Progress Report May 2nd, 2019

Capstone: IoT Forensics
University: University of Nebraska at Omaha
Members: Elisabeth Henderson, Ashley Leedom, Amber Makovicka, Ronald Ramirez, & Nathan Wood

Overview

Milestone 3 was primarily concerned with conducting further research on IoT forensic frameworks and hands-on experimentation with various techniques of data collection. It became readily apparent that the field of IoT forensics was far less developed than that of traditional digital forensics, with minimal work available that defines prospective frameworks. In IoT forensics, instead of proposing traditional frameworks, researchers develop specific technology that addresses the various needs of a digital investigation. Emphasis is placed on gathering forensic data from IoT-based infrastructures and designing applications that can store, interpret, and report on the assembled evidence.

We started our hands on experiment with the Google Home Mini. We wanted to observe its network traffic, and view any logs available on the cloud platform. Unfortunately the network traffic is encrypted and not human readable. The cloud platform however, has logs available to developers and to law enforcement. Another approach that we took when analyzing the Google Home Mini was using a network monitoring tool developed by Princeton called IoT Inspector. This tool creates graphs from the data it observes, allowing us to see what the Google Home Mini interacts with such as advertisements domains.

The next technique employed was Bluetooth sniffing, which was done using Ubertooth One, an open source project. Sniffing was done between an Android phone and two IoT devices, a MetaWear CPRO and Garmin Vivosmart HR+. We were able to capture and analyze this traffic. The Metawear CPRO sent its packets in plaintext, but the Garmin used encryption. Even when in plaintext, however, the traffic was difficult to decipher.

The last technique employed involved a more traditional approach using FTK Imager, XRY, and Autopsy. We attempted to connect each device to FTK Imager, but only the Garmin was recognized. While we were not expecting any device to be recognized, we were able to gather a lot of data from the Garmin. We then used XRY to create an image of the Android phone that had been connected to each IoT device. In examining the image of the phone, we were able to pull some of the data that had been generated by the IoT devices.

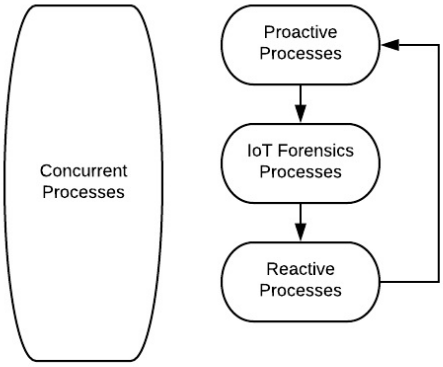
Outcomes

IoT Forensic Frameworks

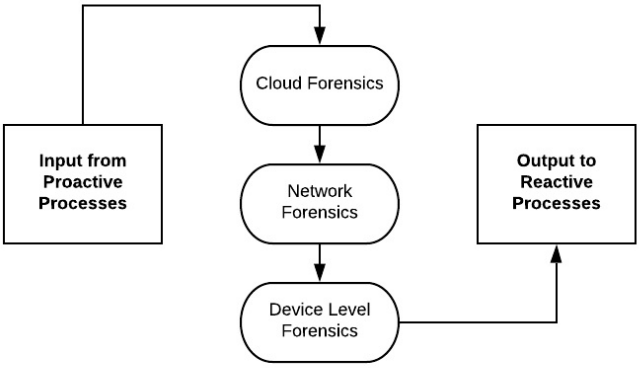
We expanded our IoT forensic framework research in the section below.

A Generic Digital Forensic Investigation Framework for Internet of Things (IoT)
Authors: Victor R. Kebande and Indrakshi Ray
Date Published: September 26, 2016
Published In: 2016 IEEE 4th International Conference on Future Internet of Things (IoT) and Cloud (FiCloud)
Summary: Kebande and Ray have proposed a generic IoT forensic investigation framework meant to guide digital investigations in IoT-based infrastructures. The framework complies with the ISO/IEC 27043:2015 international standard and provides a baseline for improvement in the field.
Framework: The authors propose the Digital Forensic Investigation Framework for the Internet of Things (DFIF-IoT). DFIF-IoT consists of three process blocks: proactive processes, IoT forensics processes, and reactive processes. Proactive processes represent the activities that prep an organization for a forensic investigation to take place. Common activities in this module are incident scenario definitions, evidence source identification, planning incident detection, potential digital evidence collection, and digital preservation. IoT forensics processes represent all the IoT-based aspects and infrastructures where digital evidence can be acquired from. Forensic investigators should have the technical skill and tools necessary to acquire data from IoT devices, networks, and cloud-based environments. Reactive processes represent the digital forensic investigation itself and are triggered after an incident has been detected. There are three stages in this module: initialization, acquisitive processes, and investigative processes. During initialization, investigators follow establish protocol for commencement of a digital forensic investigation. During the acquisitive processes, evidence is collected from data sources, transported, and stored securely. Finally, the investigative processes involve analyzing, interpreting, and reporting the gathered evidence.

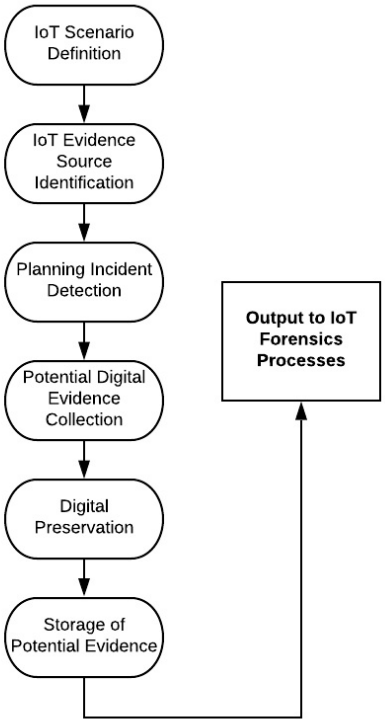
The diagrams below visualize the various processes involved in DFIF-IoT.



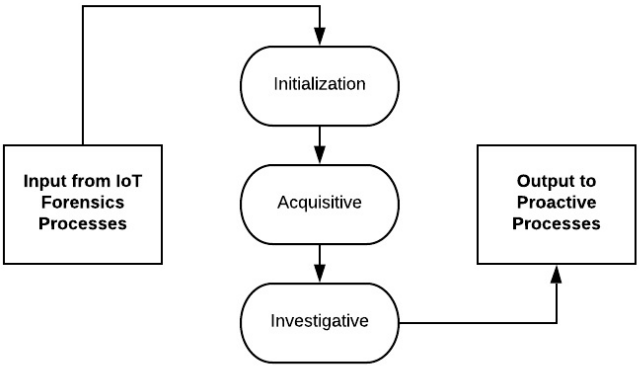
DFIF-IoT Overview



IoT Forensics Proceses



Proactive Processes



Reactive Processes

Concurrent Processes include obtaining authorization, documentation, managing information flow, preserving chain of custody, preserving digital evidence, and interaction with the physical investigation.

Proactive processes are primarily concerned with prepping an organization or smart environment to undergo a digital investigation. These processes could be considered optional, as an organization should continuously revise its security and incident response policies.

IoT Forensics processes are left intentionally generic by the authors, but entail the forensic methodologies investigators employ to conduct a digital investigation. Investigators should have procedures defined for extracting evidence from cloud-based, network-based, and IoT devices.

Reactive processes encompass the entirety of a digital investigation from commencement to reporting.

FIF-IoT: A Forensic Investigation Framework for IoT Using a Public Digital Ledger

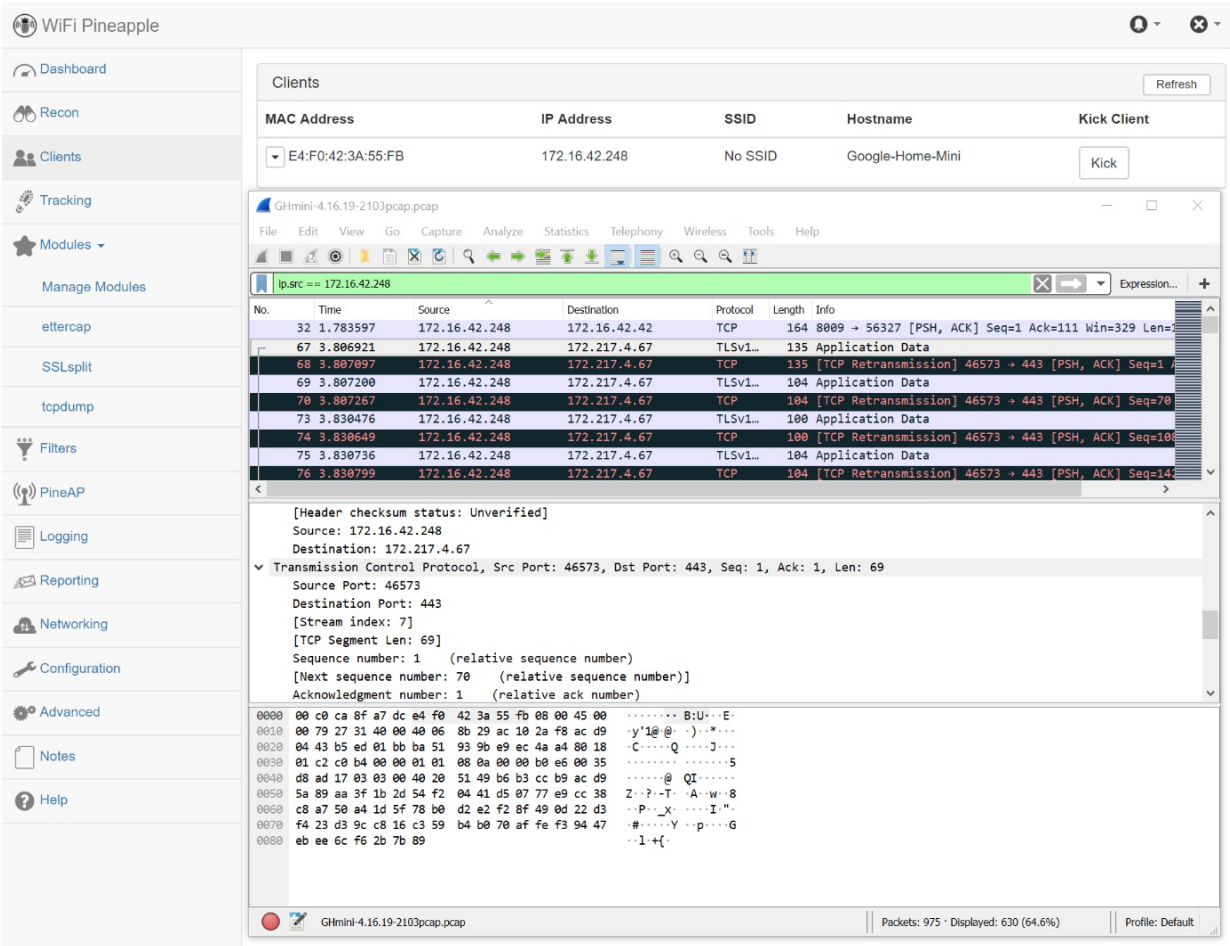
Authors: Mahmud Hossain, Yasser Karim, and Ragib Hasan
Date Published: September 27, 2018
Published In: 2018 IEEE International Congress on Internet of Things (ICIOT)
Summary: The authors have developed an IoT forensic framework that uses a public digital ledger to collect and store data pertaining to criminal incidents in an IoT-based system. FIF-IoT collects and stores interactions from an IoT network in a decentralized blockchain network. The purpose of FIF-IoT is to remove single points of failure in a digital forensic investigation process, as well as ensure the integrity, confidentiality, anonymity and non-repudiation of the gathered evidence.
Framework: FIF-IoT collects interactions between nodes located in an IoT-based system. FIF-IoT then creates transactions using the information gathered from those interactions. The transactions are sent to the public ledger network where the Miners use those transactions to create interaction blocks. These blocks are added to the blockchain in chronological order. Investigators can later refer to these interaction blocks to establish a timeline of the incident.

