Wyze Camera Final Report

Alma Nkemla, Katherine Paton-Smith
Embedded Systems Cyber Security VIP
Georgia Institute of Technology
Atlanta, GA
ankemla3@gatech.edu, kpatonsmith@gatech.edu

Houlton McGuinn
Georgia Tech Research Institute
Atlanta, GA
houlton.mcguinn@gtri.gatech.edu

Jake Ashmore
Advisor
Georgia Tech Research Institute
Atlanta, GA
jacob.ashmore@gtri.gatech.edu

***Abstract*—This paper is the final report for the Wyze Camera Team in the VIP: Embedded Systems Cyber Security. The document starts by outlining characteristics of the Wyze camera in addition to previous works regarding vulnerabilities of the camera. Then current research into determining the RF protocol is described. The goal of the research is to ultimately create a RF testing system to reveal more vulnerabilities of the Wyze camera.**

I. Introduction

The Wyze Camera V2 is an affordable Internet of Things (IoT) Device that can serve as a security camera. The camera is able to connect through the Wyze smartphone application, available on both Android and IOS mobiles and can be used synchronously with virtual assistant technology such as Google Assistant.

The additional Wyze Sense allows for multiple wireless contact sensors and motion sensors to be connected with the camera and application. The Wyze security camera is sold by Wyze Labs, a company based in Seattle, Washington that specializes in smart home products and wireless cameras.

Manufacturers of IoT devices have been trying to increase the number of connected devices, cloud access capabilities, and mobile applications to further benefit consumers. An increase in IoT devices, such as the Wyze camera also enable malicious attacks as IoT devices are connected to the internet and lack security capabilities and encryption protocols [1]. In addition, small companies choose not to upgrade their device security in the development stage as the cost of security may negate its financial value [2].

II. Functional Descriptions

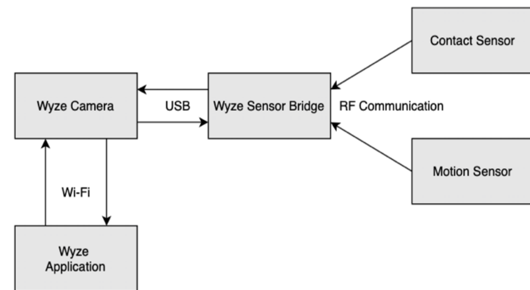

Figure 1. Functional diagram of Wyze Sensor Bridge

The ecosystem of the Wyze IP Camera V2 as shown in Figure 1 consists of the Wyze camera, the Wyze App, a sensor bridge, and additional contact and motion sensors. The camera communicates through Wifi with the Wyze application. The sensor bridge is connected by USB to the back of the camera, and can then communicate via RF with up to 100 additional contact and motion sensors [3].

A. Camera

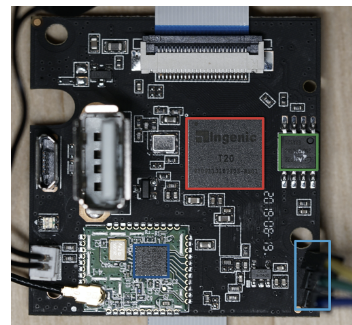

Figure 2. Main Board

The camera has a sandwich structure of 3 printed circuit boards (PCB): the main board shown in Figure 2, the microSD card board, and the camera sensor board. The main board contains the Ingenic T20 System on a chip, outlined in red, a high-performance video processor based on the MIPS

instruction set architecture. The board has a microUSB port for power supply, a USB-A port to connect to the sensor bridge, and a wireless daughter board for 802.1 1n Wi-Fi connection.

B. Sensor Bridge

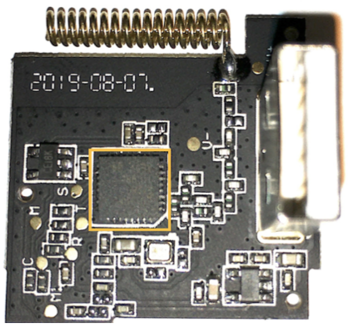

Figure 3. Functional Diagram of Wyze Sensor Bridge

The sensor bridge shown in Figure 3 connects to the back of the Wyze Camera in the USB-A port. It has an antenna for communication with the Contact and Motion Sensors. The CC1310 microcontroller (MCU), outlined in orange, is responsible for processing the RF communication.

C. Motion Sensor and Contact Sensor

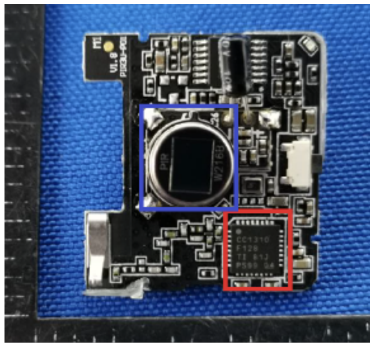

Figure 4. Motion Sensor

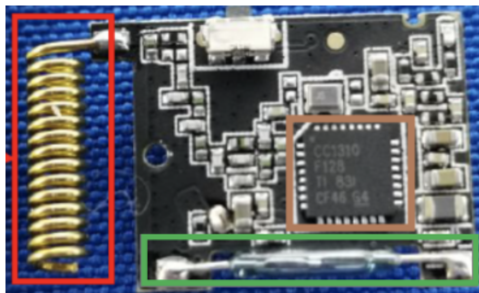

Figure 5. Contact Sensor

The motion and contact sensors each have antennas and the CC1310 MCU for RF communication, outlined in red on the motion sensor Figure 4 and in brown on the contact sensor Figure 5.

