Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Security Analysis of TP-Link Tapo Family of Home Security Cameras

Diploma Thesis

2022
Bc. Jakub Šimo

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Security Analysis of TP-Link Tapo Family of Home Security Cameras

Diploma Thesis

Study Programme:	Computer Science
Field of Study:	Information Security
Department:	Faculty of Informatics
Supervisor:	RNDr. Richard Ostertág, PhD.

Bratislava, 2022
Bc. Jakub Šimo

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Jakub Šimo
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Security Analysis of TP-Link Tapo Family of Home Security Cameras
Bezpečnostná analýza domácich bezpečnostných kamier rodiny TP-Link Tapo

Anotácia: Keď sa výrobcovia snažia ušetriť pri vývoji svojich produktov, tak je to často na úkor ich bezpečnosti. Zneužitá inteligentná domáca kamera umožní útočníkovi prístup k jednému z najsúkromnejších miest v našom živote – našim domovom.

Preto cieľom tejto práce je analyzovať (z hľadiska IT bezpečnosti) konkrétnu skupinu domácich bezpečnostných kamier od známeho výrobcu. Prvým krokom bude popis vonkajšej aj vnútornej činnosti skúmaného zariadenia. Potom sa zameriame na firmvér, jeho aktualizačný mechanizmus a pokúsime sa zistiť, či existuje spôsob, ako upraviť systémové súbory a trvalo kompromitovať zariadenie.

Vedúci: RNDr. Richard Ostertág, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 27.04.2021

Dátum schválenia: 28.04.2021 prof. RNDr. Rastislav Kráľovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Abstrakt

Bezpečnostné domáce kamery sa stali jedným z najpopulárnejších zariadení smart home trendu. Vďaka pokrokom sa stali bezpečnostné kamery v domácom prostredí ľahšie použiteľnými a zároveň dostupnými zariadeniami. Pri vpúšťaní kamier do našich domovov je ale bezpečnosť týchto zariadení veľmi dôležitá. V našej práci sa pozrieme, či sú najpopulárnejšie modely kamier dobre chránené pred útočníkmi.

Kľúčové slová: kamera, bezpečnosť, domácnosť, smart

Abstract

Home security cameras have become one of the most popular devices of the smart home trend. Thanks to the advances in technology, they have become more usable, while becoming more affordable than ever before. However, when we let cameras into our homes, their security should be up to the task. In our thesis, we take a look at one of the most popular camera line-ups and see whether they are well secured against potential attackers.

Keywords: camera, security, home, smart

Contents

Introduction

In this chapter, we will try to paint a picture of the current state of affairs in the space we will be taking a look at. There is a myriad of factors that all contributed to the current situation we have found ourselves in at the time of writing. We will try to state the points in somehow chronological order, to the best of our ability.

Real life situation

Globalisation, at least before the COVID-19 pandemic, has been at its highest point. The vast majority of items were and still are, produced in China. This is even more true when it comes to electronics. Shenzhen is the world's hub for manufacturing, assembly, and other things to do with the creation of electronics. This is thanks to the economic policies introduced by the Chinese government in recent decades. Thanks to this heavy concentration and the labor force available there, electronics can be manufactured and shipped around the world for almost unbeatable prices. The Chinese government has been able to turn what once used to be a rather regular village into the world's manufacturing hub in less than half a century.

The cheap cost of concentrated just-in-time manufacturing helped give birth to a new era. With the price of electronics being so low, more and more people are now able to afford to buy new gadgets each year. What once used to cost multiple months of salaried income now costs but a fraction of the minimum monthly wage. In effect, this has brought something of smartphone ubiquity. Smartphones have become the major computing devices in households of developed countries, whilst hyper-accelerating the development of poorer countries. At the time of writing, some governments are even mandating processes that can only be done with a smartphone. Governmental bodies are traditionally one of the slowest to adopt new technologies, so it should be apparent that smartphones have reached critical mass. A new era, the Age of the Smartphone!

Thanks to smartphones, electronics could be made even cheaper. In the past, manufacturers needed to include various components in their devices, such as displays, buttons, speakers, etc. for the user to be able to interact with them. This adds cost and complexity, especially if you happen to be including moving parts, such as buttons or potentiometers. However, brilliant minds have been able to progressively get rid of

these problems. Long story short, virtually everything is moving towards capacitive controls. Manufacturers can save costs by not needing to include moving parts. This includes both the parts themselves and the related design and manufacturing costs. Manufacturers have been able to sell these cost-cutting measures as premium features, which has been fascinating to witness, as many times, the capacitive controls were much inferior to their physical counterparts. The current pinnacle of this optimisation process is a device with a small number of or no controls, exclusively controlled by a smartphone. Nowadays, an increasing number of electronics have a companion smartphone app with which you can control said device. No need to include any displays, buttons, or anything of the sort. In this manner, manufacturers have been able to reach price points unheard of even ten to twenty years back.