IoT Forensic: Bridging the Challenges in Digital Forensic and the Internet of Things
Authors: Nurul Huda Nik Zulkipli, Ahmed Alenezi, and Gary B. Wills
Date Published: December 2016
Published In: 2nd International Conference on Internet of Things, Big Data and Security
Summary: In this paper, the authors provide a comprehensive review of current IoT development trends, the most common threat vectors in IoT-based infrastructures, and the challenges inherent to constructing an IoT forensic framework.
Framework: The authors do not propose a framework, but seek to establish two general aspects of an investigation that should be considered essential components of IoT forensic frameworks moving forward. Those two items are pre-investigative readiness and real-time investigative solutions. As the name implies, the pre-investigative readiness process is primarily concerned with prepping an organization or environment to undergo a digital investigation. Managerial processes, technical procedures, and response plans must be defined before an incident occurs. The more interesting aspect of this proposal is the call for real-time investigative solutions. The authors propose that in the absence of a forensic standard and in response to the unique challenges present in IoT, technology should be used to gather, analyze, interpret, store, and present potential evidence. In IoT forensic investigations, technology will fulfill the duties that were formerly performed by digital investigators.

IoTDots: A Digital Forensics Framework for Smart Environments
Authors: Leonardo Babun, Amit Sikder, Abbas Acar, and A. Selcuk Uluagac
Date Published: September 3, 2018
Published In: ArXiv 2018
Summary: In this paper, the authors introduce IoTDots, a digital forensic framework designed to service smart environments. IoTDots is not a traditional framework, but instead a software solution to the lack of IoT platforms with strong digital forensic capabilities. Reportedly, IoTDots produces no overhead to the smart environments it's employed in, and only very minimal overhead to the cloud server it interacts with.
Framework: IoTDots functions by examining the activities of users and looking for security policy violations. IoTDots consists of two components: IoTDots-Modifier (ITM) and IoTDots-Analyzer (ITA). ITM analyzes the source code of applications in a smart environment, specifically looking for forensically-relevant information. If sensitive data is discovered, ITM modifies the application by inserting logs that send that data to the IoTDots Logs Database (ITLD). ITA then analyzes the logs present in the ITLD and constructs an accurate representation of the state of the smart environment and user behavior during that moment. From there, the detection of a security policy violation is possible. IoTDots can reliably identify instances where users try to remove, disable, or tamper with IoT devices present in the smart environment.

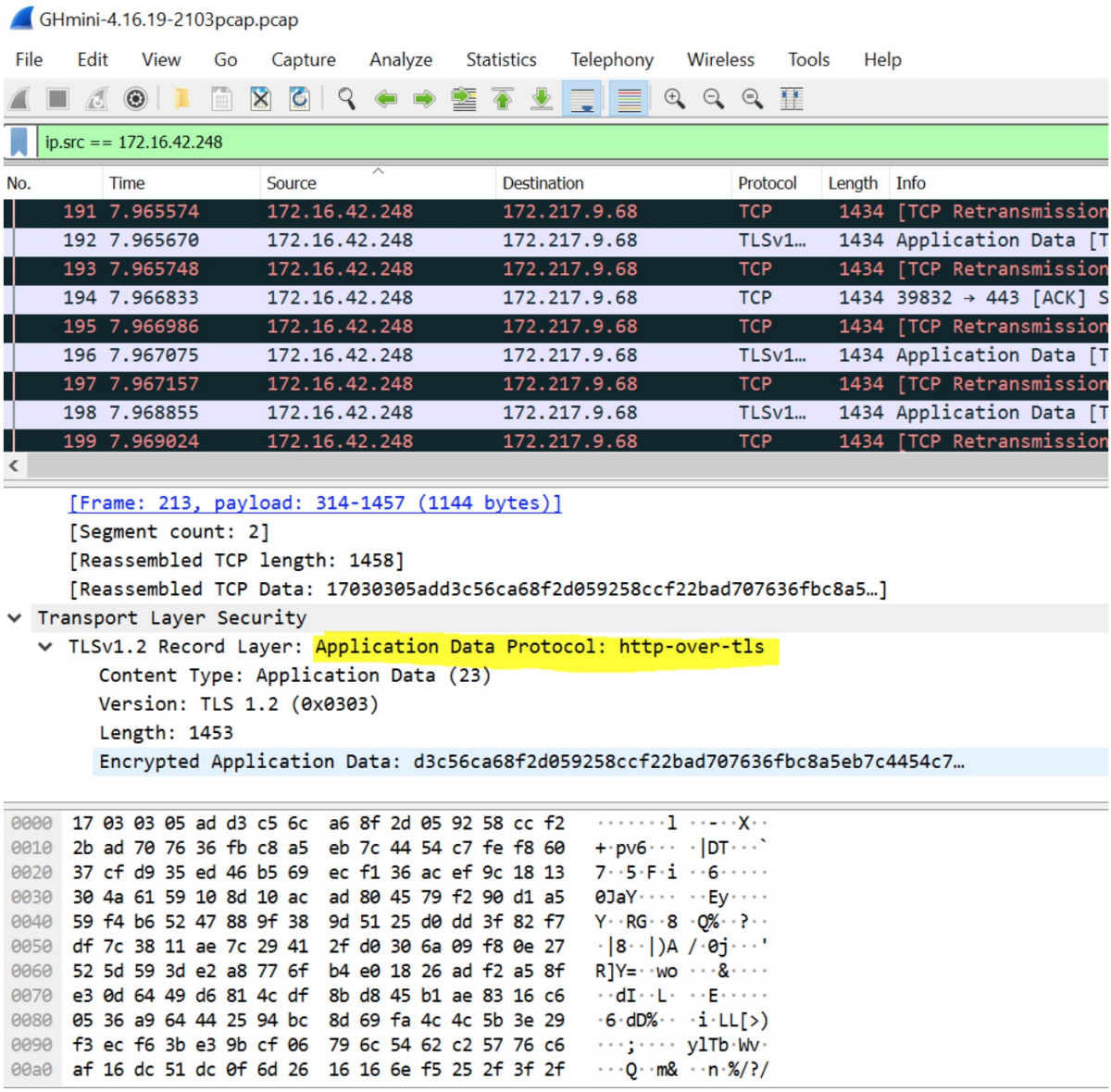
Hands on research with Google Home Mini

We setup a private WiFi network to conduct our study on the Google Home Mini IoT device. This was accomplished with a [WiFi Pineapple (https://wifipineapple.com/pages/nano)](https://wifipineapple.com/pages/nano) to broadcast the network and do a packet capture (pcap) on the IoT device.

