Issues in Information Systems
Volume 23, Issue 4, pp. 265-278, 2022

DOI: https://doi.org/10.48009/4_iis_2022_123

An investigation of previously undiscovered security vulnerabilities of common household IoT devices

Joseph V. Homan, *Stephenson Technologies Corporation (LSU), jhoman@stc-ntc-lsu.org*
Ryan Smith, *Stephenson Technologies Corporation (LSU), rsmith@stc-ntc-lsu.org*
Michael Marchese, *Stephenson Technologies Corporation (LSU), mmarchese@stc-ntc-lsu.org*
Philip Kim, *Walsh University, pkim@walsh.edu*

Abstract

Convenience versus privacy is a choice many must make as we rely more on connected devices, online services, and software to make our lives easier. There is well-documented literature about the risks associated with sharing personal information and preferences on social media sites and sharing personal interests on websites, thereby allowing targeted advertising and better information delivery. Less understood are the risks associated with smart devices, referred to as Internet of Things (IoT) devices. As with almost any technology solution, IoT devices open the door to potential danger. Risks associated with physical IoT devices are not as well documented. While some risks are small and of minimal threat, some open the user to real personal danger, including unfettered physical access into their homes. This paper documents a primary research effort to investigate and document multiple threat vectors associated with standard consumer IoT devices.

Keywords: Internet of Things (IoT) Hacking, Cybersecurity Risks, Physical Security, Awareness

Introduction

There is a rapidly increasing prevalence of small form factor internet-connected devices, commonly referred to as Internet of Things (IoT) devices. Unfortunately, the growth rate in the IoT sector is outpacing growth in the corresponding IoT security sector. There were an estimated 10 billion active IoT devices in 2021. The estimated increase by 2030 is projected to be over 25 billion IoT devices (Jovanovic, 2022). By design, many consumer IoT devices are set to continually monitor and collect data to be prepared to respond to a user's request or action. This constant connectivity may provide a more satisfying end-user experience but also increases the potential for security breaches (Wurm et al., 2016). The result is many vulnerable IoT devices across private networks providing points of entry for opportunistic attackers. According to a 2018 IoT Cybersecurity Readiness Report by Trustwave, 61% of organizations have experienced an IoT security incident (Trustwave, 2018).

We tested more than 30 IoT devices for cyber or physical vulnerabilities through a project funded by the US Air Force Research Laboratory. Engineers and analysts conducted the research at Stephenson Technologies Corporation (STC), a non-profit affiliate of Louisiana State University. In part, this research effort aims to identify potential threats created by IoT devices, inform the manufacturers of the vulnerabilities, and provide awareness to users and consumers of these devices.

Acknowledging the increased prominence of IoT devices and the increased risk related to IoT security gaps (Meneghello et al., 2019), our organization dedicates a subsegment of its research efforts towards IoT

device security and vulnerability analyses. This paper presents various research case studies associated with IoT technology focus areas and details related research and experimentation findings.

The operating environment for this research is an "IoT Living Lab" – a model home in a new housing development in Baton Rouge, LA. STC outfitted the house with standard consumer IoT devices in conjunction with the building contractor. An earlier testing facility in a more traditional laboratory setting was not conducive to real-world testing that would simulate a typical residential neighborhood. The IoT Living Lab environment was designed to facilitate IoT research and experimentation in a location closely reflecting the home of a typical residential user of IoT devices. This living lab is a user-centric, iterative, open-innovation ecosystem that allows engineers to test new and emerging devices in an emulated real-world context.

Research and Experimentation Domains

The research focused on four IoT technology areas:

1) Smart Assistant Research
2) Wi-Fi Camera Research
3) QR Code Research
4) Home Security System Research

Except for the QR code research, which we document in a separate forthcoming paper, the following sections detail case studies for each focus area and describe methods, findings, and discussion elements.

Smart Assistant Research (Amazon Alexa Case Study)

Amazon Alexa devices are prevalent among personal and corporate networks, making them an ideal attack vector. According to a survey conducted by Armis of their clients, 82% of companies surveyed had an Amazon device in their corporate networks (Armis, 2017). The popularity of IoT devices and smart assistant devices, in conjunction with the rapid growth in the IoT sector, demands research into the numerous potential security threats posed by IoT smart assistant devices in organizations.

We conducted primary research to develop methods of exploiting Amazon Alexa devices. The team gained remote access to an Amazon account in the experiments detailed below. We utilized this access to remotely control devices and configurations on smart devices attached to the Amazon account.

Research Goal(s) and Objectives

Goal 1: Remotely control Amazon Echo devices

Goal 1 objectives:

1. Instantiate a software utility to leverage the Amazon Alexa Application Programming Interface (API)
2. Connect software utility to test Amazon account
3. Send remote commands to Amazon Alexa API

Methods

The research team developed methods for remotely controlling Amazon Alexa devices through two phases of research. The first phase involved searching for community-driven knowledge and tools (Programmable Web, n.d.) and investigating the public API utilized by these tools (Alexa Developer, n.d.). The next phase involved developing routines and scripts for pulling data and controlling devices attached to Amazon accounts using the public API.