D. CC1310 Microcontroller

The CC1310 MCU is responsible for processing the sub 1GHz RF communication between the sensor bridge and the contact and motion sensors. The chip consists of two CPUs, the ARM Cortex-M3 responsible for the application layer and the ARM Cortex -M0 which processes the RF signals [4].

The CC1310 MCU can use many different protocols such as IEEE 802.15.4, Wireless M-Bus (T, S, C Mode), 6LoWPAN, or MIOTY [5]. However Wyze uses their own proprietary protocol for the Wyze Sense.

III. Existing Vulnerabilities

Up to now, known vulnerabilities in the Wyze camera are mainly the result of weak encryption which can result in leakage of sensitive data such as users' passwords or emails. In addition, the Wyze camera uses a variety of open source libraries and frameworks that can be found and used by anyone on the internet.

In 2019, Wyze Labs, the company that owns the Wyze Camera, was sued following a data breach that exposed the information of 2.4 million users, who had their cameras connected to their phones or WIFI. This attack targeted a database which Wyze Labs had copied from their main production servers and left open on the internet with its previous security protocols removed [6]. Although this security breach was the result of human error, Wyze camera vulnerabilities have historically been caused by weak encryption and leakage of data [7].

A. Denial Of Service Attacks

A Denial-of-Service (DoS) attack intentionally makes a service inaccessible to its users by disrupting the functions of the device or network. The Wyze camera firmware Linux Kernel version (3.10) is vulnerable to a DOS attack through an auth_reply message that triggers an attempted build_request operation [8]. In addition, the TCP Stack in the kernel version has improper implementation of a SYN cookie protection mechanism in the case of a fast network connection. This vulnerability can allow attackers to send TCP SYN packets which can lead to total shutdown of the affected resource and a denial of service for the user [9].

B. Man-In-The-Middle Attacks

Because of lack of authentication, the camera is vulnerable to man-in-the-middle attacks. A man-in-the-middle attack is when a third party intercepts the communications between two parties and listens in to or modifies the traffic between the two.

A previous semester launched a successful MITM attack on the Wyze Camera to capture firmware for the Wyze Sense bridge, discussed in more detail in Section VI.A.

C. Weak Root Password

The Wyze camera can also be accessed directly through a serial connection on its main board. Following this connection, the camera has a root password set by the manufacturers on every device produced to protect its firmware from tampering. However, this password only uses an encryption algorithm that uses 8 characters of entropy, meaning that the password is very weak. The password has previously been deciphered and was known to the public and available online as "iSmart12"[10].

The serial port connection on the main board can be seen in Figure 2 (Section II.A).

D. Semester's Focus

The focus of this semester is determine the RF protocol to then build SDR spoofing software using Scapy for the contact and motion sensors. There will also be an attempt at reverse engineering the main program binary to find how wireless data is used by the program. The goal is to ultimately combine these tools to build a RF testing system to discover more vulnerabilities with the camera.

IV. ICAMERA BINARY

Communication between the host (the camera) and the dongle is done through packets. By going through various functions related to packet read and editing names and data types information can be discerned about the packet structure of incoming RF data [11].

A. Understanding Packets

Communication between the camera and the dongle is done through packets. The general communication between the dongle and the camera is done in on packets of variable lengths:

- The two bytes magic field can be:
  - [aa][55] indicates sending information from the host to the dongle
  - [55][aa] indicates sending information from the dongle to the host.
- The single byte type field indicates how communication between the host and the dongle behaves. The type can be either:
  - 0x43: This means that the receiver will respond with a packet of type 0x43 and "cmd" field set to "X+1" with any additional data put in the payload.
  - 0x53: This means that the receiver will first respond with a simple ACK packet of type 0x53, "cmd" field set to 0xFF and the length set to X. There is no pay load in this case. Following the ACK packet, depending on the command, the receiver can respond with zero or more packets of type 0x53 and "cmd" field set to "X+1". The response may or may not include a payload and its length field will be se to reflect the size of the data.
- "Payload" field describes the parameters for a given command.
- There exists a length field tallying the total length of the cmd field, payload and checksum.
- The checksum which is big-endian uint16_t sum of the data in the packets [11].

The description of the protocol between the dongle and the camera can provide a little bit of insight regarding to the packet structure of communication between the camera and sensors.

V. SIGNAL ANALYSIS OF THE RF PROTOCOL

This semester the focus is on reverse engineering Wyze's proprietary RF protocol between the contact to determine the encoding mechanism which can then lead to finding the sync word and preamble. Determining the RF protocol can be done by analysing Over-The-Air (OTA) packets of recordings between the camera and the sensor.

A. Packet Captures

A packet capture intercepts a data packet crossing a point in a data network, and once captured, the packet can then be stored and later analyzed. To capture packets going between the dongle and sensors, an Ettus N210 SDR with a standard vertical antenna was used. GNU Radio was used as the software to record the packets. The GNU Radio flowgraph used to record and the block diagram for data flow can be seen below:

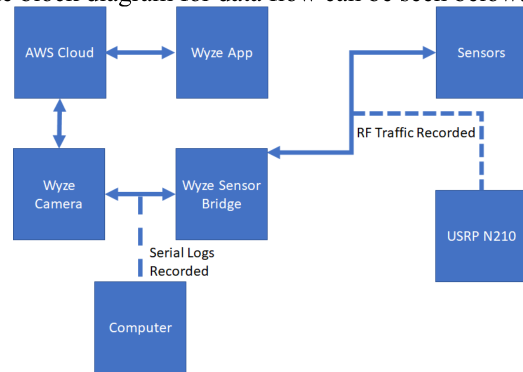

Figure 6: Block diagram showing where packets and logs were captured.