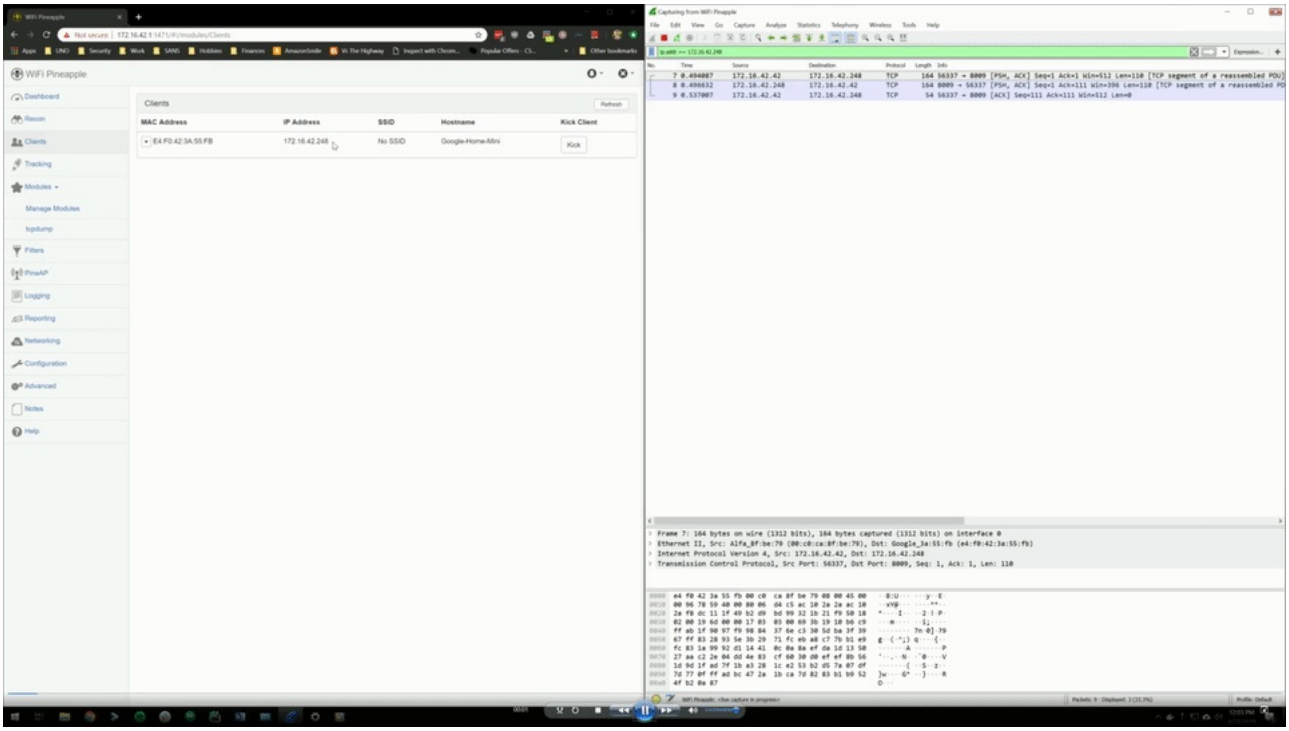


WiFi Pineapple = 172.16.42.1
Google Home Mini = 172.16.42.248

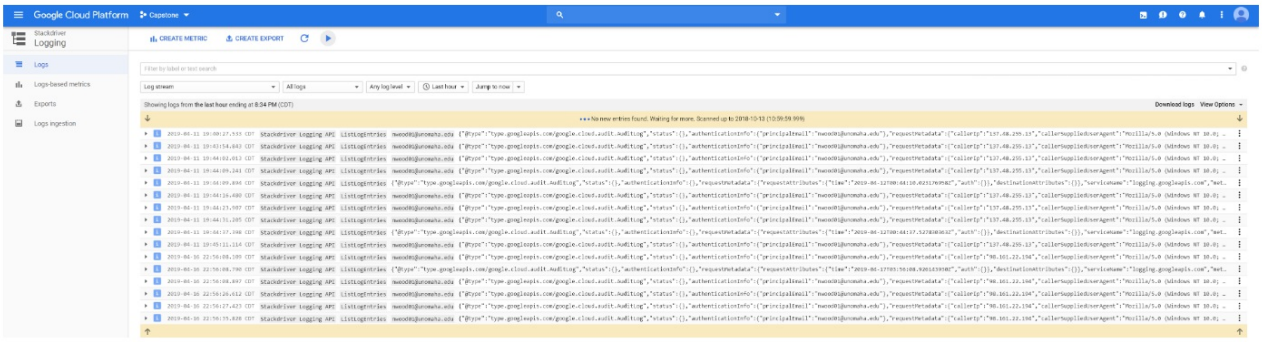
Unfortunately for our research the application traffic was encrypted, and we were unable to find valuable data from the pcap. The application data was sent via the http-over-tls protocol meaning it was encrypted before it was transmitted.



Video Capture:



[Link to full video (https://app.vidgrid.com/view/wbprCwUI0Xo3?public=1)](https://app.vidgrid.com/view/wbprCwUI0Xo3?public=1)
Luckily, we are able to access the [developer portal (https://developers.google.com/actions/smarthome/logging)](https://developers.google.com/actions/smarthome/logging) of the Google Home Mini - this allows us to see what a law enforcement officer might be able to subpoena for logs in a criminal investigation.



An example of what the logs look like from the Google Cloud Platform is available in our repo here: [GHMini-stackdriver-logs.json (/GHMini-stackdriver-logs.json)](/GHMini-stackdriver-logs.json). Logs are cleaned by Google to only share the failure type/error reason with the developer, not a detailed trace. Additional details on how to collect logs as a developer are available at the [developer portal (https://developers.google.com/actions/smarthome/logging)](https://developers.google.com/actions/smarthome/logging).

Google directs US based agencies to their [Transparency Report Help Center (https://support.google.com/transparencyreport/answer/7381738?hl=en)](https://support.google.com/transparencyreport/answer/7381738?hl=en). This page details what requirements are set for the legal process for user data requests. The website CAL-MASS [Google LE Guide (https://calmass.org/?wpdmpro=google-le-guide)](https://calmass.org/?wpdmpro=google-le-guide) article indicates there are exigent circumstances which allow law enforcement agencies to expedite the legal process to request user data.

Hands on research with Google Home Mini continued - Using IoT Inspector from Princeton

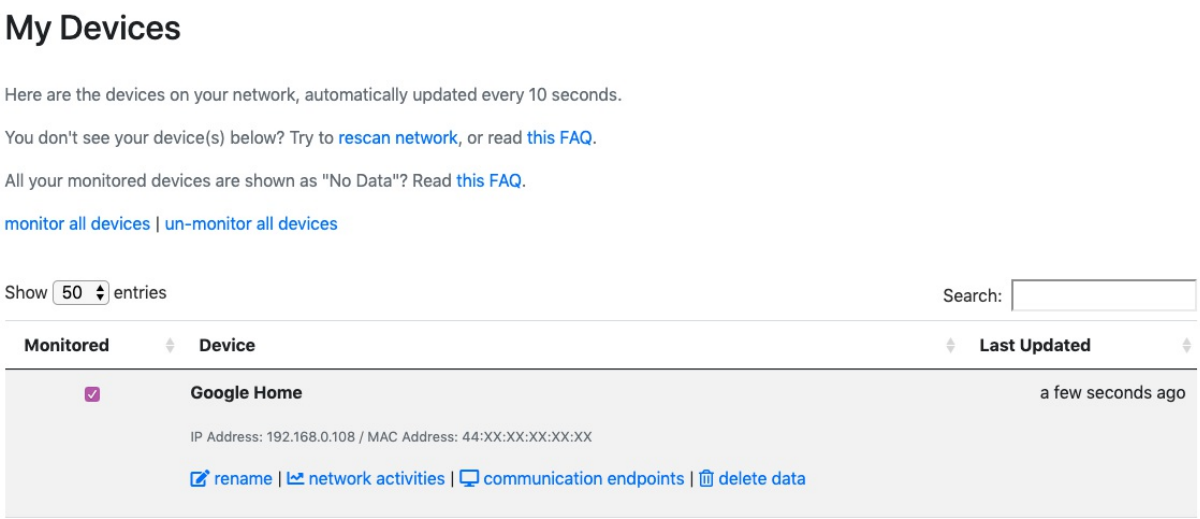
IoT Inspector (https://iot-inspector.princeton.edu/) is an open-source tool that inspects traffic through a browser, capturing all the devices connected on the network. This tool was developed by Princeton as a research initiative to help academic researchers as it can be difficult to produce generalizable results in the study of IoT security and privacy. We wanted to see what data this application could collect from the Google Home Mini. IoT Inspector presents traffic in the following views:

- Default view – displays all the traffic monitored
- Companies view – displays company names the device contacted
- Ads/Trackers view – displays ad/tracking services the device contacted
- No Encryption view – displays cases where device sent or received plain HTTP traffic
- Insecure Encryption view – displays cases where device interacted with Internet using insecure or outdated encryption
- Weak Encryption view – displays cases where device interacted with Internet using weak encryption

Below are a few images of the data that was captured while using the Google Home Mini for about 30 minutes. During these 30 minutes, we asked the Google Home Mini a variety of questions such as “What is the weather like today?” and also asked it to play music from YouTube.

Start Up View

The start up view is where you select devices to scan. Once you have allowed IoT Inspector to scan a device, its traffic will be visible in the views above.



Default View

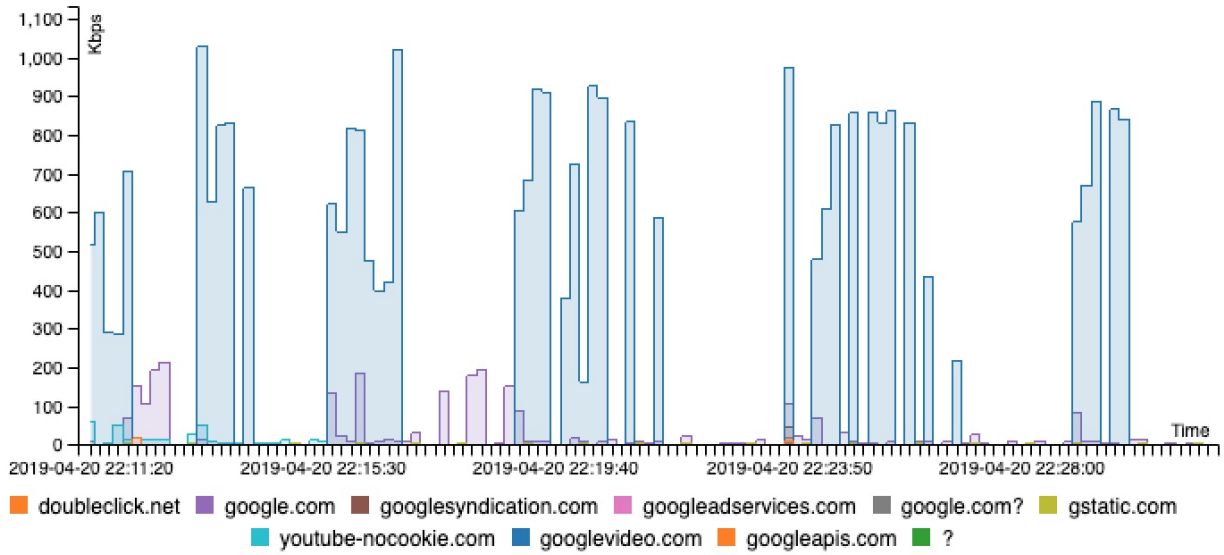
The default view shows the total traffic monitored during the duration of the scan.

Set view: default / companies / ads/trackers / no encryption / insecure encryption / weak encryption
Current view: **Default** — all my device traffic

Jump to: past 20 minutes / past 1 hour / past 24 hours / past week
Current zoom: **past 20 minutes, live chart**

Navigate: zoom in / zoom out / move left / move right

If you see a domain name with a question mark "?", this is the reason. If you see an empty chart below, see this FAQ.



Companies View

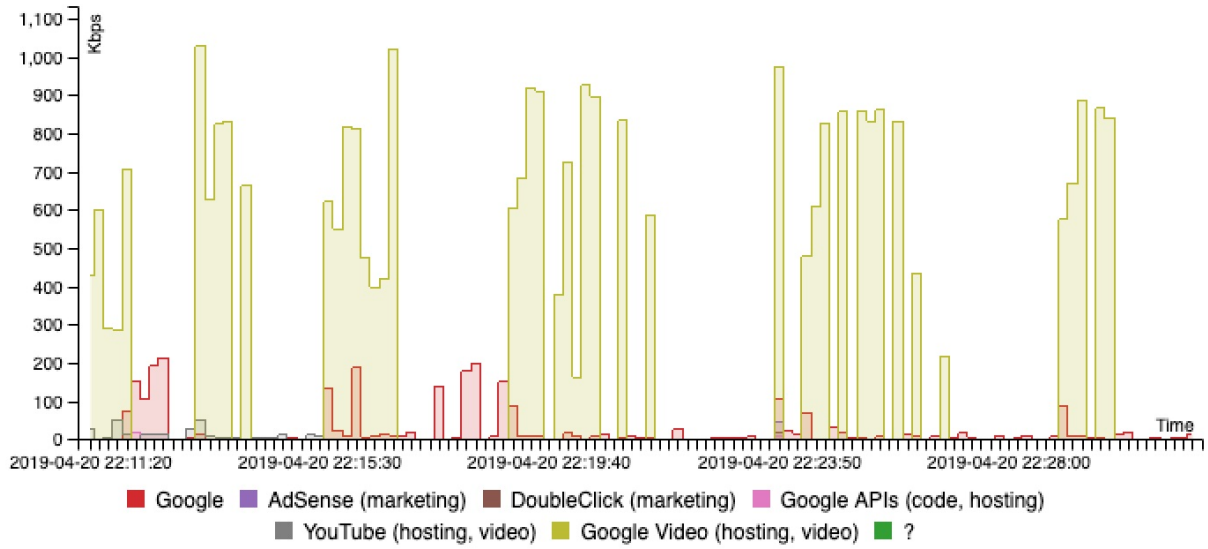
For the image below, you can see all the names of the companies the Google Home Mini interacted with.