There are further cuts to be made. The software has been able to steadily replace parts of the hardware. A debouncing capacitor here, some analog logic there, and suddenly, you are left with an almost bare board with a single microcontroller and a few sensors. This is much cheaper to manufacture, even when we take into account the cost of developing the microcontroller software. In addition, you can differentiate your product stack with software. This saves further costs as you reduce the number of assembly lines. It is so efficient, that some car manufacturers sell their cars fully kitted out and selectively enable some features, such as heated seats. This is of course possible even without software - the most basic implementation being a dip switch or a zero Ohm resistor. However, the software version has a much higher barrier to entry and can be further strengthened through cryptographic methods.

Now that we have arrived at software, let us talk about the cost optimisations here. As with everything, the cost of software development depends on the importance and severity/cost of a failure. With consumer goods, there is usually no immediate risk, e.g. death of the user, so the stakes are not that high. In the past, without the Internet, software needed to be working out of the gate, as it was distributed through physical media. Patching the software afterward was many times simply not possible. Nowadays, companies can ship the devices in an unfinished state and continue developing the software whilst the hardware is being manufactured, shipped, and sold. However, developers are more often than not being rushed to meet an arbitrary deadline. This leads to prioritisation and the security of the product is usually the first thing on the cutting board.

The security of the devices we use is an afterthought. The tides are slowly turning, but we cannot run away from the fact that, due to mostly capitalistic reasons, the security of our devices ranges from poor to non-existent. Today's homes usually have two particular pieces of electronics - a modem/router/wifi combo box (colloquially referred to with a misnomer router) and smartphones. In both cases, they are kept woefully out of date. In an overwhelming majority of the cases, firmwares are never

updated by its users. This is simply due to the fact that users do not know that their router needs to be kept up to date. Manufacturers do not provide user-friendly ways of finding out about new firmwares or do not even provide the updates themselves. Manufacturers are currently trying to provide companion smartphone apps, but this is still in its infancy and user benefit is usually a side effect - more on this later. As for smartphone firmwares, the situation is a little better, at least when looking at more reputable manufacturers. They do tend to provide some form of updates for a year or two, but they quickly stop and the quality of the new software is usually lacking. It is such a norm, users have already learned that it is not worth it to update, as they risk breakage and possibly a new, unfamiliar interface. The need for better software support is becoming apparent as we are currently at a point where even five to seven-year-old phones are perfectly capable of executing their duties all those years later.

Unfortunately, software support and the survival of a company are going in the opposite directions. There are a few business models that have been tried out, but currently, we have ended up on a live-service wave. Prices of hardware have hit proverbial rock-bottom, the differentiating factor is becoming the ecosystem and companion offerings, such as the companion apps or related services. This is a double-edged sword, as while the idea is sound, more often than not, the companion offerings are lackluster and used mostly to subsidize the cost even further by collecting user data. This would be fine, but regular users are not aware of the fact. In addition, we have gotten to a point where devices can become literal e-waste if the user stops paying or the company goes under. It is one thing when your television stops working, it is another when your "smart" door suddenly cannot be opened. This finally leads us much closer to the topic of this thesis.

We have a new wave of home automation on our hands. There have been flashes of this in the past, such as home alarms, CCTV camera systems, automatic garage door openers, etc. Currently, the market has become flooded with various home automation appliances - the marketing term is "the smart home". Instead of a washing machine, which can be set to run at a certain time, it can run when you leave work so that it finishes when you arrive home. Forgot a window open? An app on your smartphone can let you know. The useful ideas and devices will stick around - especially when it comes to home appliances. However, home appliances have lifetimes of a different magnitude compared to other consumer electronics - a ten-year-old oven is perfectly functional and commonplace, but a ten-year-old smartphone is practically ancient and a security risk. This brings us to an impasse. Manufacturers should keep supporting their devices if they put a "smart" label on their products, but people are not educated enough to care and force them to do so.

In addition, privacy concerns are mounting up. A case study example is the Ring smart camera doorbell. It is a doorbell with a camera, you can access the camera

feed through your smartphone, from anywhere, thanks to the magic of the Internet. However, doorbells are naturally set up to face opposite from the home they are serving, in average case recording the sidewalk or neighbours on the other side of the street without their consent. Let us not get into additional problems, such as someone hacking the camera over the Internet, because it is not being patched, or the cameras being used by law enforcement without an informed consent of the camera owner. Especially the latter is legally possible due to impossible to understand E.U.L.A.s of manufacturers who gorge on the data gathered by such devices. As is the tradition, the laws have not caught up with the technology we possess today, so it is a bit of a wild west out there. We have just scratched the surface in regards to the privacy in a digital, always-connected world.