In the research phase of the experiment, we discovered that a public API exists as an interface for Amazon Alexa devices. This API allows for remotely executing tasks that Amazon Alexa devices control. Building upon a community-sourced script, our researchers could remotely control the devices using a Command Line Interface (CLI) in the second phase of the experiment.

Table 1. Experiment Procedures

Step	Event	Outcome
1	Remote control script downloaded onto Ubuntu Server 18.04 Virtual Machine	Rogue host established to perform experiment tests
2	Remote control script configured to utilize Amazon test account credentials	Control script linked to target test account and can send command functions to the target API
3	Text-to-speech (TTS) commands sent to Amazon Echo devices via remote control script	Amazon Echo devices responded by repeating typed phrases
4	Alexa test routines were created	Automations created in the Amazon API to act as footholds for remote command and control
5	Automation commands sent to activate test routines	Amazon Echo devices responded by performing targeted test routines.

STC researchers set the test account credentials and configuration options (including language and Amazon domain) in the alexa_remote_control.sh file as shown in Figure 1. The multifactor authentication (MFA) information was obtained from amazon.com by logging into the test Amazon account and navigating to the "Login & security" page.[1] Enabling MFA in the amazon_remote_control.sh file improved the stability of results.

```
SET_EMAIL='1
SET_PASSWORD='
SET_MFA_SECRET='3

SET_LANGUAGE='en-US'


SET_TTS_LOCALE='en-US'

SET_AMAZON='amazon.com'


SET_ALEXA='alexa.amazon.com'

SET_CURL='/usr/bin/curl'

# cURL options
#  -k : if your cURL cannot verify CA certificates, you'll have to trust any
#  --compressed : if your cURL was compiled with libz you may use compression
#  --http1.1 : cURL defaults to HTTP/2 on HTTPS connections if available
SET_OPTS='--compressed --http1.1'
#SET_OPTS='-k --compressed --http1.1'

# browser identity
BROWSER='Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36'

# oathtool command line tool
SET_OATHTOOL='/usr/bin/oathtool'

# tmp path
SET_TMP="/tmp"

# Volume for speak commands
SET_SPEAKVOL="100"
# if no current playing volume can be determined, fall back to normal volume
SET_NORMALVOL="10"
```

Figure 1. Configuration options for alexa_remote_control.sh

We developed a series of basic routines for testing the remote command capabilities of the alexa_remote_control.sh script. Figure 2 illustrates these routines. These routines were to be called later to elicit TTS actions or automated routines from an Amazon Echo device linked to the test account.