Set view: default / companies / ads/trackers / no encryption / insecure encryption / weak encryption
Current view: **Companies** — all the companies that my device contacted

Jump to: past 20 minutes / past 1 hour / past 24 hours / past week
Current zoom: **past 20 minutes, live chart**

Navigate: zoom in / zoom out / move left / move right

If you see a domain name with a question mark "?", this is the reason. If you see an empty chart below, see this FAQ.



Ads/Trackers View

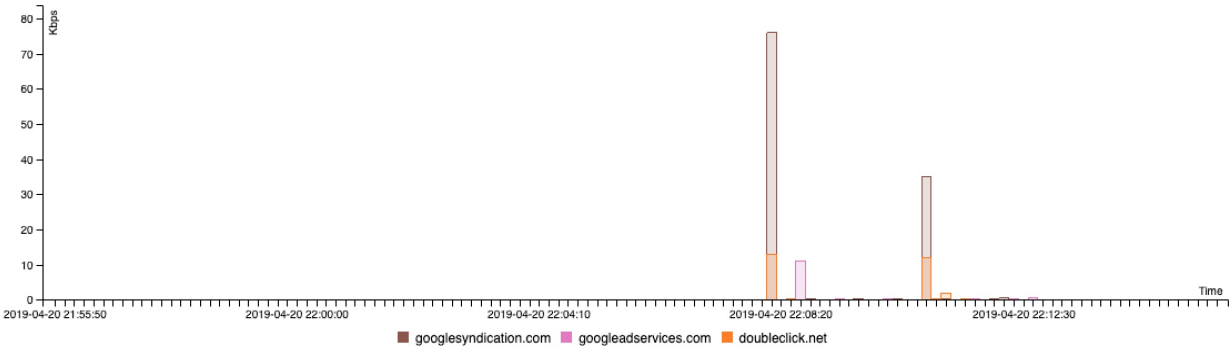
Below you will see the advertisement and tracker names that appeared, some of which occurred after the songs ended when playing music on YouTube.

Set view: default / companies / ads/trackers / no encryption / insecure encryption / weak encryption
Current view: **Ads/Trackers** — only showing cases where my device contacted ad/tracking services

Jump to: past 20 minutes / past 1 hour / past 24 hours / past week
Current zoom: **past 20 minutes, live chart**

Navigate: zoom in / zoom out / move left / move right

If you see a domain name with a question mark "?", this is the reason. If you see an empty chart below, see this FAQ.



Hands on research with Ubertooth

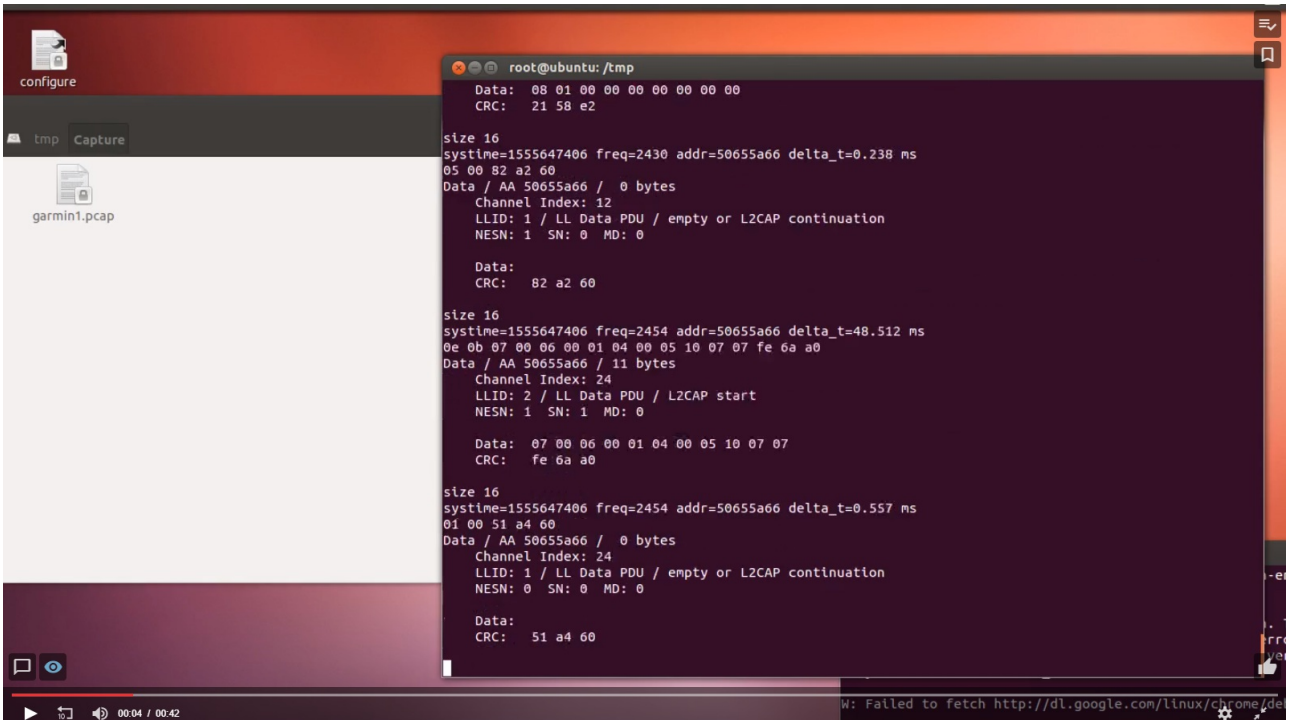
We set up an [Ubertooth One (https://github.com/greatscottgadgets/ubertooth/wiki/Ubertooth-One)](https://github.com/greatscottgadgets/ubertooth/wiki/Ubertooth-One) environment using an Ubuntu VM designed for Bluetooth sniffing, an obsolete, available Android phone, and IoT devices. The IoT devices used were a Garmin HR+ and Metawear CPRO. (The Android phone available was mostly arbitrary, since we were just interested in capturing the Bluetooth packets to analyze.)

Within the tmp/ folder we created a folder named "Capture/" to hold Packet Capture (PCAP) files generated to later open and analyze in Wireshark. To do this, the following command was used in the terminal while in the tmp/ folder: "sudo ubertooth-btle -f -c ./Capture/[device].pcap". The "-f" flag is used to follow connections, and "-c" is used to tell the command where to save the file with what name.

The video below was recorded during a capture of Bluetooth packets between the Android phone and Garmin HR+. Packets are captured very quickly and information on the screen runs by too quickly for a human to read, however when making a Bluetooth connection, the packets run by faster. Following the established connection, the scrolling of packets returns to the pattern it was before.

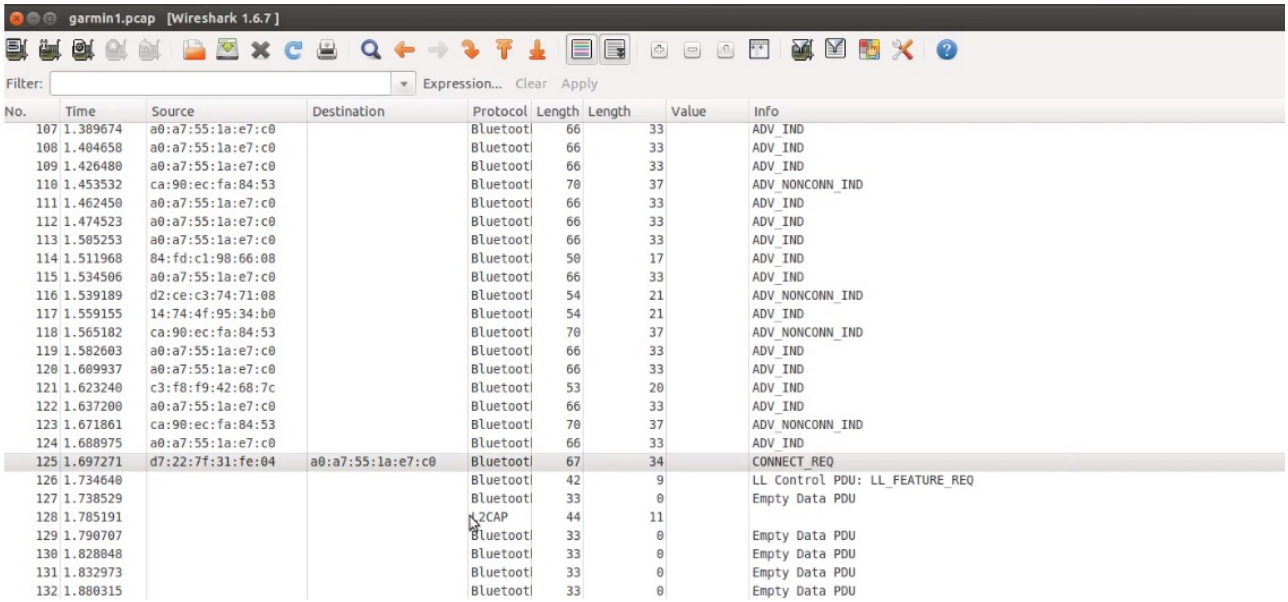
Video Capture:

(click to play)



[(https://use.vg/5wPFS7)](https://use.vg/5wPFS7)

At the end of the video, the file saved from the packet capture, 'garmin1.pcap' is opened in Wireshark. Using a string search, we can find the connection request packet, "CONNECT_REQ". This includes the MAC addresses of the devices used. It is worth noting the endian format of the MAC addresses are reversed when listed on Wireshark and on the phone/IoT devices. On a phone, it would list the Bluetooth MAC address as aa:bb:cc:dd:ee:ff, whereas on Wireshark (run on Ubuntu, little endian) it appears as ff:ee:dd:cc:bb:aa. This was only when looking at packets on an Ubuntu machine. On a Windows machine, MAC addresses were in aa:bb:cc:dd:ee:ff (big endian) order.