So finally, we have come to the devices which we will be taking a look at as a goal of this thesis. People want to secure their homes and cameras are one way to do it. They act as a deterrent and a way of identifying potential culprits. CCTV cameras have been the traditional way of doing this, but they are cumbersome, due to their analog nature. Modern cameras are digital and usually connect to the local computer network, hence their name, IP cameras. In fact, the cameras are computers themselves. IP cameras are a valuable target to hackers. They can be used for a multitude of nefarious goals, such as blackmail, stalking, or burglary. One would think that these devices, which are supposed to protect our homes, would be the ones that receive care both from their users and manufacturers alike. However, that is not the case and even these devices are left unattended and insecure, becoming parts of bot-nets.

Chapter 1

Security Cameras

The market is currently flooded with various consumer-level cameras, mostly targeted at home or small business use. We are interested in the home use segment as it is the more problematic one in regards to security. This is simply due to the fact that businesses are usually more risk-averse and they make an actual effort to secure their networks.

1.1 Manufacturer choice

Naturally, there exist camera lines available from various manufacturers, ranging from no-name ones from the Chinese grey market to reputable ones which you can buy in any electronics store. We have chosen the cameras in a rather simplistic way. We have taken a look at the best-selling security cameras in the biggest eshop in Slovakia (and Czechia), Alza.sk. At the time of our choice, five of the TOP10 best-selling cameras have been from only two manufacturers - Xiaomi and TP-Link. After a little research, the Xiaomi cameras seemed to have piqued the interest of mostly homebrew hackers already, but TP-Link offerings did not seem to incite many researchers or hackers. Therefore, we have chosen to take a look at the *TP-Link Tapo* line-up. This choice has proven to be a good one, as for a long stretch of almost two years, the TP-Link Tapo line has had all of its cameras stay in the TOP10, one model, in particular, being the top-selling product in the home section for the full period.

Let us then delve deeper into what kind of a manufacturer TP-Link is. When we try to search for the brand name, we come up with the manufacturer's official website - always a good sign. The title reads: "TP-Link: WiFi Networking Equipment for Home & Business" - not exactly what one would expect, but let us continue. Visiting the website, we can find the "About us" section at the bottom of the page. It is a simple page with the following text:

> Founded in 1996, TP-Link is a global provider of reliable networking devices

> and accessories, involved in all aspects of everyday life. With a proven heritage of stability, performance, and value, TP-Link has curated a portfolio of products that meet the networking needs of all individuals.
>
> Now, as the connected lifestyle continues to evolve, the company is expanding today to exceed the demands of tomorrow.

This is a rather nondescript on its own, but if we take a look at other information sources, it starts to make sense. TP-Link has been a network equipment manufacturer since its inception, think switches, routers, Wi-Fi access points, etc. Due to various circumstances, they have decided to pivot to being a "lifestyle"-oriented brand in September 2016. On September 30, 2019, this branch has been spun off as a separate daughter company called *Tapo*.

It is safe to say that TP-Link is a recognizable brand. They are not a household name, but they might as well be, as their cheaper offerings are being used by a majority of small-to-medium-sized European Internet service providers. In addition, if we take a sample from Alza.sk, after opening the "WiFi routers" TOP section, we are greeted by a page on which seventeen out of twenty-four products are made by TP-Link. If we switch over to the "Most sold" tab, this number decreases to fourteen, which is still a majority, rather impressive showing. This makes the pivot to a separate brand name for their home offerings a little confusing, as they could have kept using their renown. Nevertheless, even if a manufacturer is popular, it does not mean that they are keeping their devices safe.

We can gauge the seriousness of the manufacturer by taking a look at the support of their low-to-mid-end offerings. Naturally, the higher-end products *should* receive better support, so we are not that interested in them. Taking a look at their current cheapest available consumer router, TL-WR820N, coming in at just shy of 12 euros, it is apparent from the price that we should not be expecting much. To put it in contrast, you will get a single decent lunch for this price. There are currently two "versions" of this device. At least the way TP-Link does it, they have "versions" and "revisions". Revisions are usually a minor design or Bill of Materials (BOM) change, designated by a decimal point bump (v1.0 to v1.1), while versions are revisions, but more drastic, think changing the CPU/System-on-a-Chip (SoC) used in the device, designated by a major number increase (v1 to v2). Because of this, revisions traditionally share the firmware while versions require completely different firmware. These changes can be done for a multitude of reasons:

- the manufacturer might simply want to lower costs by optimising the design
- some parts might have become unavailable throughout the years, either due to the semiconductor fabs discontinuing some chips used or closing down altogether

- software support for an older chip might be getting too expensive compared to other offerings. This can be for example due to the chip manufacturer no longer providing support, which puts the onus on the router manufacturer

- the product requires a newer feature, which is not possible with the older platform. Naturally, the manufacturer could then create a product with a different name, but this is just a matter of syntax.