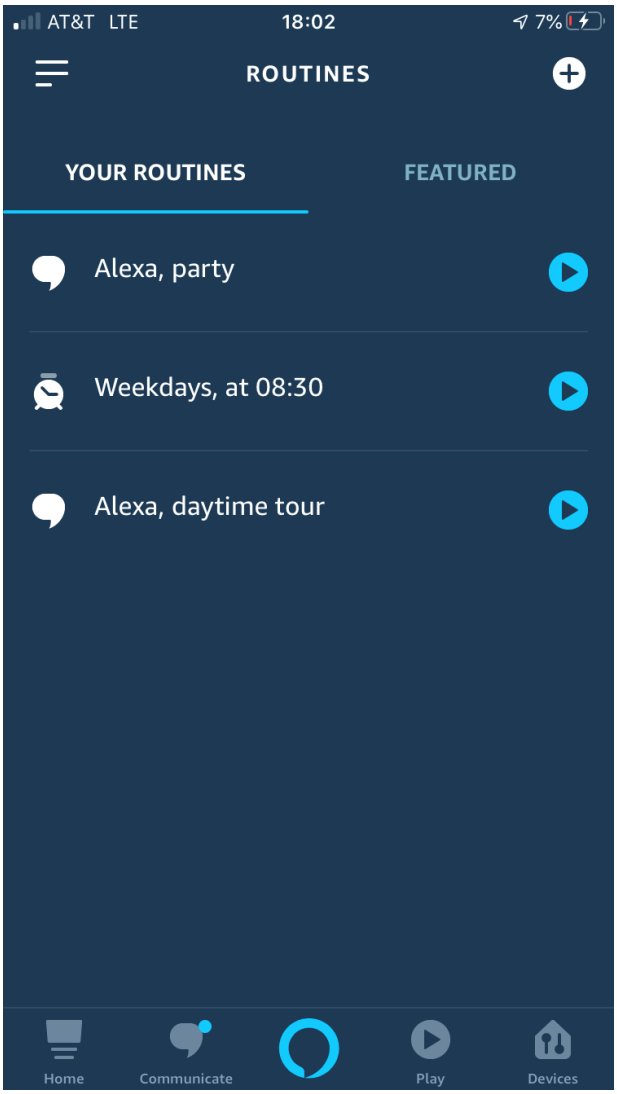

Figure 2. Alexa Test Routines

In the research phase of this experiment, we discovered an undocumented API at https://alexa.amazon.com/api/. Amazon Echo devices interact with this API when processing voice commands. One route in this API, https://alexa.amazon.com/api/behaviors/automations, returns all native and created Amazon Alexa automations. Figure 3 shows the formatted JavaScript Object Notation (JSON) of the "goodnight" automation, which is an automation Alexa supports by default.

```
{
    "@type": "com.amazon.alexa.behaviors.model.Automation",
    "automationId": "amzn1.alexa.behaviors.preconfigured:good_night_custom_utt_trigger",
    "name": null,
    "triggers": [
        {
            "payload": {
                "marketplaceId": "A[redacted]",
                "customerId": "A[redacted]",
                "locale": "en-US",
                "utterance": "goodnight"
            },
            "id": null,
            "type": "CustomUtterance",
            "timeLapse": null,
            "skillId": null
        }
    ],
    "sequence": {
        "@type": "com.amazon.alexa.behaviors.model.Sequence",
        "sequenceId": "amzn1.alexa.sequence.[redacted]",
        "startNode": {
            "@type": "com.amazon.alexa.behaviors.model.SerialNode",
            "name": null,
            "nodesToExecute": [
                {
                    "@type": "com.amazon.alexa.behaviors.model.OpaquePayloadOperationNode",
                    "type": "Alexa.GoodNight.Play",
                    "skillId": null,
                    "operationPayload": {
                        "deviceType": "ALEXA_CURRENT_DEVICE_TYPE",
                        "customerId": "A[redacted]",
                        "locale": "en-US",
                        "deviceSerialNumber": "ALEXA_CURRENT_DSN"
                    },
                    "name": null
                },
                {
                    "@type": "com.amazon.alexa.behaviors.model.OpaquePayloadOperationNode",
                    "type": "Alexa.Operation.SkillConnections.Launch",
                    "skillId": "amzn1.ask.skill.[redacted]",
                    "operationPayload": {
                        "customerId": "A[redacted]",
                        "targetDevice": {
                            "deviceType": "ALEXA_CURRENT_DEVICE_TYPE",
                            "deviceSerialNumber": "ALEXA_CURRENT_DSN"
                        },
                        "locale": "en-US",
                        "connectionRequest": {
                            "uri": "connection://AMAZON.Launch/amzn1.ask.skill.[redacted]"
                        }
                    },
                    "name": null
                }
            ]
        }
    },
    "owner": null,
    "clientAlias": null,
    "condition": null,
    "status": "DISABLED",
    "creationTimeEpochMillis": 1579877226343,
    "lastUpdatedTimeEpochMillis": 1579877226343
},
```


Figure 3. Example automation JSON

Using Curl commands, the alexa_remote_control.sh script generated POST requests to https://alexa.amazon.com/api/behaviors/preview that simulated what the Amazon API was expecting. As shown in Figure 4, the command parameters stored in the variable ALEXACMD are first echoed to a file. The Curl command transfers this file to the API through the Hypertext Transfer Protocol (HTTP) POST method.

```
echo $ALEXACMD > "${TMP}/.alexa.cmd"

${CURL} ${OPTS} -s -b ${COOKIE} -A "${BROWSER}" -H "DNT: 1" -H "Connection: keep-alive" -L\
 -H "Content-Type: application/json; charset=UTF-8" -H "Referer: https://alexa.${AMAZON}/spa/index.html" -H "Origin: https://alexa.${AMAZON}"\
 -H "csrf: $(awk "\$0 ~/.${AMAZON}.*csrf[ \\s\\t]+/ {print \$7}" ${COOKIE})" -X POST -d @"${TMP}/.alexa.cmd" \
 "https://${ALEXA}/api/behaviors/preview"
```

Figure 4. Example Curl command

Findings

The experiments conducted by the research team led to the discovery of an undocumented Application Programing Interface (API) used by Amazon Alexa devices. Using these APIs, we could remotely activate native Alexa automations (including traffic, flash briefing, weather, start my day, and every other default Alexa automation) and custom Alexa routines. The researchers also could POST TTS commands eliciting custom spoken phrases from the Amazon Echo and Echo Dots. Furthermore, Alexa devices connected to the test account could be enumerated from the discovered Amazon API, and the TTS commands could be sent to any of the Alexa devices registered to the test account. This vulnerability allowed researchers to simultaneously send custom speech to any Amazon Echo device linked to the test account or multiple Echo devices.

STC researchers successfully sent all commands from a CLI without utilizing human voice recordings. This meant that all commands used the API directly, and the speech commands were elicited by submitting typed phrases to the Amazon API. The alexa_remote_control.sh script was successful regardless of the location of the remote host activating the call as long as the Curl was sending correct data to the Amazon API.