No.	Time	Source	Destination	Protocol	Length	Length	Value	Info
107	1.389674	a0:a7:55:1a:e7:c0		Bluetoot	66	33		ADV_IND
108	1.404658	a0:a7:55:1a:e7:c0		Bluetoot	66	33		ADV_IND
109	1.426480	a0:a7:55:1a:e7:c0		Bluetoot	66	33		ADV_IND
110	1.453532	ca:90:ec:fa:84:53		Bluetoot	70	37		ADV_NONCONN_IND
111	1.462450	a0:a7:55:1a:e7:c0		Bluetoot	66	33		ADV_IND
112	1.474523	a0:a7:55:1a:e7:c0		Bluetoot	66	33		ADV_IND
113	1.505253	a0:a7:55:1a:e7:c0		Bluetoot	66	33		ADV_IND
114	1.511968	84:fd:c1:98:66:08		Bluetoot	50	17		ADV_IND
115	1.534506	a0:a7:55:1a:e7:c0		Bluetoot	66	33		ADV_IND
116	1.539189	d2:ce:c3:74:71:08		Bluetoot	54	21		ADV_NONCONN_IND
117	1.559155	14:74:4f:95:34:b0		Bluetoot	54	21		ADV_IND
118	1.565182	ca:90:ec:fa:84:53		Bluetoot	70	37		ADV_NONCONN_IND
119	1.582603	a0:a7:55:1a:e7:c0		Bluetoot	66	33		ADV_IND
120	1.609937	a0:a7:55:1a:e7:c0		Bluetoot	66	33		ADV_IND
121	1.623240	c3:f8:f9:42:68:7c		Bluetoot	53	20		ADV_IND
122	1.637200	a0:a7:55:1a:e7:c0		Bluetoot	66	33		ADV_IND
123	1.671861	ca:90:ec:fa:84:53		Bluetoot	70	37		ADV_NONCONN_IND
124	1.688975	a0:a7:55:1a:e7:c0		Bluetoot	66	33		ADV_IND
125	1.697271	d7:22:7f:31:fe:04	a0:a7:55:1a:e7:c0	Bluetoot	67	34		CONNECT_REQ
126	1.734640			Bluetoot	42	9		LL Control PDU: LL_FEATURE_REQ
127	1.738529			Bluetoot	33	0		Empty Data PDU
128	1.785191			L2CAP	44	11		
129	1.790707			Bluetoot	33	0		Empty Data PDU
130	1.828048			Bluetoot	33	0		Empty Data PDU
131	1.832973			Bluetoot	33	0		Empty Data PDU
132	1.880315			Bluetoot	33	0		Empty Data PDU

Garmin Captures

One of the wearables we analyzed over Bluetooth was a Garmin Vivosmart HR+. Through trial and error, we recovered "CONNECT_REQ" packets between our wearable and Android phone when pairing. However, traffic back and forth from the devices showed little information. The communication between the two devices were encrypted and thus we were unable to read the packets. Therefore, we attempted to use [Crackle (https://github.com/mikeryan/crackle)](https://github.com/mikeryan/crackle) which is designed to crack Bluetooth Low Energy (BLE) Encryption. Crackle has two modes: Crack TK (Temporary Key) and Decrypt with LTK (Long Term Key). We attempted to use Crackle's Crack TK mode which was unsuccessful with each of the four .pcap files from the Garmin and Android phone. We used the following command sequence "./crackle -i [inputfile].pcap -o [inputresult].pcap".

```
root@ubuntu:/home/seat/Documents/crackle-master# ./crackle -i garminpcap/garmin4.pcap
Warning: No output file specified. Decrypted packets will be lost to the ether.
Found 3 connections

Analyzing connection 0:
  04:fe:31:7f:22:d7 (public) -> c0:e7:1a:55:a7:a0 (random)
  Found 7 encrypted packets
  Unable to crack due to the following error:
    Missing LL_ENC_REQ

Analyzing connection 1:
  04:fe:31:7f:22:d7 (public) -> c0:e7:1a:55:a7:a0 (random)
  Found 17 encrypted packets
  Unable to crack due to the following error:
    Missing both Mrand and Srand

Done, processed 0 total packets, decrypted 0
root@ubuntu:/home/seat/Documents/crackle-master#
```

An error occurred about missing packets, such as "LL_ENC_REQ" and "LL_ENC_RSP". Upon further inspection on the Crackle's [FAQ (https://github.com/mikeryan/crackle/blob/master/FAQ.md#crackle-is-complaining-about-missing-packets-why-cant-i-crack)](https://github.com/mikeryan/crackle/blob/master/FAQ.md#crackle-is-complaining-about-missing-packets-why-cant-i-crack) page, Crackle will not work on previously paired devices. If used on previously paired devices, a previously exchanged LTK will be used for secure communication, and there will be no key exchange. Unless the LTK is used to try and decrypt packets on Decrypt with LTK mode, we cannot get the key information. Another dilemma is that Ubertooth is not 100% guaranteed to capture 100% of packets. Even when a capture of a

"CONNECT_REQ" is found, it is not guaranteed to capture all of those packets over the air. In an attempt to try and use the Crackle tool again, a newly-factory reset Android phone was used to pair with the Garmin in an attempt to pair the devices and derive the LTK. However, it was a 1/3 chance the Ubertooth would be on the right channel to derive the necessary packets for Crackle. In addition, it was guaranteed all the necessary packets would be captured even on the right channel.

```
root@ubuntu:/home/seat/Documents/crackle-master# ./crackle -i garminpcap/newGarmin.pcap -o /ga
rminpcap/result.pcap
Found 1 connection

Analyzing connection 0:
  7c:2e:d2:3e:f2:fb (random) -> c0:e7:1a:55:a7:a0 (random)
  Found 2 encrypted packets
  Unable to crack due to the following errors:
    Missing one of Mrand and Srand
    Missing LL_ENC_RSP

Did not decrypt any packets, not writing a new PCAP
Done, processed 0 total packets, decrypted 0
root@ubuntu:/home/seat/Documents/crackle-master#
```

Even when using a factory-reset Android phone, the Ubertooth was unable to sniff out all necessary packets even on the right channel. The phone was factory-reset a second time to try again, but the results were similar. Out hands on testing with Crackle indicated it would be very difficult to utilize it in a forensic analysis of IoT devices. t is more feasible to procure a warrant for the Garmin device and then perform forensic analysis against that device.

Metawear Captures

For sniffing for packets between our Metawear CPRO device and Android phone, packets were easy to recover. Compared to the Garmin device, packets were not encrypted, and human readable. The CPRO device's Bluetooth uses the "Just works" protocol. Packets were in cleartext, but it was still a challenge to decipher the contents of the packets following the established connection between devices. In our experiment, we created three separate .pcap files related to the Metawear. One for turning on the red LED light and turning it off and repeating this for a total of three times. The other two worked similarly as the first, but for the green and blue LED lights. Please note that this is all the same LED light bulb, it can just change colors.

From our findings, we were able to decipher initialization, read requests, read responses, LED turn-on requests, and LED turn-off requests. As noted in the Garmin capture section above, the Ubertooth does not capture 100% of packets, and thus with the Metawear device's .pcap files, we could not discover all associated packets, but could still make interpretations.

What we were able to interpret from the Metawear .pcap files are shown below. The different hex values for turning on an LED light represent the color of what the LED light will be. All have the same hex value for turning of the LED, 020101.

Red LED:

Value	Time	Activity
0b84	5.023	Initialization
11090600060000005802	5.151	Initialization
010103	5.482	Read Request
010101	5.490	Read Response
020201	8.092	Sandwiched between Empty PDU
0203011021f0000f4010000e8030000ff	8.499	RED On

020101	8.506	RED Off
020201	9.058	Sandwiched between Empty PDU
0203011021f0000f4010000e8030000ff	10.121	RED On
11090600640000005802	11.840	Before Disconnect

Blue LED:

Value	Time	Activity
0b84	3.848	Initialization
11090600060000005802	4.024	Initialization
020302021f1f0000f4010000e8030000ff	7.045	BLUE On
020201	7.549	Sandwiched between Empty PDU
020201	8.140	Sandwiched between Empty PDU
020302021f1f0000f4010000e8030000ff	8.808	BLUE On
020101	8.813	BLUE Off
020201	9.344	Sandwiched between Empty PDU

Green LED:

Value	Time	Activity
0b84	3.558	Initialization
11090600060000005802	3.735	Initialization
010101	4.127	Read Response
020300021f1f0000f4010000e8030000ff	6.780	GREEN On
020101	6.784	GREEN Off
020300021f1f0000f4010000e8030000ff	7.406	GREEN On
020101	7.420	GREEN Off
020201	7.987	Sandwiched between Empty PDU

In summary, it is undesirable to utilize Bluetooth to collect data from IoT devices. Many IoT devices communicate over Bluetooth connections, but interpreting the information in captured packets is very difficult and time consuming, presuming the packets are not encrypted. Crackle could be used to decrypt/crack packets, but this only works if the LTK is already known, or if the packet capture observes the initial Bluetooth connection. Even when first establishing a connection, using the Ubertooth to sniff out traffic to later use with Crackle can be a daunting task. Chances are minimal that the Ubertooth would not only be on the right channel the first time, but also be able to capture all the key exchange packets.

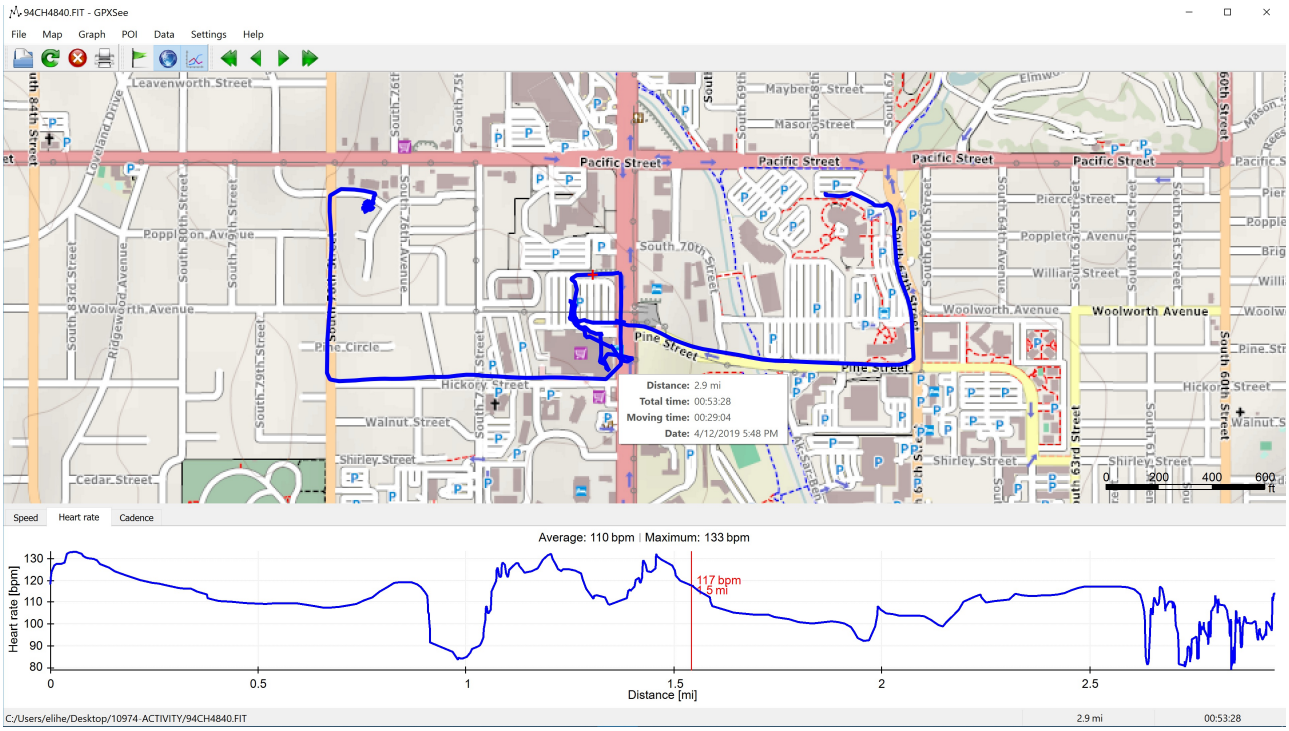
Hands-on Research with FTK Imager, Autopsy, and XRY