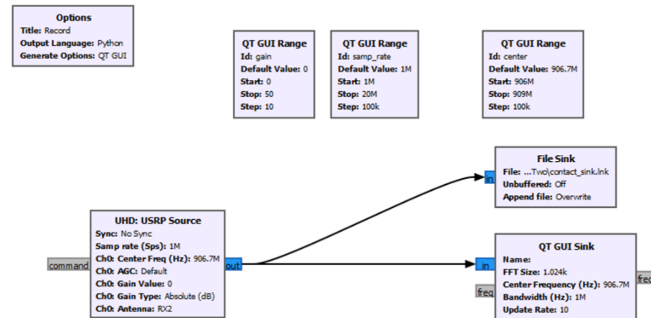

Figure 7: GNU Radio flowgraph used to record packets for playback.

Serial logs were captured using a direct connection to the Wyze Camera, which outputs received packets as part of its debug information. Alternatively, serial logs can be directly captured using WyzeSensePy [12], and a USB connection to the dongle, which implements the communication protocol between the dongle and camera.

B. RF Overview

Information from the contact and motion sensor is generated by their microcontrollers and modulated through Frequency Shift Keying (FSK) as seen by last semester's students. Universal Radio Hacker (URH) can aid in analyzing air

captures of packets to analyze the protocol between the camera and the dongle.

C. Sensor Over-The-Air Packet

By looking at Over-The-Air (OTA) packets of recordings, using URH, between the motion and contact sensors with the camera, we can try to gain a better understanding of the communication protocol.

1) Packet Contents

Not all data transmitted in over-the-air (OTA) packets is currently known. We suspect OTA packets from sensor-to-dongle contain the MAC of the sensor which is used for identification. Additionally, there is a 16-bit sequence counter that increments each time there is an event (open/close, motion/no motion). This sequence counter resets to 0 every time the sensor is powered down. Finally, the type of event is transmitted (open/close, motion/no motion).

2) Communication Protocol

When an event is detected, the sensor transmits a packet to the dongle. After the dongle has received the packet, the dongle replies with an "ACK", communication that it received the event alert from the sensor. The dongle keeps state of sensors in memory. For a contact sensor, sending two "open" event alerts consecutively, will result in the second message being considered an error, and not another event. This state keeping seems only to be related to the state the sensor is in (open/close, motion/no/motion) and not related to the sequence counter embedded in the message.

3) Recordings

The packets were transmitted through a frequency of 906.7MHz with a modulation type of FSK. Using URH the packets could be sent to the dongle and from the dongle to the host. These recordings were taken from three states of the motion sensor: motion, connect, delete. Further information of the radio parameters of captures can be seen in figure 11.

4) Motion Sensor

An analysis of the motion sensor captures was done through the tools of URH along with an analysis of the log files using a parser in order to gain more insight on the data about Over.

D. Using Universal Radio Hacker

URH serves as a tool to analyze the OTA packets from communication between the camera and the motion sensor.

There are two discernable packets: Ones in which the first 16 bytes are the repeating hexadecimals, 0xd534acca as seen in Figure 8 (yellow) and the other in which the first 16 bits are 0xaa695995 as seen in Figure 9 (yellow).

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
1	d	5	3	4	a	c	c	a	d	5	3	4	a	c	c	a	a	a	c	b	5	3	3	5	0	0	6	9	f
2	d	5	3	4	a	c	c	a	d	5	3	4	a	c	c	a	a	a	c	b	5	3	3	5	0	0	6	9	f
3	d	5	3	4	a	c	c	a	d	5	3	4	a	c	c	a	a	a	c	b	5	3	3	5	0	0	6	9	f
4	d	5	3	4	a	c	c	a	d	5	3	4	a	c	c	a	a	a	c	b	5	3	3	5	0	0	6	9	f
5	d	5	3	4	a	c	c	a	d	5	3	4	a	c	c	a	a	a	c	b	5	3	3	5	0	0	6	9	f
6	d	5	3	4	a	c	c	a	d	5	3	4	a	c	c	a	a	a	c	b	5	3	3	5	0	0	6	9	f
7	d	5	3	4	a	c	c	a	d	5	3	4	a	c	c	a	a	a	c	b	5	3	3	5	0	0	6	9	f
8	d	5	3	4	a	c	c	a	d	5	3	4	a	c	c	a	a	a	c	b	5	3	3	5	0	0	6	9	f
9	d	5	3	4	a	c	c	a	d	5	3	4	a	c	c	a	a	a	c	b	5	3	3	5	0	0	6	9	f
10	d	5	3	4	a	c	c	a	d	5	3	4	a	c	c	a	a	a	c	b	5	3	3	5	0	0	6	9	f
11	d	5	3	4	a	c	c	a	d	5	3	4	a	c	c	a	a	a	c	b	5	3	3	5	0	0	6	9	f

Figure 8. First type of packet capture of communication between the motion sensor and the camera

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
1	a	a	6	9	5	9	9	5	a	a	6	9	5	9	9	5	5	5	9	6	a	6	6	a	0	0	d	3	e
2	a	a	6	9	5	9	9	5	a	a	6	9	5	9	9	5	5	5	9	6	a	6	6	a	0	0	d	3	e
3	a	a	6	9	5	9	9	5	a	a	6	9	5	9	9	5	5	5	9	6	a	6	6	a	0	0	d	3	e

Figure 9. Second type of packet capture of communication between the motion sensor and the camera

From the documentation in the TI SDK of the CC1310 MCU [12], the structure of packets will include the preamble, sync word, a header of variable length containing length and address information for the receiver such as the length field, magic field and type field, payload and a crc to verify the message's integrity. There are different packets structures in which the order of these fields vary with a format in which the length field doesn't have to be located at the beginning of the header and a format in which the header includes the crc.