Discussion

Our experiments showed that any Alexa automated actions could be remotely initiated from a CLI outside the target network. The connection can be made with or without multifactor authentication (MFA), and the control can be initiated without alerting the owner of the Amazon device. If the user has MFA enabled, we can gain permanent access to their MFA secret once we get access to the account at any point. This allows us to circumvent MFA after the initial compromise.

The experiment reveals the device owner's inability to control access and calls into question the ability to prevent malicious actions in a target household. If custom routines and default automations can be initiated remotely, it is also possible that third-party skills and integrations with third-party equipment can be controlled as well.

The research team plans to leverage this research to see if the discovered API can be used to extend third-party skills and integrations to perform malicious physical actions within a target environment. Future research explores the security model and architecture of the shared ecosystem between Amazon devices and third-party software. Even the most security-conscious consumer has difficulty evaluating security risks associated with a single device, let alone entire ecosystems (Sequeiros et al., 2020). An ecosystem is only as strong as its weakest link. With Amazon Alexa's growing influence in the IoT market, potential risks in the Alexa ecosystems could put numerous consumers and businesses at risk (Asplund & Nadjm-Tehrani, 2016). Further research should be performed to evaluate the security model of the Amazon Alexa system of systems.

Wi-Fi Camera Research (TRENDnet IP110w)

In this second research area, our team identified a shell code injection vulnerability that can be used to exploit TRENDnet and TP-Link model cameras. In this experiment, we attempt to leverage this shell code injection vulnerability to extract user information and network credentials from a vulnerable camera. The

following demonstrates how the vulnerable cameras were exploited and suggest how this exploit can be used to gather information and gain persistence within a target network.

Goals and Objectives

The following are the goals of this experiment:

- Identify vulnerabilities in IoT cameras
- Attempt to exploit weaknesses in IoT cameras to achieve remote access or control
- Identify how IoT camera exploits can be used as pivot points to other assets

The following are the objectives of this experiment:

- Analyze IoT camera firmware for backdoors and vulnerabilities
- Examine what information is stored by the camera and how it is secured
- Identify software and services for identifying vulnerable IoT cameras, reverse engineering firmware, and testing web vulnerabilities.
- Research common vulnerabilities of cameras from different vendors

Methods

The research team developed exploits through two phases of research. In the first phase, the camera firmware was obtained from the manufacturer's website and reverse engineered using Ghidra and standard Linux file system tools. We could extract information from the analyzed firmware that supported the next research phase. Such information included passwords, device architecture, and insight into the device operations. The next phase involved identifying vulnerabilities and creating exploits based on the information found in phase one. Automated tools were created for faster execution for vulnerabilities confirmed to be exploitable.

In the project's research phase, our team discovered an HTTP server built into the camera that makes several calls to the underlying Linux system on the camera. One of these calls directly passes a user-provided parameter, "action," as an argument for a system call. The camera's firmware runs the sent parameter in a pre-built command as the root user. By prepending and appending a semicolon, our researchers were able to inject an arbitrary command and have the camera firmware run it as the root user. We then used this discovery as the basis for creating a tool for a remote Command Line Interface (CLI) used to control target cameras that utilize the researched firmware version.

Table 2. Experiment Procedures

Step	Event	Outcome
1	The device's filesystem is obtained by downloading the firmware from the vendor's website. The firmware is then extracted using Binwalk and analyzed using Ghidra.	The GET parameter "action" on the "/cgi/maker/unittest.cgi" web page is identified as a potential target for command injection.
2	The URL and GET parameters identified in the research phase are tested using Curl to send the camera shell injection code for a list files command.	The HTTP server built into the camera responds with HTTP headers and a "not authorized error."
3	The URL and its GET parameter identified in the research phase are tested using Curl to authorize as the backdoor user account "productmaker:ftvsbannedcode" when sending the camera shell injection code for a list files command.	The HTTP server built into the camera responds with HTTP headers and a listing of files on the camera.
4	The camera is sent a "whoami" command at that address via the same shell code injection technique in step 2	The HTTP server built into the camera responds with HTTP headers and one line of text confirming that the commands injected are run as the root user.
5	Using the CLI tool to explore the filesystem on the camera, a usr.ini file containing the username and passwords of all users on the camera in Base64 encoding was found. These passwords are decoded using the base64 utility on Linux.	We now have usernames and passwords used by the target camera, potentially allowing access to other devices on the network that reuse passwords.
6	Further exploration of the filesystem found a configuration file in the HTTP server directory containing the Service Set Identifiers (SSID) and passwords of all wireless networks the camera is configured to access.	We now have a listing of Wi-Fi networks and passwords for further access to the networks used by the target.
7	Using the knowledge from the above steps, our team created a bash script that acts as a Command Line Interface on the camera by taking the following steps: • Accept a user command • Convert it to a URL escaped string with semicolons concatenated to the beginning and end • Pass the command as the "action" parameter to the specified URL on the camera HTTP server • Strip the HTTP headers from the server response and then display the response to the user	We can now generate a reverse shell-like interface that can execute shell code on most TRENDnet and TP-link cameras designed in the mid-2010s and before.