To do hands-on research with more traditional forensic tools such as FTK Imager, Autopsy, and XRY, a test bed using a Garmin HR+, Google Home Mini, Metawear CPRO, and a mobile device was set up. The Garmin is a wearable that records the user's heart rate, steps taken, stairs taken, and when prompted GPS location as the user exercises outdoors. The device also has a mobile app that was downloaded on the phone. An account was set up with Garmin and the device was connected to the phone. A weekend was spent getting data on the device and phone. One teammate wore the device and periodically went out for "runs" to collect GPS data. The Google Home Mini and Metawear were also connected to this mobile phone and were periodically used over the weekend. The Google Home Mini was asked questions and instructed to create events. We were unable to upload the data collected by the Metawear to Google Drive due to an application error.

Initially FTK Imager was used to get an image of the devices, similar to a traditional investigation involving a hard drive. The Google Home Mini was not recognized by FTK Imager and the Metawear does not have USB connection capabilities. Surprisingly the Garmin device was recognized by FTK Imager and we were able to collect a bit-for-bit image of the device. The image was then analyzed using Autopsy, a free forensic tool. Most of the files stored on this device are a Garmin proprietary ".FIT" file extension. These files contain information about the device, data that is passively collected, and data that is actively collected when a user starts a workout. We were able to find an application called GPXSee that can read and display this file format, but only if there is GPS data in the file. With this application, we were able to see the exact routes taken by the teammate when they turned on the GPS function. In addition, we could see their speed and heart rate at all points along the route. The short video below shows this in action.

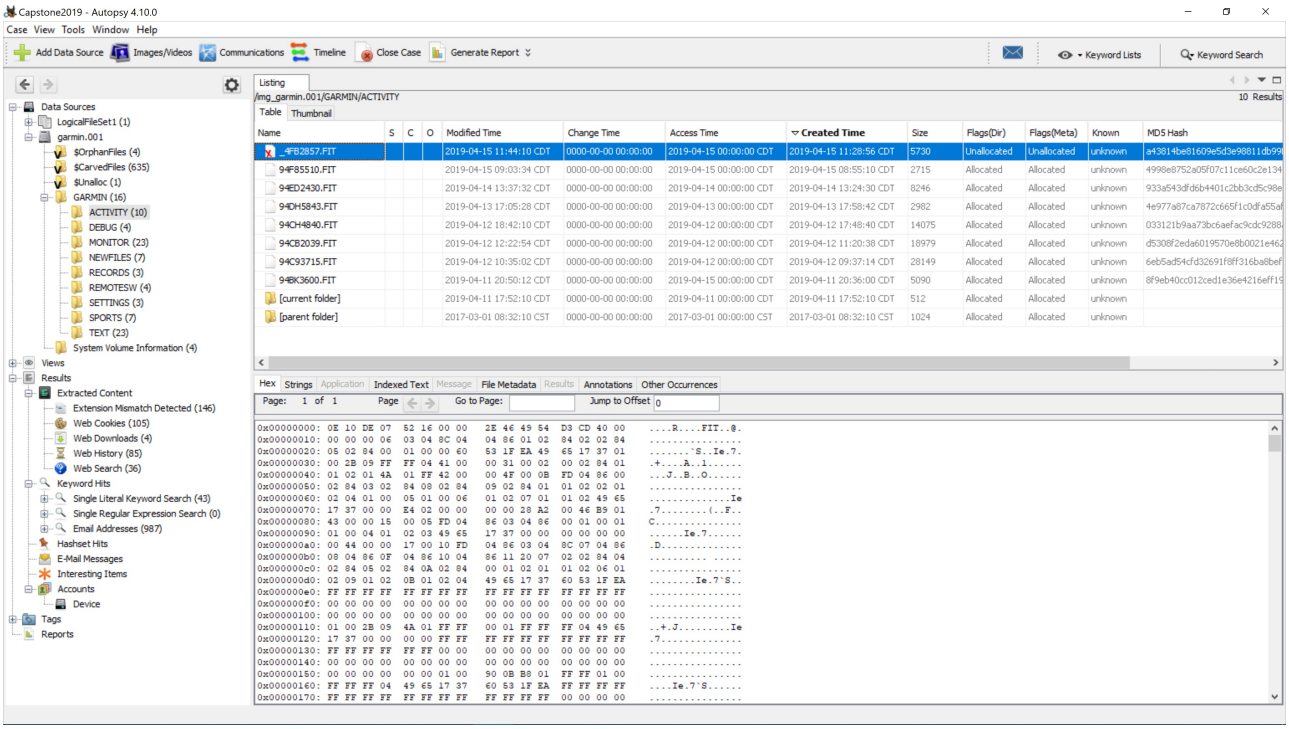
Video Capture:

(click to play)

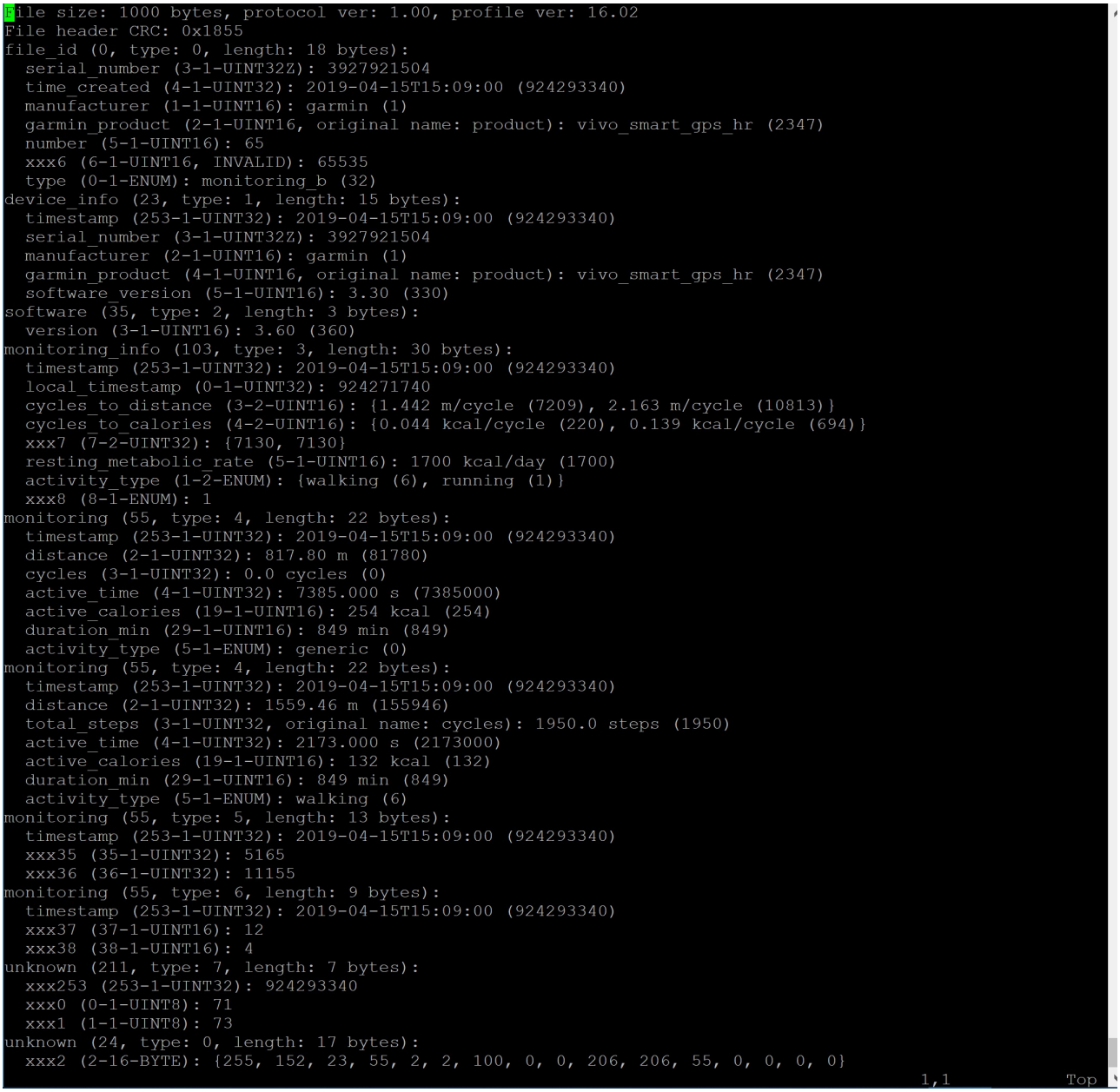


(https://use.vg/ql7cdf)