Discarding the first 16 bytes as the information before the header as the preamble and the sync word, we can try get information about the header. All packets with a repeating hex of 0xd534acca have columns 17 and 18 with hex of 0xaa indicating the magic field or the communication direction between from the camera to the sensors. In addition the packets next couple of bytes were 0xcb53350069f ending at column 34 as seen in Figure 6 (green). From columns 41 and onwards we can see that were the payload begins as information being transmitted by each packet differs. Packets with starting bits of 0xaa695995 instead have a magic field of 0x55 at columns 17 and 18 as seen in Figure 9 (green). The remaining columns from column 34 to column 41 (payload) serves as byte that may contain the type or the length field, however more work is needed to pinpoint these two fields.

E. Log Analysis

In addition to analyzing the OTA packets between the motion sensor and the camera, comparing the logs from communication between the camera and the motion sensor, and the packets can provide insight into the RF protocol.

The WyzeSensePy parser is a python package attempting to implement the communication protocol between the camera and the USB dongle [12]. However many of the commands found in the parser can be viewed in the logs for the communication between the camera and the sensors. The parser sorts through the various commands in the log in terms of hexadecimals and type as seen in Table 1. In Table 1, HD refers to commands initiated from the host (camera) and DH refers to

commands initiated by the dongle. The type of command can either be asynchronous with hexadecimal of 0x53 or synchronous with hex of 0x43. The fields in the logs are not always commands as indicated with CMD but could also be notifications, indicated with NOTIFY.

Table 1. WyzeSensePy Parser Commands

Name	Type	CMD
HD_CMD_GET_ENR	0x43	0x02
HD_CMD_GET_MAC	0x43	0x04
HD_CMD_GET_KEY	0x43	0x06
HD_CMD_INQUIRY	0x43	0x27
HD_CMD_UPDATE_CC1310	0x43	0x12
HD_CMD_SET_CH554_UPGRADE	0x43	0x0E
ASYNC_ACK	0x53	0xFF
DH_CMD_FINISH_AUTH	0x53	0x14
DH_CMD_GET_DONGLE_VERSION	0x53	0x16
DH_CMD_START_STOP_SCAN	0x53	0x1C
DH_CMD_GET_SENSOR_R1	0x53	0x21
DH_CMD_VERIFY_SENSOR	0x53	0x23
DH_CMD_DEL_SENSOR	0x53	0x25
DH_CMD_GET_SENSOR_COUNT	0x53	0x2E
DH_CMD_GET_SENSOR_LIST	0x53	0x30
DH_NOTIFY_SENSOR_ALARM	0x53	0x19
DH_NOTIFY_SENSOR_SCAN	0x53	0x20
DH_NOITFY_SYNC_TIME	0x53	0x32
DH_NOTIFY_EVENT_LOG	0x53	0x35

From the log files, communication between dongle to camera always begins with the first two fields as either [55][aa] indicating a message received from dongle or [aa][55] indicating a message written to camera. This is shown in both Figure 10 and Figure 11 in blue (The fields are bolded for better clarity). Following the communication direction the type field of the packet either 0x43 or 0x53 determines the response of receiver. In the case of the packets from the motion sensor, most of the log files with type field of 0x43 where discarded due to errors in recording the packets so no information could be discerned from those packets. However packets with type field 0x53 (green in both Figure 10 and Figure 11 results in the receiver sending an ack packet using the command, ASYNC_ACK, 0xff (red in both Figure 10 and Figure 11) with no payload.

Figure 10 differs from Figure 11 due to the field following the type field 0x53. From the logs, the 4th field could either be the hex 0x19 (pink), which refers to the DH_NOTIFY_SENSOR_ALARM notification from Table 1, followed by the hex 0x35 (orange), which refers to the DH_NOTIFY_EVENT_LOG notification shown in Figure 10 or a command (orange) followed by the hex 0x19 (pink) shown in Figure 11.

```
[DONGLE_RECORD->dongle.c,2354]:====================received from
dongle==================
[DONGLE_RECORD-
>dongle.c,2355]:[55][aa][53][19][35][00][00][01][78][27][19][db][3b][0e][a2][37][37][
39][39][30][37][36][45][02][01][00][03][05][e7]

[DONGLE_RECORD->dongle.c,3144]:====================write to dongle==================
[DONGLE_RECORD->dongle.c,3145]:[aa][55][53][35][ff][02][86]
[DONGLE_RECORD->dongle.c,148]:DONGLE SEND ACK
[DONGLE_RECORD->dongle.c,611]:log content len = 22
[DONGLE_RECORD->dongle.c,2354]:====================received from dongle=============
[DONGLE_RECORD-
```

Figure 10. Communication with no commands between [53] and [19]