Findings

The experiments conducted by the research team led to the discovery of critical shell code injection exploits in common TRENDnet and TP-link cameras. This exploit is run as the root user on the device, giving access to everything on the camera. Additionally, a hard-coded backdoor account was discovered in the device's firmware, allowing access to any camera regardless of the configuration. This discovery led to creating a CLI tool for remotely controlling the camera in a shell-like manner. From here, methods were developed for quickly gathering the device's Wi-Fi and camera account credentials to facilitate further actions.

We obtained the firmware for the TRENDnet camera by downloading from the official TRENDnet website. STC researchers then extracted the filesystem from the firmware by using Binwalk. Once extracted, the filesystem was mounted as a directory using WSL, and the files were analyzed.

```
63          iVar3 = system(sysString);
64          FUN_000088e4(*(undefined4 *)(iVar1 + 0x48),&DAT_00008e24,iVar3,
65                       *(undefined4 *)(iVar2 * 4 + *(int *)(iVar1 + 0x40) + 0x104));
66          system("/etc/init.d/savecfg.sh 1>/dev/null 2>/dev/null&");
67        }
68        goto LAB_00008a50;
69      }
70      iVar3 = strcmp(__s1,"test");
71      if (((iVar3 != 0) && (iVar3 = strcmp(__s1,"normal"), iVar3 != 0)) &&
72         (iVar3 = strcmp(__s1,"system"), iVar3 != 0)) goto LAB_00008a50;
73      snprintf(sysString,0x80,"/etc/init.d/unitest.sh %s 1>/dev/null 2>/dev/null",
74               *(undefined4 *)(iVar2 + iVar5 + 0x104));
75      iVar3 = system(sysString);
76      puVar4 = *(undefined **)(iVar2 + *(int *)(iVar1 + 0x40) + 0x104);
77      goto LAB_00008a44;
78    }
79    snprintf(sysString,0x80,"/etc/init.d/unitest.sh rtc 1>/dev/null 2>/dev/null");
80    iVar3 = system(sysString);
```

Figure 5. Finding Shell Injection Using Ghidra

Using Ghidra, our researchers discovered the shell injection vulnerability by searching for calls to "system" in the web backend files. The highlighted function call shown in Figure 5 is used to prepare a string for a call to the underlying Linux system. The function prepares a system call by combining a prebuilt command with a string directly taken from the URL GET parameters without input validation. This makes it easy to exploit the target with command injection attacks by using semicolons to escape out of the shell command shown and replacing it with a custom shell command.

The next part of the experiment was verifying the shell code injection exploit discovered in the research phase. To do this, a small network consisting of a TP-Link TL-5G108 switch, a client (in this experiment, a Microsoft Surface laptop was used), and the TRENDnet IP110w Camera. All devices were connected via Cat 5 ethernet cables plugged into the switch for this test. The first step involved accessing the camera's HTTP interface to verify that the device was password protected. We added Wi-Fi credentials for the future wireless and credential exfiltration tests.

```
curl -s -u "$user":"$pass" "$port$ip$url;${input// /%20};" | sed 1,1d | head -n -6 ;;
```

Figure 6. Example of Curl Command

Using Curl (figure 6), we attempted to access the web URL at "/cgi/maker/unittest.cgi" using no authentication. This resulted in a 401 response back from the server. We then went into the filesystem previously extracted from the firmware in the research phase and found the Url.ini file. This file dictates the permissions for directories on the web server. We noticed that not only was this URL linked to a second account, "productmaker," this account had a backdoor password that the user could not change. We then

used this account and gave the argument ";;" to the action parameter to access the web URL, resulting in an HTTP response containing all built-in system commands.