- and other probably business-related reasons

However, more often than not, older versions are left unsupported, in favour of their newer counterparts. It is enough to take a look at the latest update on the v1 vs v2 of TL-WR820N in question. The latest update for the v2 version has been released on 21st January 2022 - not stellar - but at least it is from the current year. The update has even added WPA3 support, a pleasant surprise. But, if we take a look at the v1, the story gets a little sadder. With an initial firmware release on 10th October 2018, the last, second update (so a third firmware), has been released on 17th of September 2019. At the time of writing, the device has not been formally discontinued yet, so there is at least the possibility that further fixes for major exploits will be released.

Nevertheless, it is more of a question of ethics, whether companies should be releasing devices they are not willing to support. There are currently no laws that would forbid them from releasing a new model, whilst abandoning the old one. TP-Link is one of the better players in this regard, but as we have shown, even they are not perfect. Let us then take a look at the Tapo line of products and see, whether the story there is different or more of the same.

1.2 Tapo line-up

As previously mentioned, the Tapo line has been launched on the 30th of September 2019, before being spun off as a separate company. At the time of writing, they offer three product categories - smart bulbs, smart plugs, and smart cameras. The term “smart” is just a marketing term, in reality, all of these products are just able to be connected to the user’s Wi-Fi network and controlled from a companion app. We are interested in the cameras.

There are three lines to choose from - C1xx, C2xx, and C3xx. The original lineup was C100, C200, and C310, but last year a C110 and C210 has been added to the roster and this year, they have been joined by a C320 model.

The cameras all seem to offer identical features, only adding things like the servo motor in C2xx and water-proofing and ethernet in the C3xx. To make this work

Figure 1.1: TP-Link Tapo C100

shorter, we will be taking a look at and describing only the C1xx line, highlighting any important differences if we encounter them.

1.3 Hardware

The full specifications can be found in the appendix. To summarize the main hardware features of the camera based on the specifications provided by the manufacturer:

- a 1920x1080p camera - no Frames Per Second stated, but we will find out later it is 15
- night vision - an array of IR LED lights
- a microphone
- a speaker - an interesting addition
- an SD card slot - supports up to 128GB cards
- Wi-Fi connectivity
- C2xx only - two-axis motorised movement
- C3xx only - ethernet, water resistance, “3MP” camera

Not too bad for around 20-25 euros. Let us take a look at what we are actually getting for our money.

1.3.1 Disassembly

The device has no obvious screws or access holes through which it can be opened. After a little bit of prying, the front black plastic part can be separated from the body, to which it is attached by four retention clips, each located in the center of each of the sides of the device's body. After the black front piece is removed, the whole assembly can be easily slid out of the plastic body. We are left with the motherboard.

Inspecting the board, it indeed looks to be designed in-house, by TP-Link themselves - they take pride in this, as many manufacturers use pre-made designs. Two nice surprises are present on the board:

- there are nicely labeled UART serial connection pads available
- there is an unused header labeled *ETH* - possibly Ethernet - unfortunately, there are no further labels to indicate its pinout. The connector used here is a 4-pin Molex PicoBlade Wire-to-Board.

Otherwise, there seems to be nothing of special note. The heart of the camera appears to be a Realtek RTS3903 SoC, paired together with a 64Mbit XMC XM25QH64A SPI NOR flash. The board or the camera sensor bears no markings to indicate its origins.

1.3.2 System-on-a-chip

Focusing on the SoC, searches for the RTS3903 model come up with practically no results. Realtek's website does not contain any information about this SoC, not even acknowledging its existence. The only results are research blogs discussing cameras, product pages of cameras, and Chinese suppliers selling the chip, the latter two only briefly mentioning the name. It is of course possible, that this SoC model and its corresponding documentation is only provided after signing a non-disclosure agreement. After further search, we were able to find some leads pertaining to the family of chips. We have been unfortunately unable to record the site itself before its disappearence. However, we have a picture of the website with useful pieces of information. The information was most probably supposed to be inaccessible, as after fully loading the website, its contents were replaced by a Chinese text indicating unauthorized access.