This difference in the 4th and 5th field of the logs also affected the way in which the ACK packet sent by the dongle. The ACKS packets 4th field is always equivalent to the initials message's 5th field. The 5th field of every packet is referred to as the cmd field indicating a command (cmd) between sensor and the camera (orange in Figure 10 and pink in Figure 11). Following the cmd field and the ASYNC_ACK, 0xFF, is the hexadecimal 0x02 which refers to the command HD_CMD_GET_ENR, in the ACK packets and either the hex 0x6a or the command ,HD_CMD_GET_KEY (Figure 11) or 0x86 (Figure 10), values in which a command has yet to be determined.

```
[DONGLE_RECORD->dongle.c,2354]:====================received from dongle=============
[DONGLE_RECORD-
>dongle.c,2355]:[55][aa][53][1d][19][00][00][01][78][27][19][db][3e][a2][37][37][39][
39][30][37][36][45][02][1a][5f][00][01][01][00][03][30][06][6e]

[DONGLE_RECORD->dongle.c,3144]:====================write to dongle==================
[DONGLE_RECORD->dongle.c,3145]:[aa][55][53][19][ff][02][6a]
[DONGLE_RECORD->dongle.c,148]:DONGLE SEND ACK
```

Figure 11. Communication with command between [53] and [19]

A further analysis of the logs is required to find out more about commands following 0xff in the ACK packets and commands prior to the payloads in packets carrying information.

F. Replay Attack

Using captured packets from a contact and motion sensor, an USRP N210 was used to replay the recorded packets back to the dongle. The dongle successfully received the packets, as verified by the chosen input being displayed in the Wyze app. This was done using the GNU Radio Flowgraph seen in Figure 10. The first packet associated with an alert contains a 4-hex character value, that increments upwards each alert and resets when a sensor is powered off. When captured packets were replayed, this 4-character field returned to the value that was captured. It is unknown what these 4-characters represent, but given their incrementing when an event happens it is possible, they are some kind of sequence counter. Recorded packets are able to be sent in any order, as long as the event types alternate, and will be processed successfully by the dongle.

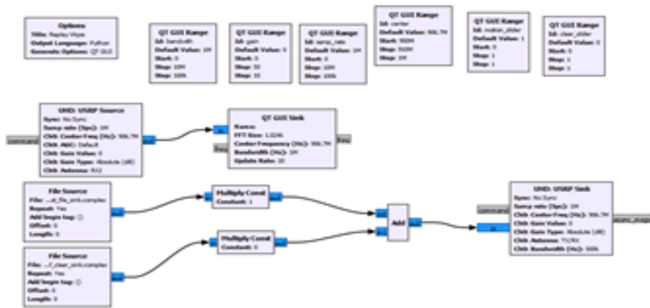

Figure 12. GNU Radio Flowgraph used for Successful Replay Attack

Expanding on the replay attack, URH was used to modulate packets so that arbitrary changes could be made. The settings used to successfully modulate packets can be seen in Figure 13.

Encoding:	-	Sample Rate:	1M
Carrier Frequency:	906.7MHz	Modulation Type:	FSK
Carrier Phase:	0°	Bits per Symbol:	1
Symbol Length:	100	Frequencies in Hz:	-19K/19K

Figure 13. Settings Used to Modulate Packets

Using URH, arbitrary packets were able to be sent to the dongle. Using a recorded contact sensor open alert, nibbles and then bytes were zero'd out sequentially and then transmitted with a contact sensor close alert in between. However, no change in the message's logs was observed. Given this, and the large variance that was observed for OTA packets payloads, it is very likely Wyze is performing packet whitening to aid in transmission. Packet whitening improves the transmission by exclusive-or'ing the data payload, with a stream of bits given by a pseudo-random sequence [14]. The whitening parameters used are currently unknown.

VI. Reverse Engineering RF Protocol

A. *Capturing the CC1310 Firmware*

Every time the camera is started, it calls out for the latest version of the CC1310 firmware. By setting up a Man in the Middle (MITM), a copy of the firmware can be obtained. A previous semester used mitmproxy to do this. The basic idea is that a host running mitmproxy sits in the middle of the traffic flow between a client and server, pretending to be the server to the client and pretending to be the client to the server [15] . In this case, mitmproxy acts as the server to Wye's client camera application while also acting as a client to the Wyze servers, as shown in Figure 14. When the camera calls for the CC1310 firmware update, mitmproxy is then able to relay that request from the camera to the server, and then capture and decode the firmware being sent back from the server.

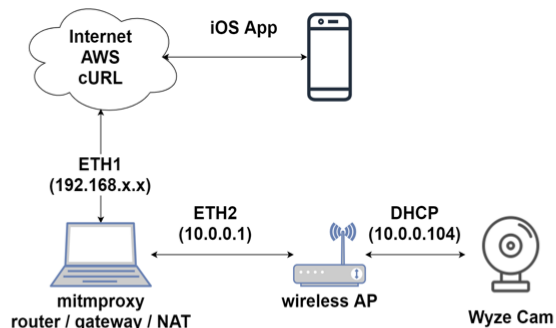

Figure 14. Mitmproxy Setup

This semester's focus is working on reverse engineering the captured CC1310 binary to find where the RF protocol is defined. The binary is run on the ARM Cortex -M3 processor, which is a 32-bit architecture.

B. *Ghidra*

Ghidra is an open-source software reverse engineering tool developed by the National Security Agency. Some of its capabilities include disassembly and decompilation of binary code . This makes Ghidra a great tool to use for reverse engineering the RF protocol, as it can disassemble the captured CC1310 binary into ARM Cortex -M3 assembly code, and also decompile that into easier to read C code.

C. *Setting up Memory Map in Ghidra*

A memory map details the structure of memory in a CPU, which often includes Flash memory, SRAM, and various peripherals. Part of the Cortex -M3 memory map can be seen in Figure 15 [4]. This is important as memory addresses are often used directly in assembly code. Without the memory map, it can be difficult to understand what each address is referencing, and therefore difficult to understand the functionality of the code. Ghidra is able to use a memory map to create clearer disassembled and decompiled code. However manually setting up the memory map can be tedious. A faster method to do so involves using a Ghidra script.