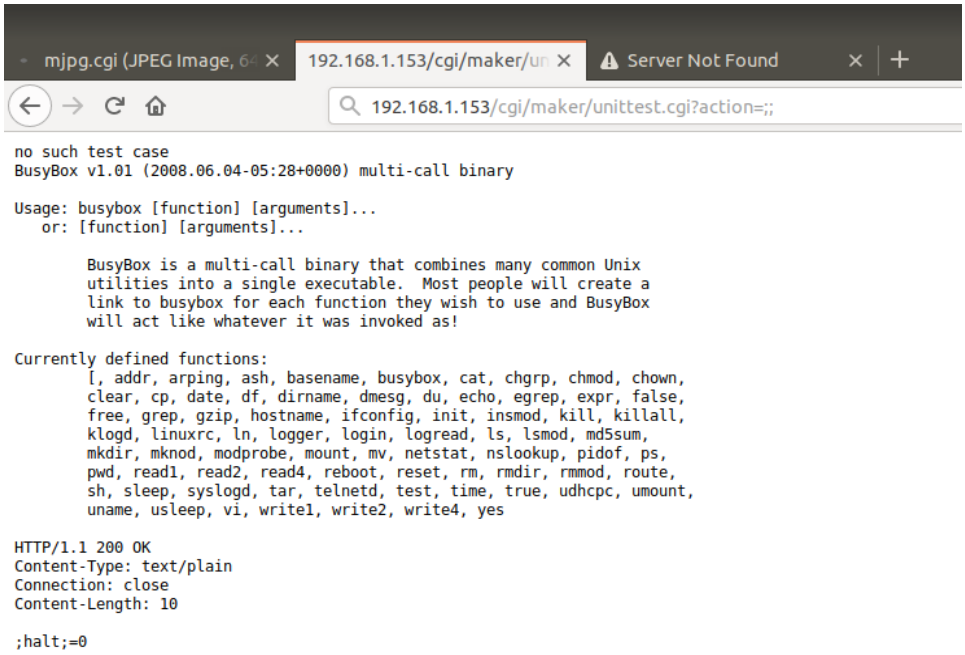

Figure 7. An Example Response from Command Injection

In summary, it is possible to access the webpage shown in Figure 7 without knowing the admin user credentials, utilize the command injection exploit to run any command on the underlying system as root, and get the standard output of the command back in the form of an HTTP response.

When we investigated the file system used by the target camera, we discovered multiple files that store sensitive user information in plain text. The "usr.ini" file, which contains all accounts with names and passwords for the camera, was encoded in Base64 with the standard encoding string, meaning decoding these credentials is trivial. Figure 8 illustrates the process of retrieving and decoding user accounts.

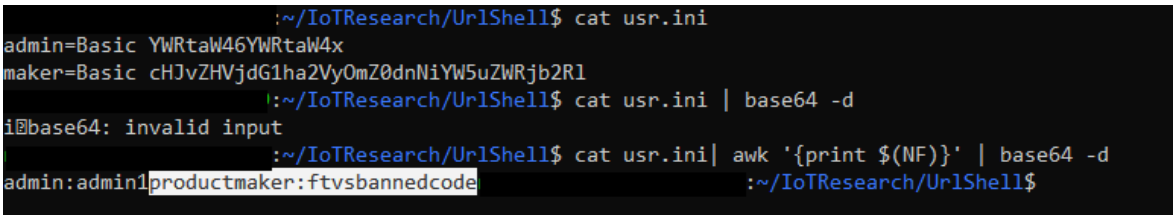

Figure 8. Retrieving and Decoding User Accounts

The network configuration file stores all Wi-Fi network credentials for the camera in plain text. A "cat" command passed with the command injection URL retrieves all the credentials.