The website indicates the existence of RTS3901, RTS3902, RTS3903 and RTS390X chips with different variations. Curiously, searches for these variations did not come up with much in the way of products, but we did find a few "leaked" SDKs and datasheets with "confidential" watermarks. This corroborated our speculation about the non-disclosure agreements. The chip family is, according to the materials, designed specifically for the development of IP camera devices, containing a moderately fast single-core CPU paired with hardware video and audio encoders and networking capabilities. The

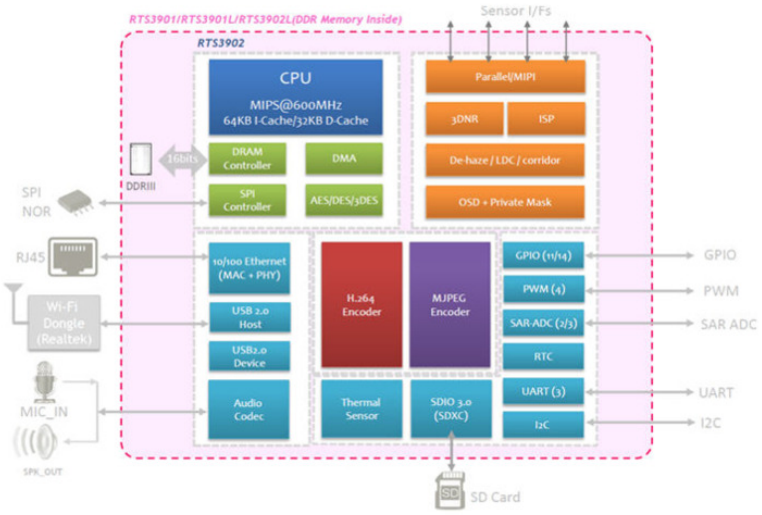

Figure 1.2: Realtek RTS390x family of SoC topology

CPU has a Lexra core. Lexra is a 32-bit variant implementation of the MIPS architecture, leaving out some instructions covered by patents. The architecture seems to be almost an industry secret at this point. According to a former Lexra engineer, "Many Lexra licensees do not want their use of Lexra to be known.". In our opinion, this cumbersome situation is probably a culmination of different factors. Realtek, the manufacturer of the SoC, presumably does not want to pay licensing costs for a MIPS or an ARM license. TP-Link and others are happy to be provided a tightly integrated chip with an SDK for this specific purpose.

1.3.3 Networking

The device contains a USB-connected RTL8188-based Wi-Fi chipset. After tracing the ETH header, it indeed seems to be connected to the correct pins on the SoC. After creating an adapter, we were able to get the device to connect to the network. The device appears to switch off the Wi-Fi when Ethernet is connected.

1.3.4 Rest of the hardware

There is not much else of note on the board - a generic microphone, IR LEDs, a speaker. C200 contains two servo motors and their respective drivers.

Now that we have become acquainted with the hardware, let us take a look at the provided software, the companion app.

1.4 Companion app

TP-Link provides the Tapo smartphone app. The logo of the app is a house, indicating that it is supposed to be more of a smart home ecosystem app than an app specifically for cameras. The app is not functional without a TP-Link Cloud account. After creating the account, you are let into the app, where you are able to set up your smart devices, after which you can access them from anywhere with an Internet connection. The interface for the cameras provides access to all of the features of the devices and from our experience work well and is simple to use. Here are the most notable functions available:

- transmitting audio to be played one-way through the camera
- two-way audio communication through the camera
- playback of recordings
- inverting the camera image
- turn on/off the status LED
- microSD card setup and management
- device sharing
- management of the account used to access RTSP/ONVIF streams
- other settings, mostly image correction related