Base Address	Module	Module Name
0x0000 0000	FLASHMEM	On-Chip Flash
0x2000 0000	SRAM	Low-Leakage RAM
0x2100 0000	RFC_RAM	RF Core RAM
0x4000 0000	SSI0	Synchronous Serial Interface 0
0x4000 1000	UART0	Universal Asynchronous Receiver/Transmitter 0
0x4000 2000	I2C0	I2C Master/Slave Serial Controller 0
0x4000 8000	SSI1	Synchronous Serial Interface 1
0x4001 0000	GPT0	General Purpose Timer 0

Figure 15. Cortex-M3 Memory Map

A System View Description (SVD) file contains detailed functional descriptions of the system in ARM Cortex-M CPU's, including the memory map [18]. The YouTube video "Bare-metal ARM firmware reverse engineering with Ghidra and SVD-Loader" explains how an SVD-Loader can be used to make reverse engineering firmware easier [19]. The SVD-Loader is a Ghidra script which parses an SVD file to automatically set up the memory map. For use with the CC1310 firmware, the SVD-Loader is downloaded from GitHub [20] and added to the Ghidra Scripts path. It is then run with a

CC1310.svd file. The generated memory map in Ghidra for the CC1310 MCU can be seen in Figure 9, which matches that from the technical manual.

Memory Map [CodeBrowser(2): VIP-Wyze:/cc131

File Edit Tools Help

Memory Map – Image Base: 00000000

Memory Blocks

Name	Start	End	Length	R	W	X	Vola...	Type
ram	00000000	0001ffff	0x20000	✓	✓	✓		Default
I2C0_UART0...	40000000	40002fff	0x3000	✓	✓		✓	Default
SSI1	40008000	40008fff	0x1000	✓	✓		✓	Default
GPT0_GPT1_...	40010000	40013fff	0x4000	✓	✓		✓	Default
GPIO_I2S0_U...	40020000	400223ff	0x2400	✓	✓		✓	Default
CRYPTO	40024000	400247ff	0x800	✓	✓		✓	Default
TRNG	40028000	40029fff	0x2000	✓	✓		✓	Default
FLASH_VIMS	40030000	400343ff	0x4400	✓	✓		✓	Default

Figure 16. CC1310 Firmware Memory Map in Ghidra

D. Proprietary Radio

Wyze uses their own proprietary radio protocol. Texas Instruments (TI) details their recommended proprietary radio commands and packet structure in the Technical Manual [4].

The Radio commands are used to set up the radio mode, transmit packets, and receive packets. Figure 17 shows what appears to be potential radio setup function in the firmware. The function most likely passes in a pointer to a radio command and checks which radio setup command it is. The scalars it checks correspond to the command ID's in the technical manual [4].

Decompile: 0000abd8_radio_setup_maybe – (0.0.0.33.bin)

```
1
2   undefined4 0000abd8_radio_setup_maybe(short *param_1)
3
4   {
5     short radio_mode;
6     int iVar1;
7     uint uVar2;
8     int iVar3;
9     int *piVar4;
10    int iVar5;
11    uint uVar6;
12    uint *puVar7;
13    char cVar8;
14    int iVar9;
15
16    radio_mode = *param_1;
17                    /* CMD_RADIO_SETUP Command ID  */
18    if (radio_mode == 0x802) {
19      cVar8 = *(char *)((int)param_1 + 0xf);
20      iVar9 = *(int *)(param_1 + 10);
21    }
22    else {
23                    /* 0x3806 is propietary mode at 2.4GHZ
24                       */
25      if (radio_mode == 0x3806) {
26        cVar8 = '\x02';
27      }
28      else {
29                    /* Generic propietary mode setup
30                       */
31        if (radio_mode != 0x3807) {
32          return 1;
```

Figure 17. Possible Radio Function in Ghidra

Since Wyze seems to use the proprietary radio commands, there is a high chance they follow the recommended packet format. TI specifies both a Standard Packet Format with fields for the preamble, sync word, header containing packet length and/or address, payload, and checksum, as well as an Advanced Packet Format with the same fields except that the header has more flexibility. Determining which format Wyze uses will be important.

With either format, the packet has to have a sync word so that the Sensor bridge can detect when a packet is about to be sent and capture it correctly.

E. CC13X0 Software Development Kit

A Software Development Kit (SDK) is a set of software tools that can be used to create applications for a specific platform. The Texas Instrument CC13X0 SDK offers flexible hardware, software, and tools for development of wired and wireless applications [21]. By comparing captured firmware to code in the SDK, it is determined that Wyze does use the CC13X0 SDK for their sensor bridge. The firmware appears to be using the RFCC26XX_multiMode.c driver suit, which can be used for both CC13XX chips and CC26XX chips. Additionally, a number of functions from the SDK relating to radio setup have been identified within Ghidra, including RF_fsmSetupState, RF_open, RF_fsmActiveState, and fsmPowerUpState.

F. Defining Structures

The radio in the CC1310 MCU is set up in proprietary mode by the command CMD_PROP_RADIO_DIV_SETUP. It is a structure which specifies details about the packet format and modes for the radio. The sync word can be between 8 and 32 bits and is set by formatConf.nSwBits [21]. Finding where this occurs in the firmware will help in determining the sync word. Since the command is a structure, the sync word length field is reached by offsets from the command address. Assuming Wyze uses the same command structure as detailed by TI, setting up the structure in Ghidra will make the decompiled code easier to read. The SDK contains C header files which define the radio commands.