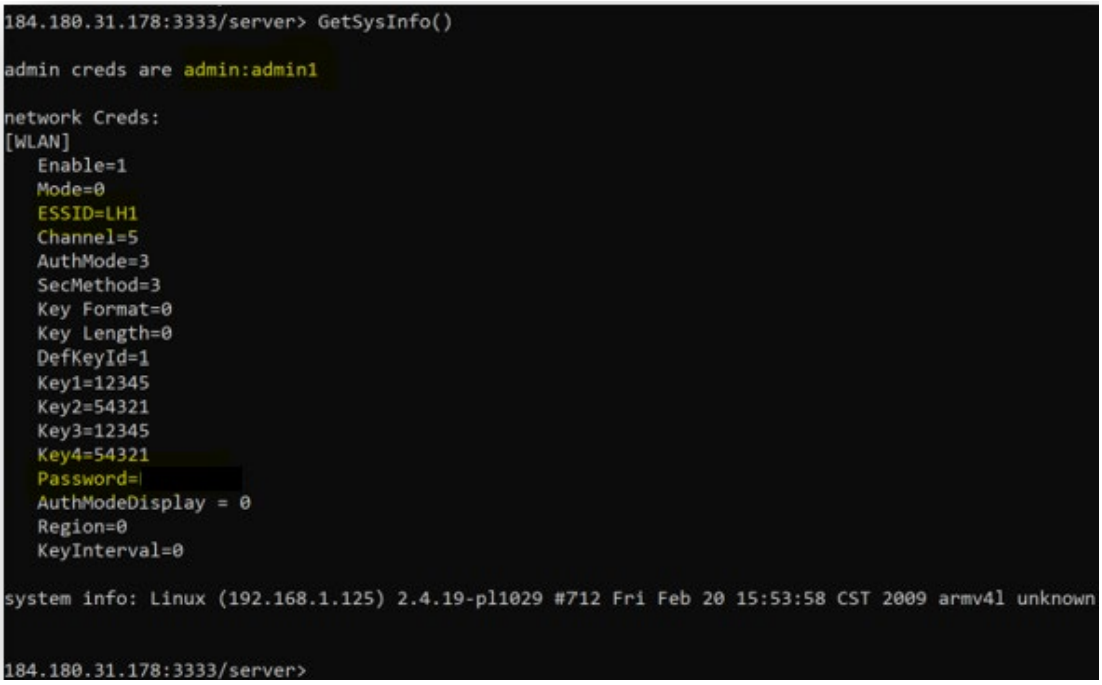

Figure 9. Using CLI Tool to Get All Device Info

Building on the previous exploits, our researchers created a utility similar to a reverse shell for persistent control over the camera. This utility establishes a pseudo-connection to the camera using the ping utility. If the camera responds, the utility takes in user commands, packaged as a URL encoded string, and sends them to the camera using the URL previously established. The response is then stripped of the HTTP header and footer info and displayed to the user. The utility also has a built-in method called “GetSysInfo," which outputs network and user info contained in the camera’s "net.conf” and “usr.ini” files, respectively (figure 9).

Discussion

The experiment succeeded in achieving all outlined goals. All cameras that share firmware with the cameras tested in this experiment are vulnerable due to a backdoor account found in the reverse engineering phase. Further research found that many of these cameras are open to outside web traffic, so the camera owner can remotely monitor them. Popular website guides demonstrate how to quickly set up these cameras to be open to outside web traffic, often suggesting ports 3333 and 3334 for this purpose. Therefore, vulnerable cameras are freely accessible on the web and unprotected from threat actors.

Though most cameras manufactured around the same time from TRENDnet and TP-link share this vulnerability, some use a slightly different backdoor account or have this vulnerable webpage on a different URL. These pieces of information may require reverse engineering of the specific model of the camera to find. These differing parameters can be quickly changed in the tool created by our team for reuse. The reverse shell utility created for exploiting all cameras that use this common firmware has additional use for attacks against new targets. The utility can be utilized to gain an easy-to-use reverse shell to any camera with an exposed API with a command injectable call. Additional features built into this utility can allow for transferring files to target devices that do not have file transfer utilities installed.

The resultant tool allows for further experiments on the device for tasks such as pivoting to other network devices or subtly disabling the camera’s functionality. The information discovered from exploring the file system led to multiple opportunities to pivot from the camera to other networks or accounts. Due to this

tool being built over a pseudo connection to the camera, it allows for a much stealthier command and control technique. Additionally, all commands being passed in the URL look like standard HTTP traffic to any low-level monitoring. This technique does have a significant weakness in that all commands and responses are in plain text, making this attack likely to be caught by any sophisticated monitoring solution.

Home Security System Research (SimpliSafe)

The test setup involved a test bench containing the SimpliSafe essential components (the base station and keypad), the remote key fob for the SimpliSafe system, a programmable HAM radio, an iPad tablet with the SimpliSafe mobile application installed, and a door sensor attached to an interior door.

Goals and Objectives

The following are the goals of this experiment:

- Identify vulnerabilities in operations of the sensor device
- Identify vulnerabilities in the communications from sensor devices to the controlling device
- Identify how any vulnerabilities discovered can be used to circumvent the security system

The following are the objectives of this experiment:

- Create a method of disabling or interfering with the operations of the sensor device
- Create a simple to use method of circumventing the security system

Methods

The consumer security system must be operational to test its functionality. This was completed via downloading the app from Apple's App Store, registering an account, and associating the device with the account. The device was then paired with the base station device as required. The system was verified to work by testing the sensor's function on a test bench. To verify this, the sensor was affixed to the door, and the magnet was simulated to close and open. The unit triggered a notification and a chime noise from the base station as the magnet was moved close enough to simulate the door closing. The sensor and magnet triggered the door to open/close approximately 2 inches apart, consistent with the manufacturer's documentation. After the sensor was triggered, the system recognized the device was opened, and a notification was sent to the user via the SimpliSafe mobile application.

While the security system was still operational, an interfering HAM radio transmitter was turned on and had its frequency set to 433.92 MHz. Approximately 10 seconds after starting the radio interference, the SimpliSafe system sends a push notification to notify that radio interference has been detected. While holding down the transmit button on the radio, the door was opened, and the blue light on the sensor went off to show it attempted to transmit the "door open" signal to the base station. However, the base station did not receive any signal in any attempts where the HAM radio was broadcasting within 200 ft of the base station. After another 10 seconds from when the interfering radio is turned off, the SimpliSafe system sends another push notification stating that the interference is no longer detected.

Correct use of the HAM radio prevents the base station from receiving any signal on the 433.92 MHz band. This interference renders the door sensors, motion detectors, and any other device that reports to the base station on the 433.92 MHz band useless. Although not tested on other auxiliary devices as of this writing, this implies that all SimpliSafe auxiliary devices that utilize this radio band are vulnerable to the same or similar attacks.

The system sends a notification that the wireless interference is detected after the HAM radio is used to broadcast. When the door is opened, the system does not set off its alarm, even after the radio is no longer broadcasting and the door is left open. The door is then closed and re-opened while the radio is set down

and no longer broadcasting. Opening the door without the radio causes the system to set off the alarm, showing the system functions as intended when the interfering radio is not used.