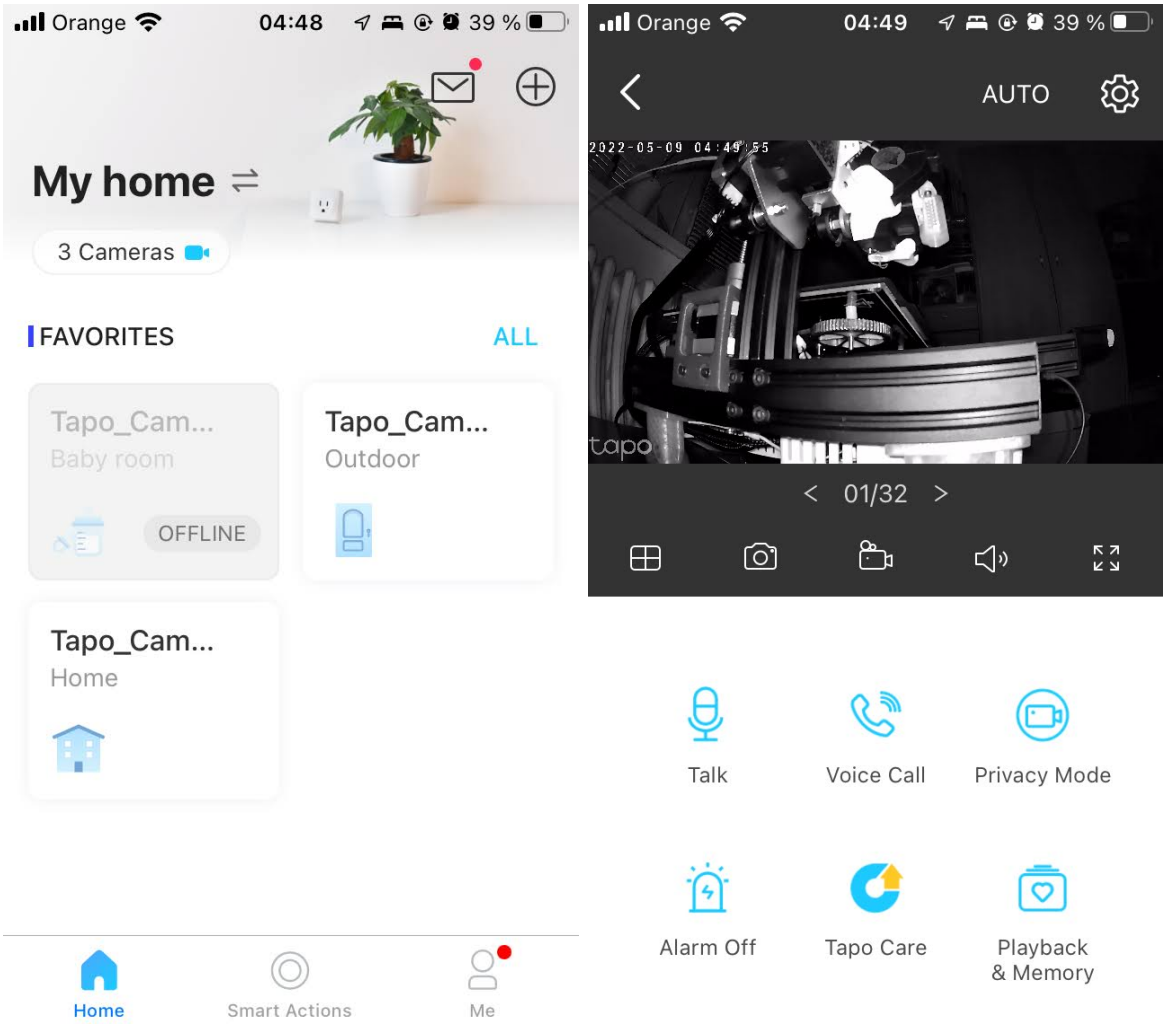

Figure 1.3: Tapo smartphone app

Chapter 2

Exploration

2.1 Past and current research

There have been only two research efforts about these cameras that we were able to initially find. A few months down the line, a little community has managed to spring up around these cameras. Sharing our different findings, we were all able to progress on various fronts.

2.1.1 Heartbleed and Pass-the-Hash attack

Probably the first published piece, Dale Pavey of NCCGroup published research about a few different cameras, one of them being our C200. He was able to identify that the device is vulnerable to the Heartbleed vulnerability, which allows the attacker to extract the working memory of the target. Through this, he was able to extract the MD5 hash, which is used for the authentication of the mobile app. This in turn allowed him to silently control the camera, with the original user being none-the-wiser. He has reported the vulnerability and the manufacturer has fixed it in a subsequent release.

2.1.2 pytapo project

pytapo is a project reverse-engineering the communication between the app and the camera, trying to provide a way to programmatically control the cameras. They were able to identify the protocol on which the communication is based. It has led to the creation of a plugin that integrates the Tapo cameras with the popular home automation Home Assistant platform.

2.1.3 Personal blog of Davide Depau

Another effort that documents the camera-app communication, this time by reverse-engineering the Java Android application itself. He also created a proof of concept

trying to decode the custom video stream used by the app, however, it only works somewhat, having low framerates and no audio compared with the app.

2.1.4 Personal blog of DrmnSamoLiu

General research around the cameras. He was able to capture and analyze parts of the firmware by analyzing the network traffic. After we shared our findings, he was able to identify a shell injection format string vulnerability in the ART partition configuration parsing. The execution is a little cumbersome, but it allowed him to enable a telnet server on the device. However, if the server crashes, the camera needs to be restarted to restore the functionality due to the nature of how it is started.

2.1.5 nervous-inhuman Github repository

The repository compiled a few pieces of information. The main discussion regarding the shell access happened here, Mr. Depau, DrmnSamoLiu, and we discussed our findings here.

2.2 Our work

As the mobile app has seen partial analysis from the pytapo project, we have decided to start tackling the camera from the hardware side.