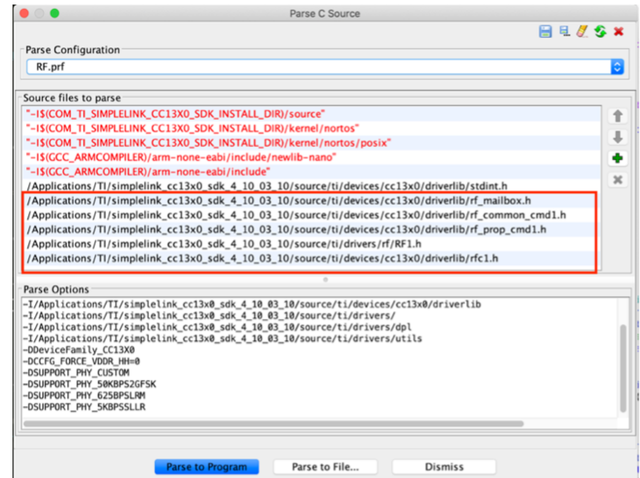

Figure 18. C Parser Configuration

The Ghira C Parser is able to parse header files and automatically create all the structures from the files. The C Parser is very finicky, so it is best to create copies of the files to parse, and simplify the code. This was done by removing any *#ifndef* or *#define* directives, and adding *typedef* before each structure that is defined. Then inside the C Parser, a new Parse

Configuration was created, adding the header files in the order in which they are referenced, and using parse options that match the compiler flags. The C Parser was run with many of the RF header files from the SDK as outlined in red in Figure 18, to define all of the structures related to the radio and RF protocol. Figure 19 shows the CMD_PROP_RADIO_DIV_SETUP structure from the rf_mailbox.h header file, and in Figure 20 the same structure defined in Ghidra by the C Parser.

Figure 19. CMD_PROP_RADIO_DIV_SETUP structure

Figure 20. CMD_PROP_RADIO_DIV_SETUP structure in Ghidra

Using the defined structures, variables in the code can be re-typed to make the decompiled code easier to read. In the example below, editing the function signature and parameter types makes it easier to see which fields in the structures are being set. This can be seen in Figure 21 and Figure 22 with the changes underlined in red.

Figure 21.Before Editing Function Signature

Figure 22. After Editing Function Signature

G. *Functions Relating to Packets*

Some of the functions found in Ghidra that relate to packets are shown below. More analysis of these functions will be helpful in determining the packet structure.

- 0000338c_main_loop_maybe
- 000014f0_seems_important_packet
- 00002d34_something_with_packets
- 00003324_big_switch
- 00014e8c_process_packet
- 0000ce6c_more_packet_processing
- 00012c4c_create_packet
- 00006e10_parse_packet

H. *Addresses*

In the functions relating to packets, there's no data structure for a packet being passed in as an argument. Instead, the functions seem to reference specific memory addresses, so it is likely that incoming and outgoing packets are stored in a buffer in memory. Analyzing which addresses and used and when can then be used to create a visualization of the packet fields in memory an idea of the structure of the packets. The following

details specific addresses and offsets that are accessed most frequently.

- 0x20001E60
  - Referenced in 00014e8c_process_packet, 00012c4c_create_packet, 0000ce6c_more_packet_processing, 00002d34_something_with_packets, 000014f0_seems_important_packet, and 00003324_big_switch
  - Explicitly uses the offset +0x134 to make address 0x20001F94 in 00014e8c_process_packet 00012c4c_create_packet, 0000ce6c_more_packet_processing, and 00002d34_something_with_packets
- 0x200012E8,
  - Referenced in 00002d34_something_with_packets, 000014f0_seems_important_packet, and 00003324_big_switch, and over 10 other unnamed functions
  - big_switch sets values at the address with varying offsets, possibly setting the fields in a packet
  - something_with_packets accesses the address with varying offsets, possibly reading fields from a packet

I. *Sensor Firmware*

The hardware used for the sensors is not locked and is able to be read/modified using the JTAG pins on the device [22]Er. Additionally, some sensors have the bootloader backdoor enabled, allowing a serial connection to be used to connect to the device [23]. Failing to disable these access points allows the firmware on the device to be modified and potentially allows malicious firmware to be loaded.

VII. Conclusion

A. *Main Takeaways*

Overall, the Wyze Camera and Wyze Sense has quite a few security vulnerabilities. Lack of sufficient security protocols led to a massive data breach compromising user information [6]. Lack of proper authentication methods and weak encryption means the Wyze Camera has a weak root password and is vulnerable to MITM attacks and the Wyze Sense is vulnerable to replay attacks, as shown by the previous and current semesters.

B. *Future Work*

Future research should focus on better understanding the packet structure, so that arbitrary packets can be formed and sent to the dongle. More analysis on the addresses mentioned in Section VI.H is a good place to start. Using a JTAG or the bootloader to read a sensors flash and SRAM would also likely give some insights. Another area of further research is if the AES capabilities on the chip are being actively used, and if so, how the keys are negotiated between devices.

In addition, further research on understanding the packet structure could be done by sorting through logs and gaining additional information of commands that have yet to be added to the parser. This can be done by cross-referencing a commands hex to functions relating to packets in Ghidra. So far, most of the commands have yet to be discovered. Students could also attempt to get OTA captures of communications between the contact sensor and compare it to those of the motion sensor and information that have already been gained from motion sensor packets.