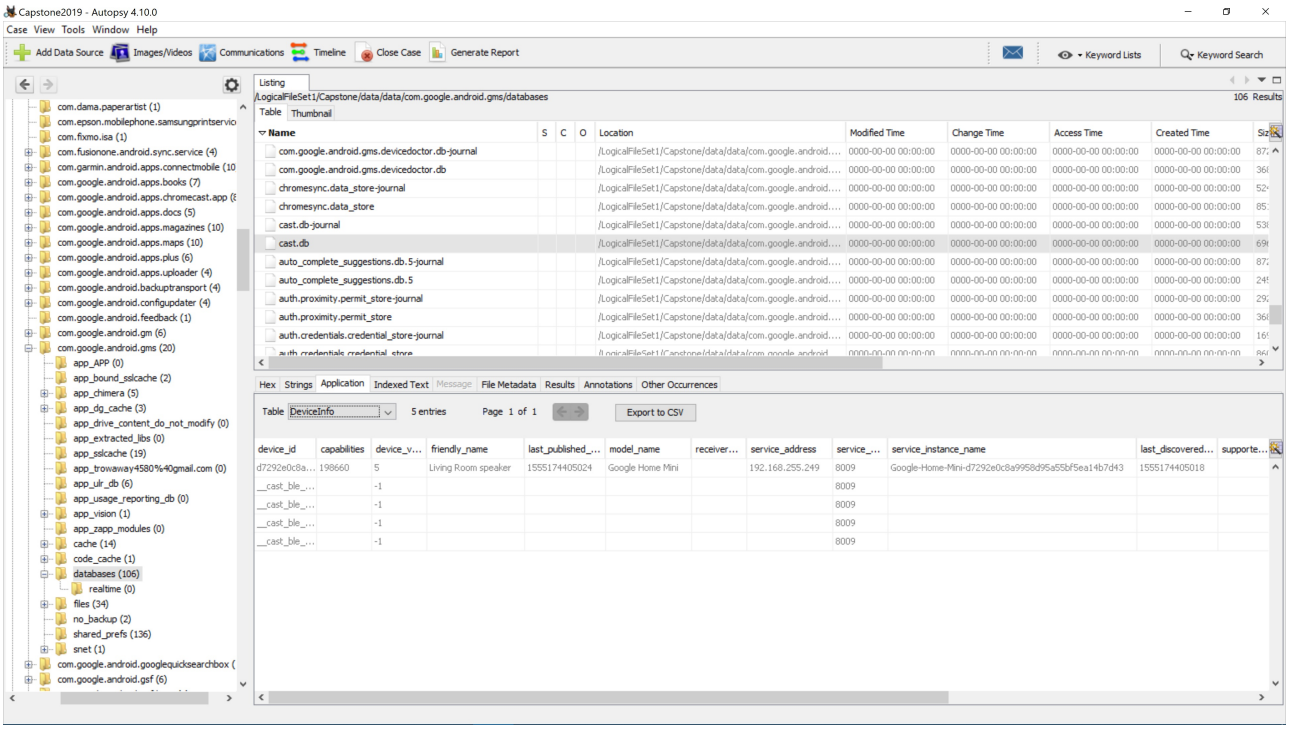
Most interesting was the one deleted file found. At the end of a workout, the user has the option to save or discard the workout. Two of the workouts done were discarded. One was presumably overwritten by subsequent workouts (of which there were 2) the other, which was the last entry before data acquisition, was recovered and viewable using GPXSee. These files are shown in Autopsy below.



As mentioned previously, GPXSee can only parse data files that include GPS data. The Garmin only captures GPS data when the user specifies and starts an outdoor workout. This means that a majority of the data collected by the Garmin may not be readable by GPXSee, as the Garmin will passively collect heart rate, steps, and other activity data. The .FIT files can be converted into .CSVs but the converted files are difficult to read and gather data from. We were able to find a command line utility called [fitdump (https://github.com/mrihtar/Garmin-FIT)](https://github.com/mrihtar/Garmin-FIT) that will parse out the data in these files and print them out. As can be seen in the screenshot below, the output is not perfect, but useful information about the wearer's activity can be gathered. While we were not expecting it, we were able to gather a lot of data from this device.

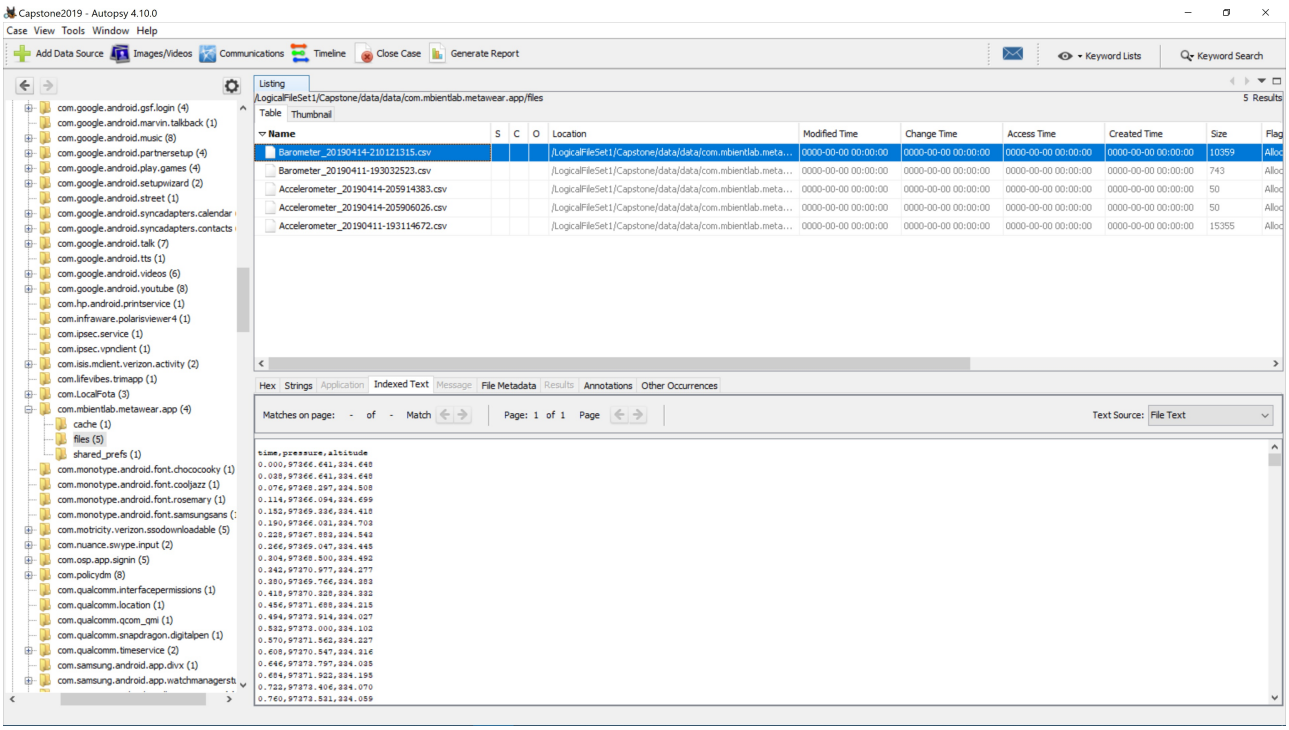


The second test done with these devices was pulling the data from the phone using XRY and using that as a proxy to get the application data from the devices. For this test, XRY Version 7.11 was used to create a full logical image of the mobile device. The image was then exported, and Autopsy was used to examine the files. The phone had been factory reset before the data propagation portion of this test, so there was not a lot of data on the phone itself. However, data was found from the Garmin device, Metawear and Google Home Mini. The only data found from the Google Home Mini was that there was a Google Home Mini connected to the mobile device and the events created using voice commands. The evidence of connection between the phone and Google Home Mini is shown in the screenshot below.