Firmware

Traditionally, TP-Link provides firmware files for their products on the support page of the respective device. Unfortunately, this does not seem to be the case when it comes to the whole Tapo product line - product pages of neither cameras, smart plugs nor smart bulbs contain any firmware files. After a brief look at the update process in the smartphone app, it appears that the firmware can only be updated through there, so the manufacturer presumably did not consider it useful to provide the firmware files. This is unfortunate, as:

- as researchers, we are unable to easily access different firmware versions for analysis
- as users, we have no good way of fixing a device with a corrupted firmware
- as researchers, the previous point holds even more true as we are at a higher risk of bricking the device

With this route being halted, we try to move on to other possible data sources.

GPL source code

The camera came with a "GNU General Public License Notice". This meant that the software of the camera uses some open-source source code licensed with the GPL license, presumably the Linux kernel. TP-Link is one of the manufacturers who is pretty good about upholding their end of the bargain when using any open-source code. They generally provide the modified source code on the product support page, together with the firmware files. Unfortunately, as we have already mentioned, the product support page for our device contained neither the firmware nor the GPL source code. TP-Link does have a designated email address to which GPL source requests can be sent. After contacting a representative, they promptly made the sources available on the product page, where they are still currently available.

Initially, we were able to gather from the provided source code that the camera indeed uses Linux based system, more precisely a rather old OpenWRT release from 2012. The sources are incomplete, missing various configuration files and folders referenced in build scripts, but as we will see further down the line, there will be some useful pieces of information hidden here. In addition, TP-Link has provided the GPL sources for the C200 model. At first, we thought that there was an error, as the C200 sources were the same as the C100 sources. However, the TP-Link representative assured us that this was indeed correct. We were able to independently confirm this after gaining access to the firmware files for each of the models.

2.2.1 UART connection and shell access

Checking for port activity

During our hardware overview, we have identified what presumably are serial UART connection pads. After connecting a USB TTL adapter and trying out various baud rates, we were able to determine that the port is indeed active. After finding the correct baud rate, we were presented with a login prompt.

Login

Unfortunately, none of the username/password combinations that are common for TP-Link devices worked. Fortunately, after combing through the provided GPL source code, we were able to find a configuration file, which mentioned a default password set in the SoC SDK - it was found in the buildroot config file:

```
tapo-c200-gpl-code/camera_slp_realtek_c200/torchlight/product_config/
ALL/buildroot.config
Line␣450:␣CONFIG_SLP_LOGIN_PASSWORD="slprealtek"
```

Fortunately for us and unfortunately for the security of the device, this password has been left unchanged. We were able to gain root prompt access by logging in as the user root with password slprealtek. We have shared our finding with the community, who was stumped at the time - the flash storage of the device has been dumped, but people were unable to break the hashed password they found within it. From this point on, we had root access and could explore inside of the device.

2.2.2 Inside of the system

After gaining root access, we confirmed that indeed, the system is using OpenWRT as its base. This makes our work a little easier, as OpenWRT comes standard with tools such as mtd for managing the flash storage, and its well-liked uci configuration interface. After determining that those tools have not been left out by the manufacturer, we were able to make a backup of our flash for analysis. The files can be extracted either by storing them on the SD card or by using devices busybox which has been compiled with netcat enabled.

Dumping and analysing the configuration

Running uci export produces some 1600 lines of configuration. Most of it is a standard configuration of different OpenWRT utilities, however, TP-Link engineers have used this mechanism for storing all of their configuration too. Let us take a look at some interesting sections.

Wireless configuration

```
config wlan 'ap0'
        option broadcast_ssid 'on'
        option region 'CN'
        option band '2g'
        option channel '6'
        option hwmode 'bgn'
        option channel_width 'ht20'
        option security 'none'
        option encryption 'ccmp'
        option wps 'off'
        option auto_disable_time '0'
        option isolation 'off'
        option acl 'none'
        option ssid 'Tapo_Cam_603C'
```

```
        option on_boot 'off'

config wlan 'sta0'
        option on_boot 'on'
        option network_id '0'
        option rssi '0'
        option freq '0'
        option security 'psk-mixed'
        option encryption 'auto'
        option key 'ORALwCpfdh+xPk4sfou8gg=='
        option ssid 'doma'
        option bssid 'd8:47:32:50:95:2a'
        option connect_onboot 'on'
```

The device seems to be set to the Chinese region regardless of which region it is operating in, without the ability to change it from the app. This might be problematic in certain regions of the world, but it should not compromise security in any way.