Regarding the interfering HAM radio, our researchers discovered that the radio needed to broadcast for approximately seven seconds after the door sensor was triggered to guarantee that the base station did not set off the alarm. This situation is assumed to be due to the residual signal from the sensor taking about that amount of time to dissipate enough to the point that the base station cannot read it.

Findings

Step #	Pass/Fail Criteria	Test Result	Test Measurement/Test Comments
1	System is operational	Pass	When the contact is broken, the sensor sends an alert.
2	The sensor alerts to the presence of interfering radio frequencies	Pass	With the interfering HAM radio transmitting, the system alerts the user to the presence of interference.
3	With the interfering radio in place, is the system still operational	Fail	When the interfering radio is in place, does the sensor chime and communicate to the base station.

Discussion

The primary issues with the current implementation stem from the system using only one frequency for all device communication. The system does not verify that signals sent from the sensor to the base station are received. Several options are available for the product to address the lack of verification.

1. The system could be configured to set off the alarm if an interfering device is present for a set period.
2. The sensor could be programmed to resend the transmission to the base station at a set interval if it does not get an "acknowledgment" signal.
3. The system could be configured to utilize frequency hopping to increase the difficulty of performing a jamming-based attack against it.

Conclusion

Consumers enjoy the convenience of smart devices, often unaware of the risks they may create (Zhang et al., 2014). The risks are real and may include the danger of threat actors hacking into video feeds and burglars breaking into a home undetected. Given these devices' widespread use and growing adoption, consumers must be aware of these security vulnerabilities (Anggorojati, 2019; Harper, 2016). This primary research effort, partially funded by an Air Force Research Lab contract and conducted by Stephenson Technologies Corporation, uncovered the vulnerabilities identified in this paper and more. Our researchers follow the "responsible disclosure" protocol in informing the manufacturers of the hardware and software containing vulnerabilities to enable them to address the issues. The limitations of this study included a focused scope of specific IoT devices installed in the "IoT Living Lab." Future research continues with other smart IoT devices such as faucets, microwave ovens, refrigerators, gas grills, baby monitors, thermostats, insulin pumps, pacemakers, radios, and other medical, industrial, and household appliances.

References

Alexa Developer Documentation. (n.d.). Alexa Smart Properties. https://developer.amazon.com/en-US/docs/alexa/alexa-smart-properties/alexa-for-residential-get-started.html

Anggorojati, B. (2019, July). Government efforts toward promoting IoT security awareness for end users: A study of existing initiatives. In European Conference on Cyber Warfare and Security (pp. 692-XV). Academic Conferences International Limited.

Asplund, M., & Nadjm-Tehrani, S. (2016). Attitudes and perceptions of IoT security in critical societal services. IEEE Access, 4, 2130-2138.

Armis. (2017). BlueBorne Cyber Threat Impacts Amazon Echo and Google Home. Available: https://www.armis.com/resources/iot-security-blog/blueborne-cyber-threat-impacts-amazon-echo-google-home/.

Harper, A. A. (2016). The impact of consumer security awareness on adopting the internet of things: A correlational study. Doctoral dissertation, Capella University.

Jovanovic, B. (2022). Internet of Things statistics for 2022 - Taking Things Apart. Retrieved from: https://dataprot.net/statistics/iot-statistics/.

Meneghello, F., Calore, M., Zucchetto, D., Polese, M., & Zanella, A. (2019). IoT: Internet of threats? A survey of practical security vulnerabilities in real IoT devices. IEEE Internet of Things Journal, 6(5), 8182-8201.

Programmable Web. (n.d.). Public APIS. https://www.programmableweb.com/category/all/apis?category=20248%2-C25691&deadpool=1.

Sequeiros, J. B., Chimuco, F. T., Samaila, M. G., Freire, M. M., & Inácio, P. R. (2020). Attack and system modeling applied to IoT, cloud, and mobile ecosystems: embedding security by design. ACM Computing Surveys (CSUR), 53(2), 1-32.

Trustwave. (2018). New Trustwave Report Shows Disparity Between IoT Adoption and Cybersecurity Readiness. Available: https://www.trustwave.com/en-us/company/newsroom/news/new-trustwave-report-shows-disparity-between-iot-adoption-and-cybersecurity-readiness/.

Wurm, J., Hoang, K., Arias, O., Sadeghi, A., & Jin, Y. (2016). Security analysis on consumer and industrial IoT devices. In: Proceedings of the 21st Asia and South Pacific Design Automation Conference (ASP-DAC), IEEE.

Zhang, Z. K., Cho, M. C. Y., Wang, C. W., Hsu, C. W., Chen, C. K., & Shieh, S. (2014, November). IoT security: ongoing challenges and research opportunities. In 2014 IEEE 7th international conference on service-oriented computing and applications (pp. 230-234). IEEE.