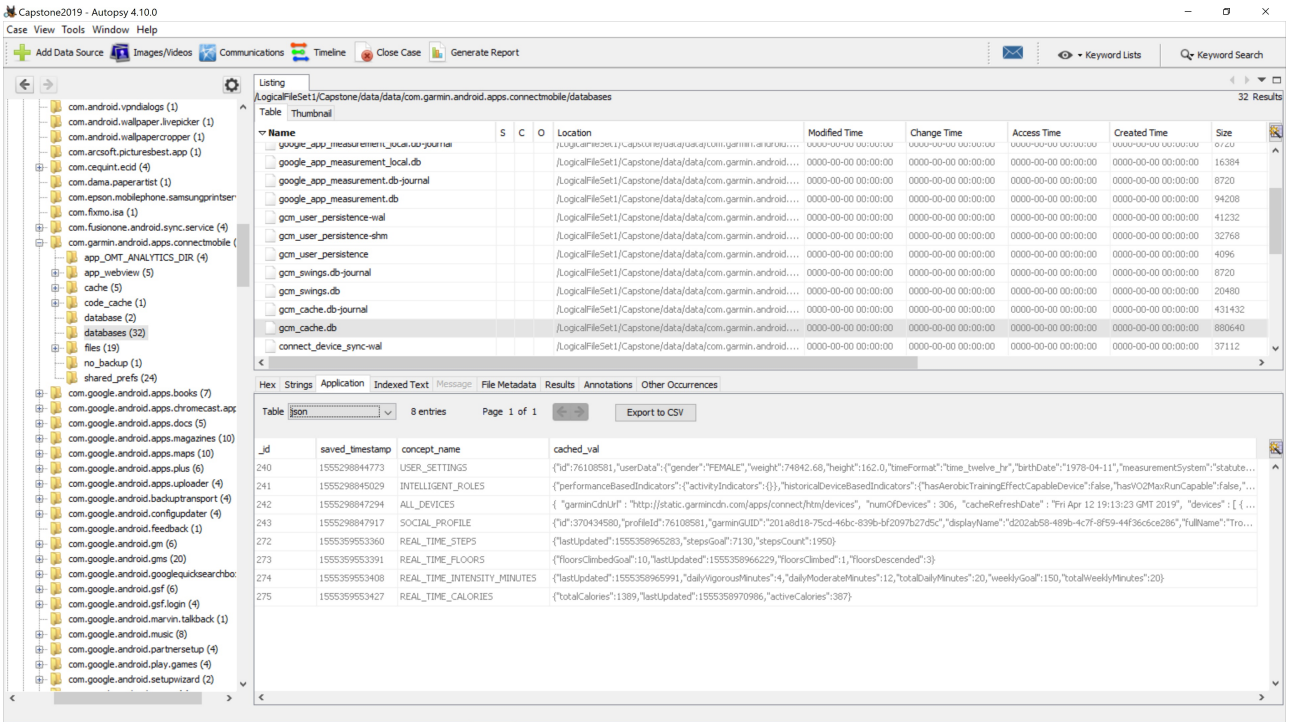


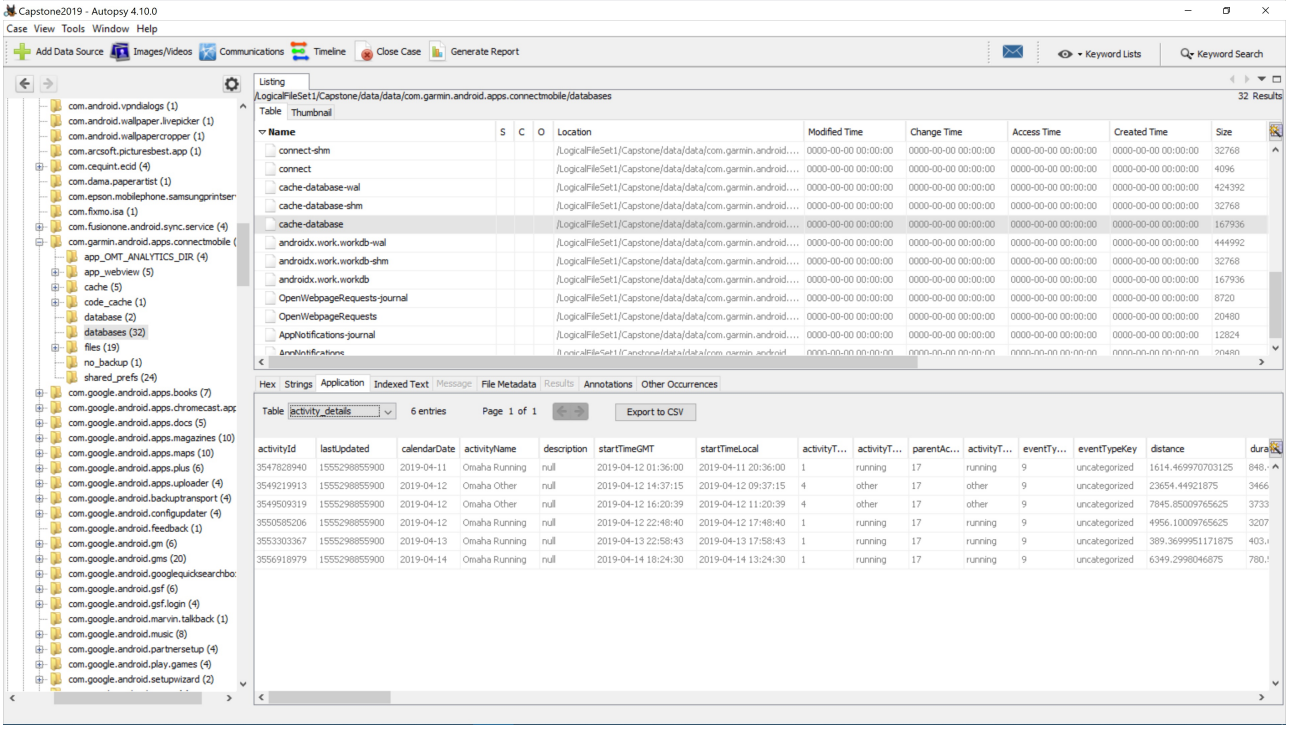
No logs of the conversations or searches requested were found. According to [Google (https://support.google.com/googlehome/answer/7072285?hl=en)](https://support.google.com/googlehome/answer/7072285?hl=en), these logs would be found on their servers through the user's Google account.

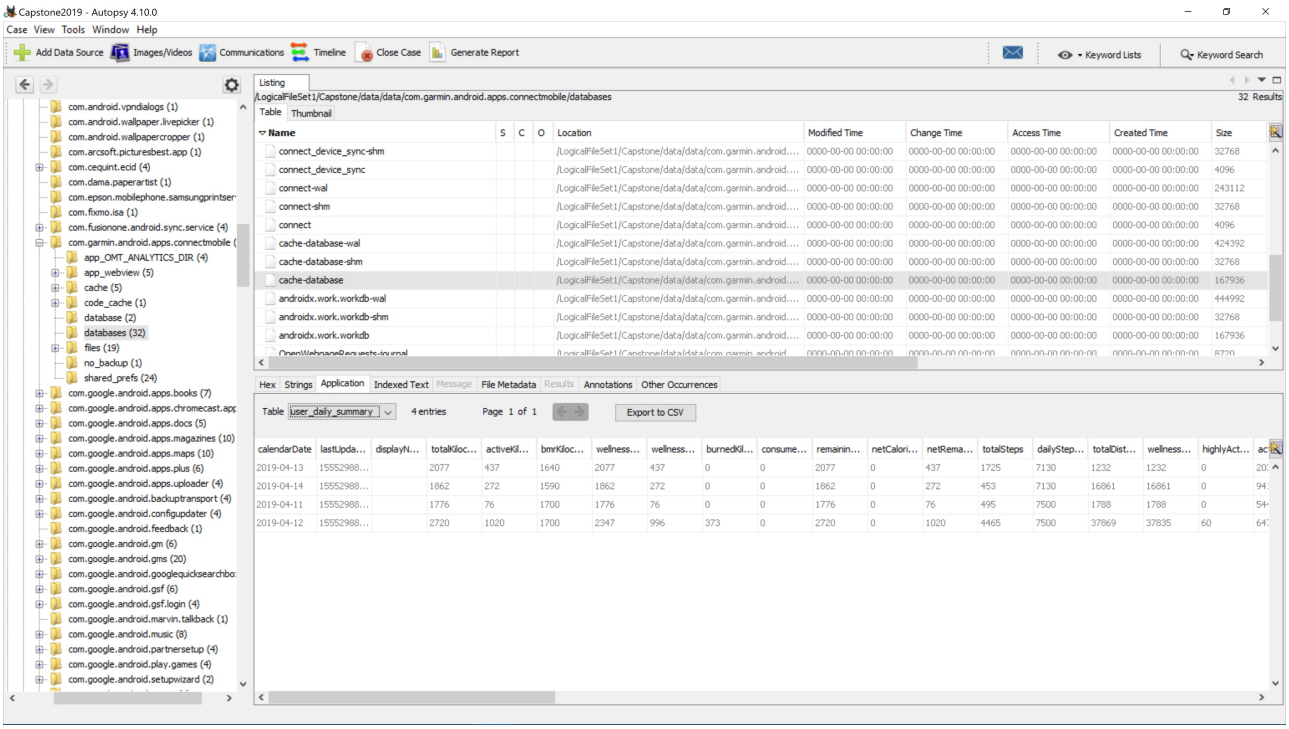
The records created using the Metawear that we were unable to export were recovered from the phone, shown below.



The records for the Garmin were the most informative. These included the user's profile information (gender, weight, height, etc.), daily summary information (total steps, max, min, and average heartrate, etc.), and activity information (activity type, start date, start location, duration, etc.). These are shown in the screenshots below. Each of these databases have many tables and columns so not all information is displayed.


Capstone2019 - Autopsy 4.10.0
Case View Tools Window Help
Add Data Source
Images/Videos
Communications
Timeline
Close Case
Generate Report
Keyword Lists
Keyword Search
Listing
/LogicalFileSet1/Capstone/data/data/com.garmin.android.apps.connectmobile/databases
32 Results
Table
Thumbnail
Name
S
C
O
Location
Modified Time
Change Time
Access Time
Created Time
Size
google_app_measurement_local.db
google_app_measurement.db-journal
google_app_measurement.db
gcm_user_persistence-wal
gcm_user_persistence-shm
gcm_user_persistence
gcm_swings.db-journal
gcm_swings.db
gcm_cache.db-journal
gcm_cache.db
connect_device_sync-wal
Hex
Strings
Application
Indexed Text
Message
File Metadata
Results
Annotations
Other Occurrences
Table
json
8 entries
Page 1 of 1
Export to CSV
_id
saved_timestamp
concept_name
cached_val
240 1555298844773 USER_SETTINGS
241 1555298845029 INTELLIGENT_ROLES
242 1555298847294 ALL_DEVICES
243 1555298847917 SOCIAL_PROFILE
272 1555359553360 REAL_TIME_STEPS
273 1555359553391 REAL_TIME_FLOORS
274 1555359553408 REAL_TIME_INTENSITY_MINUTES
275 1555359553427 REAL_TIME_CALORIES


Capstone2019 - Autopsy 4.10.0
Case View Tools Window Help
Add Data Source
Images/Videos
Communications
Timeline
Close Case
Generate Report
Keyword Lists
Keyword Search
Listing
/LogicalFileSet1/Capstone/data/data/com.garmin.android.apps.connectmobile/databases
32 Results
Table
Thumbnail
Name
S
C
O
Location
Modified Time
Change Time
Access Time
Created Time
Size
connect-shm
connect
cache-database-wal
cache-database-shm
cache-database
androidx.work.workdb-wal
androidx.work.workdb-shm
androidx.work.workdb
OpenWebpageRequests-journal
OpenWebpageRequests
AppNotifications-journal
Hex
Strings
Application
Indexed Text
Message
File Metadata
Results
Annotations
Other Occurrences
Table
activity_details
6 entries
Page 1 of 1
Export to CSV
activityId
lastUpdated
calendarDate
activityName
description
startTimeGMT
startTimeLocal
3547828940 1555298855900 2019-04-11 Omaha Running null 2019-04-12 01:36:00 2019-04-11 20:36:00
3549219913 1555298855900 2019-04-12 Omaha Other null 2019-04-12 14:37:15 2019-04-12 09:37:15
3549509319 1555298855900 2019-04-12 Omaha Other null 2019-04-12 16:20:39 2019-04-12 11:20:39
3550585206 1555298855900 2019-04-12 Omaha Running null 2019-04-12 22:48:40 2019-04-12 17:48:40
3553303367 1555298855900 2019-04-13 Omaha Running null 2019-04-13 22:58:43 2019-04-13 17:58:43
3556918979 1555298855900 2019-04-14 Omaha Running null 2019-04-14 18:24:30 2019-04-14 13:24:30



Unfortunately, this data did not include the exact route taken by the user. While that data is shown on the app, it seems it is uploaded to the user's account and only available online. This, and with the Google Home Mini, is where investigators might have to use cloud forensics to get all the data they might need for a case.

Garmin (https://www.garmin.com/en-US/forms/lawenforcement/) will also work with law enforcement if any of their units are stolen, lost, or recovered. Their website provides a web form for any officiers or legal personnel to fill out to contact them. The company does note that stolen devices cannot be tracked, and a copy of police reports should be attached if available. This is not necessarily a guarantee that they will cooperate and/or provide all requested information, and in some circumstances, a subpoena may be required.

Hinderances

- Google Home Mini network traffic was encrypted using the http-over-tls protocol. This indicates little to no forensic data is available from the device itself nor the network it resides on.
- Bluetooth works by working on three different channels between 2402-2480MHz, usually referred to as channel 37, 38, and 39. When capturing Bluetooth packets with the Ubertooth One between the Android phone and IoT devices, it had to be on the correct channel to capture. At times it would not be on the right channel to capture packets and the device would need to be reseated.
- The Ubertooth tool used, when under the correct channel and after capturing a "CONNECT_REQ" packet does not 100% guarantee that it will be able to capture all Bluetooth data that occur between to chosen devices.
- When using the Crackle tool to decrypt any captured packets from .pcap files, it was found that the tool only works when sniffing a Bluetooth connection for the first time between two devices.

- We had to try many phones before we found one that could interface with XRY. This tool has a specific list of phone models and operating systems it can work with. With the limited number of phones we had at our disposal, we were not able to find one that we could do a full bit-for-bit extraction on, but we did have one that could be used for a full logical extraction.
- The data stored on the Garmin is in a proprietary file format. The files can be converted to CSVs, but they are not easily read in that format.
- Many of these devices interact with the cloud so some of the data cannot be acquired using just traditional digital forensics or mobile forensics. Unfortunately, cloud forensics is outside the scope of this project so that data was not collected.