors does not heavily affect the results in any mode. This indicates that the sensors either take in data from such a large area that they aren't affected by intentional interference, or the they aren't particularly sensitive and could be giving muted readings. Since the device doesn't produce results with units or labels, there isn't a way to establish a quantitative accuracy of the sensors.

From the experimentation with the Paranormal Puck 2 before the device broke, the presence of the sensors in the hardware of the device was confirmed, but no tests to confirm the accuracy of them were conducted. The device does operate in conjunction with an app on an Apple or Android device, and it was found that the app requires camera permissions in order to function.

Before investigating the application file, the .ipa (Apple) or APK (Android) file has to be extracted from the tablet or smartphone, which requires jailbreaking the device, additional software to extract the application file, and moving the file to a computer. From there, software is used to decompile and analyze the application file, in order to understand what the coding instructions mean.

Through the investigation of the application for the Paranormal Puck 2, it was found that words are chosen from the word list based off the sensors' readings and converting that string argument to a double, which is then used as an index of the array of words in the wordlist.

## Further Work

With the many different areas the team did research in, there is a lot of room for future potential projects. The team hoped to investigate freeware applications in addition to the Paranormal Puck app to see the differences in security, sophistication of the sensors, and development of the apps.

In addition, the issues with the Paranormal Puck device presented large roadblocks to the basis of the team's project. In the future, it would be interesting to investigate the security of the information recorded by the device, how it interacts with the mobile app, and the operation of the device itself.

## Appendix

Any long tables, charts, etc will be placed here.