VIII. References

[1] D. Hopwood, “Lack of security in Internet of Things devices,” *Network Security*, vol. 2014, no. 8, p. 2, Aug. 2014.

[2] “IoT is Coming Even if the Security Isn't Ready: Here's What to Do,” *Wired*, 07-Sep-2017. [Online]. Available: https://www.wired.com/brandlab/2017/06/iot-is-coming-even-if-the-security-isnt-ready-heres-what-to-do/. [Accessed: 22-Apr-2021].

[3] “Wyze Sense,” *Wyze*. [Online]. Available: https://wyze.com/wyze-sense.html. [Accessed: 22-Apr-2021].

[4] “Technical Reference Manual,” *TI*, Feb-2015. [Online]. Available: https://www.ti.com/lit/ug/swcu117i/swcu117i.pdf?ts=1612380468517&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FCC1310. [Accessed: 22-Apr-2021].

[5] “CC1310 SimpleLinkTMUltra-Low-Power Sub-1 GHz Wireless MCU,” *TI*, 2015. [Online]. Available: https://www.ti.com/product/CC1310. [Accessed: 21-Feb-2021].

[6] B. Lovejoy, “Wyze camera security breach: personal data from 2.4M users,” *9to5Mac*, 30-Dec-2019. [Online]. Available: https://9to5mac.com/2019/12/30/wyze-camera-security/. [Accessed: 20-Mar-2021].

[7] D. Wroclawski, “Wyze and Guardzilla Security Cameras Have Security Risks, Consumer Reports Finds,” *Consumer Reports*, 05-Nov-2019. [Online]. Available: https://www.consumerreports.org/wireless-security-cameras/wyze-and-guardzilla-home-security-cameras-have-security-risks/. [Accessed: 18-Mar-2021].

[8] “Vulnerability Details : CVE-2013-1059,” *CVEDetails*, 01-Apr-2014. [Online]. Available: https://www.cvedetails.com/cve/CVE-2013-1059/. [Accessed: 22-Apr-2021].

[9] “Vulnerability Details : CVE-2017-5972,” *CVEDetails*, 19-Feb-2019. [Online]. Available: https://www.cvedetails.com/cve/CVE-2017-5972/. [Accessed: 22-Apr-2021].

[10] *IoT Security: Backdooring a smart camera by creating a malicious firmware upgrade*. YouTube, 2020.

[11] Hclxing, “Reverse Engineering WyzeSense bridge protocol (Part II),” *My Not-So-Boring Life*, 06-Jun-2019. [Online]. Available: https://hclxing.wordpress.com/2019/05/30/reverse-engineering-wyzesense-bridge-protocol-part-ii/. [Accessed: 22-Apr-2021].

[12] HclX, “HclX/WyzeSensePy,” *GitHub*. [Online]. Available: https://github.com/HclX/WyzeSensePy. [Accessed: 22-Apr-2021].

[13] “Packet Format - SimpleLink™ CC13x0 SDK Proprietary RF User's Guide 2.60.1 documentation,” *Texas Instruments*. [Online]. Available: http://software-dl.ti.com/simplelink/esd/simplelink_cc13x0_sdk/4.10.02.04/exports/docs/proprietary-rf/proprietary-rf-users-guide/proprietary-rf/packet-format.html. [Accessed: 22-Apr-2021].

[14] G. Christiansen, “Data Whitening and Random TX Mode,” *Design Note DN509*. [Online]. Available: https://www.ti.com/lit/an/swra322/swra322.pdf. [Accessed: 22-Apr-2021].

[15] *How mitmproxy works*. [Online]. Available: https://docs.mitmproxy.org/stable/concepts-howmitmproxyworks/. [Accessed: 22-Apr-2021].

[16] “Ghidra,” *National Security Agency*. [Online]. Available: https://www.nsa.gov/resources/everyone/ghidra/. [Accessed: 22-Apr-2021].

[17] https://www.keil.com/pack/doc/CMSIS/SVD/html/index.html#:~:text=The%20CMSIS%20System%20View%20Description,data%20in%20device%20reference%20manuals

[18] *System View Description*. [Online]. Available: https://www.keil.com/pack/doc/CMSIS/SVD/html/index.html#:~:text=The%20CMSIS%20System%20View%20Description,data%20in%20device%20reference%20manuals. [Accessed: 22-Apr-2021].

[19] *Bare-metal ARM firmware reverse engineering with Ghidra and SVD-Loader*. 27-Feb-2020 [Online]. Available: https://www.youtube.com/watch?v=q4CxE5P6RUE. [Accessed: 2-11-2021]

[20] Leveldown-Security, "leveldown-security/SVD-Loader-Ghidra," *GitHub*. [Online]. Available: https://github.com/leveldown-security/SVD-Loader-Ghidra. [Accessed: 22-Apr-2021].

[21] "SIMPLELINK-CC13X0-SDK," *SIMPLELINK-CC13X0-SDK Software development kit (SDK) | TI.com*. [Online]. Available: https://www.ti.com/tool/SIMPLELINK-CC13X0-SDK. [Accessed: 22-Apr-2021].

[22] Sycophantic, "Sycophantic/Wyzeback," *GitHub*. [Online]. Available: https://github.com/sycophantic/wyzeback. [Accessed: 22-Apr-2021].

[23] Null, "Unbricking Wyze Contact Sensor - pcb reset pin," *Wyze Community*, 04-Feb-2021. [Online]. Available: https://forums.wyzecam.com/t/unbricking-wyze-contact-sensor-pcb-reset-pin/146856/130. [Accessed: 22-Apr-2021].