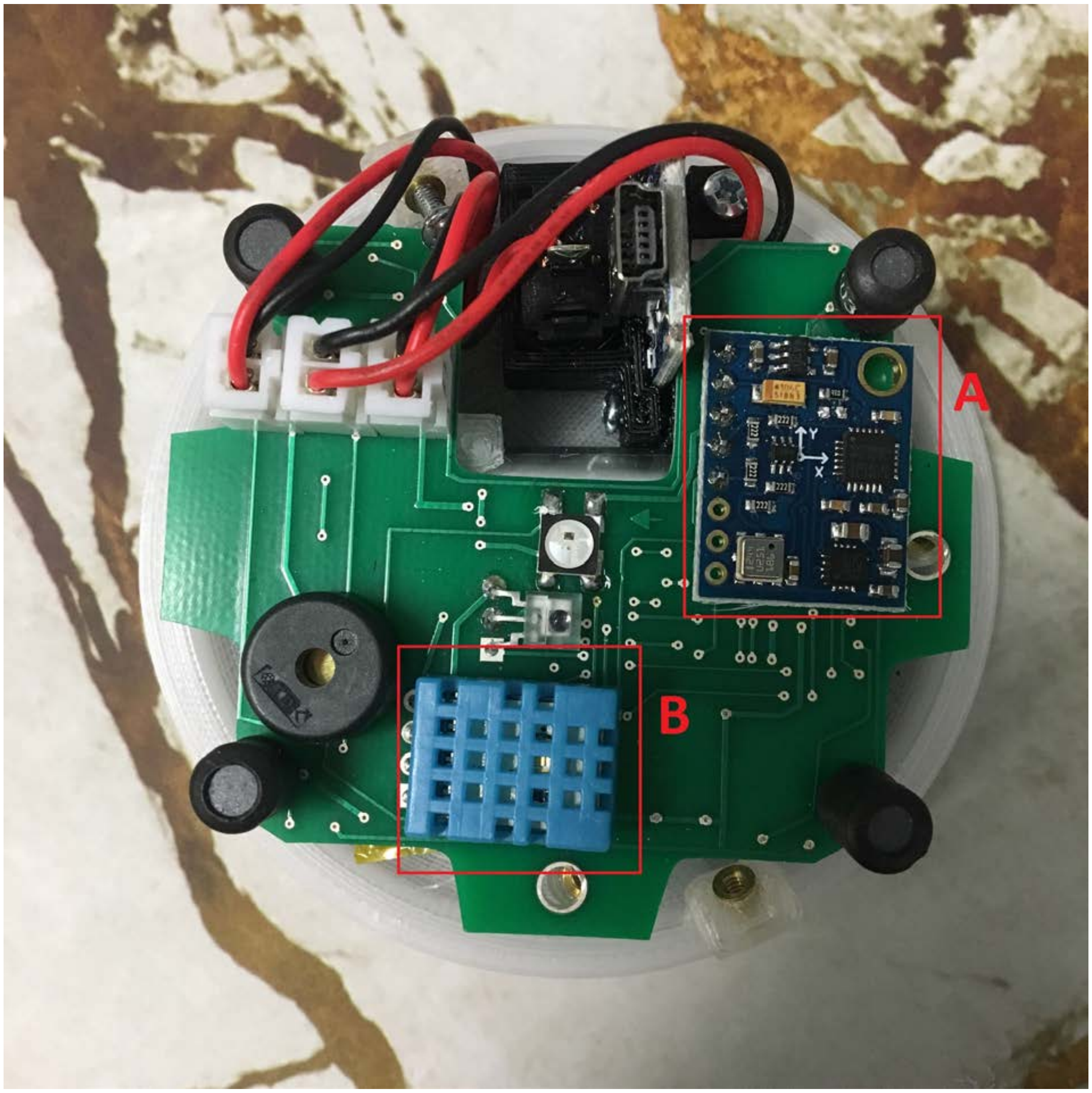


*Figure 3*

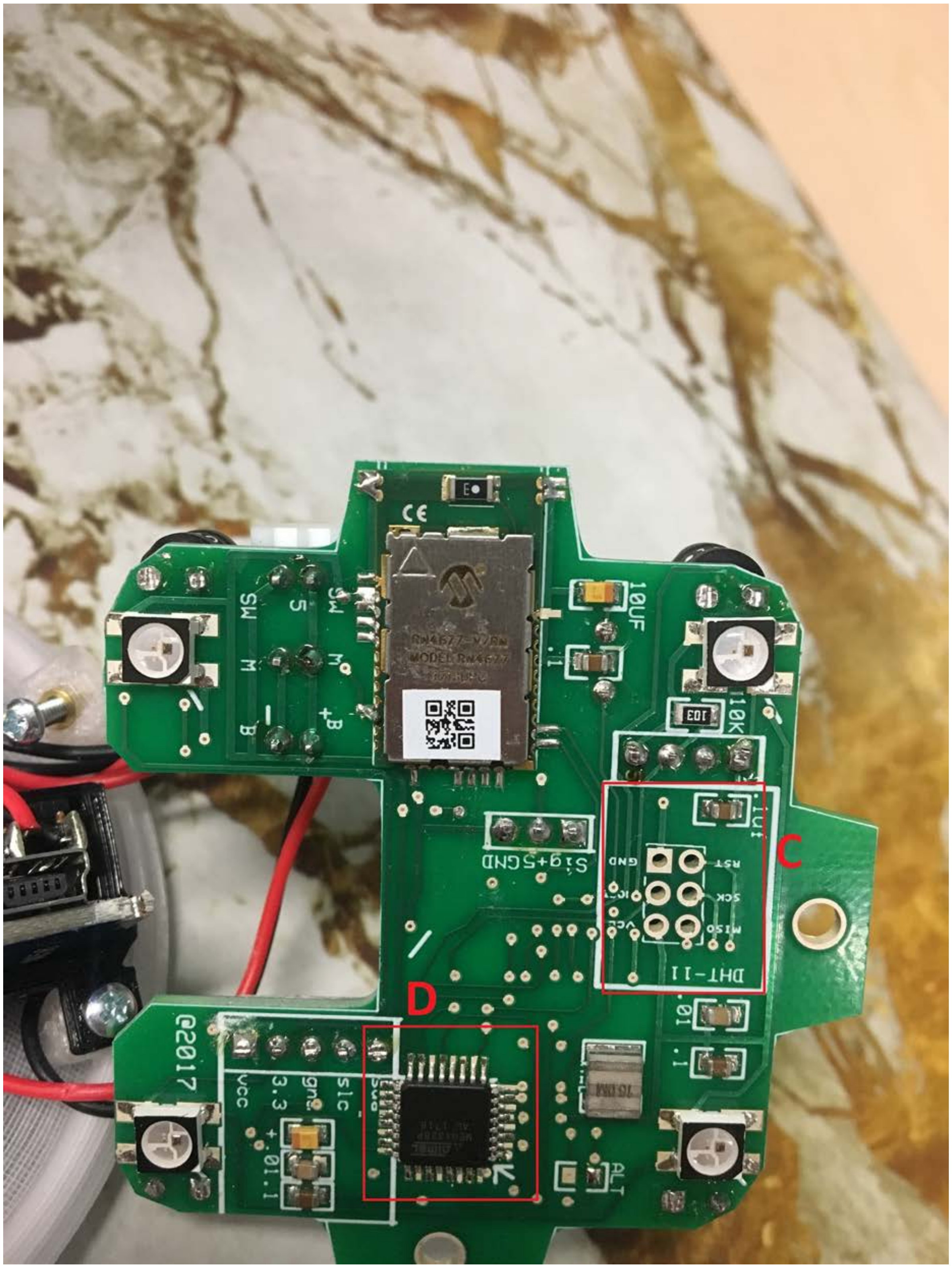

C
D
DHT-11
Sig+5GND
10UF
10K
@2017
vcc
3.3
gnd
slc
sda
GND
RST
SCK
MISO
VCC
ALT

*Figure 4*

## References

*Apk Extractor*. Meher, 2018. Vers. 4.2.6. *Google Play Store*, https://play.google.com/store/apps/details?id=com.ext.ui&hl=en

“Decompiler.” *Wikipedia*, Wikimedia Foundation, 28 Feb. 2018, en.wikipedia.org/wiki/Decompiler.

"iOS jailbreaking". *Wikipedia*, Wikimedia Foundation, 1 April 2018. https://en.wikipedia.org/wiki/IOS_jailbreaking.

“How to extract ipa file from jail broken device”. Stack Overflow web forum. 11 July 2017. *Stack Overflow*, https://stackoverflow.com/questions/40379318/how-to-extract-ipa-file-from-jail-broken-device.

Martin. “Introducing WinSCP.” *WinSCP*, WinSCP.net, 21 Dec. 2017, winscp.net/eng/docs/introduction.

Moń, Tomasz. “Open Source USB Packet Capture for Windows.” USBPcap, Tomasz Moń, Feb. 2013, desowin.org/usbpcap/.

Montegriffo, Nicholas. “What Is an APK File and How Do You Install One?” *AndroidPIT*, AndroidPIT International, Jan. 2018, www.androidpit.com/android-for-beginners-what-is-an-apk-file.

“Rooting (Android).” *Wikipedia*, Wikimedia Foundation, 12 Feb. 2018, en.wikipedia.org/wiki/Rooting_(Android).

SnakeNinny, & HangCom. (2015, September 15). IOS App Reverse Engineering (Z. Wu, 0xBBC, & F. Cheng, Trans.). Retrieved from https://github.com/iosre/iOSAppReverseEngineering/blob/master/iOSAppReverseEngineering.pdf

Ylonen, Tatu. “PuTTY - Graphical Terminal & SSH Client for Linux.” *SSH.com*, SSH Communications Security, Inc., 22 Aug. 2017, www.ssh.com/ssh/putty/linux/.