The access point used for the device setup has an SSID in the form of `Tapo_Cam_XXXX` where `XXXX` are the last four digits of the devices MAC address. This is helpful when setting up multiple devices. The communication is encrypted, however, the access point is open, meaning there is a timeframe in which the device could be compromised during setup. This issue can be alleviated by using a random default password set at the factory. This practice has already been forced onto the manufacturers by the European Union, as it has been standard practice to use traditional `admin`-`admin` username and password, which has been deemed unsafe. Interestingly, this requirement does not seem to apply to these devices. WPS functionality is also explicitly turned off, which has been the recommendation for a long time now.

The key parameter is interesting, as, on standard OpenWRT devices, it is stored in plain text. We have not been able to determine what is the exact encoding and lifepath of the key that we are seeing in the configuration. We believe that it would be just a question of time and we invested our time elsewhere, as if someone has gotten their hands on this configuration, there was already something that has gone wrong elsewhere.

Video feed credentials

```
package user_management

config root 'root'
        option username 'admin'
```

```
        option passwd '9BF1EF469286D8B1907F0E48F02136E4'
        option ciphertext '<long string>'

config third_account 'third_account'
        option username '---'
        option passwd '---'
        option ciphertext '<long string>'
config authentication 'authentication'
        option basic_enabled '0'
```

This section holds the credentials which are used for authenticating the user with the camera. The admin account seems to always be populated, with the password being the traditional admin phrase if the camera has not been set up yet. However, the credentials cannot be used to access the stream. Our second assumption was that the default credentials could be used by the app to authenticate itself during the setup, but we were not able to confirm this nor make it work. Before the camera is set up, they are not accepted, after the camera is set up, the admin password is changed, presumably to something derived from the user's password. We presume that this is something that is stored internally in the app and used for seamless authentication. The third_account is the account that can be set up under Advanced Settings - Camera Account in the app. These credentials are then used to access the standard RTSP or ONVIF streams directly from the camera. We do not see an obvious problem with this setup apart from the use of MD5 hashes instead of something stronger. We were able to determine that it is an MD5 hash from the length, from the fact that the hash of the phrase admin is used as a stand-in before setup, and as we will see later, from the names of functions in the decompiled binaries.

upnpc configuration

```
package upnpc

config on_off 'upnpc_info'
        option enabled 'off'
        option mode 'manual'

config entry 'uhttpd'
        option proto 'TCP'
        option ext_port '80'
        option desc 'uhttpd'
```

```
config entry 'rtsp'
        option proto 'TCP'
        option ext_port '554'
        option desc 'rtsp'

config entry 'onvif_service'
        option proto 'TCP'
        option ext_port '2020'
        option desc 'onvif_service'

config entry 'vhttpd'
        option proto 'TCP'
        option ext_port '8080'
        option desc 'vhttpd'
```

From this part of the configuration file, we can see which ports are being exposed through UPnP. Without commenting on the issues surrounding UPnP, the router on the user's network is more important in regards to the security of this particular protocol. The fact that the camera uses UPnP is not problematic in itself - the user just needs to take more care and device whether it is worth the risk to have UPnP enabled in their network.

HTTP server configuration

```
    config uhttpd 'main'
        option listen_https '443'
        option home '/www'
        option rfc1918_filter '1'
        option max_requests '6'
        option cert '/tmp/uhttpd.crt'
        option key '/tmp/uhttpd.key'
        option cgi_prefix '/cgi-bin'
        option lua_prefix '/luci'
        option lua_handler '/usr/lib/lua/luci/sgi/uhttpd.lua'
        option script_timeout '180'
        option network_timeout '180'
        option tcp_keepalive '0'
```

The camera uses the standard OpenWRT uhttpd server, the config looks pretty standard too. One interesting part is the key setting, as `/tmp` is mounted as a tmpfs on this device. After looking around how this file is created, we have been able to find

that is it generated at each boot. However, the server is not set up to use ephemeral keys derived from these keys for each connection - this can be seen from the network capture of the communication with the smartphone app. All of the communication that is happening is being encrypted only by these keys directly, the non-ephemeral version of the Diffie-Helman key exchange. This is not recommended nowadays, as there is no forward secrecy. The fact that the key is re-generated on each boot is a plus, but since the device is a security camera, its uptime will be long-lived, negating the usefulness of this step.

tp_manage

```
package tp_manage

config tp_manage 'factory_mode'
        option enabled '0'

config tp_manage 'bind_info'
        option owner '93185CC75BC0C0872E3C744B9F92D41B'
```

We are not hundred percent sure how to work with these settings, but the first one seems to indicate whether the device has been set up and the second one is presumably some hash or id for the cloud account. There is also a binary of the same name.

cloud - firmware update

```
config cloud_reply 'upgrade_info'
option type '1'
option version '1.0.17 Build 201112 Rel.29622n'
option release_date '2020-12-16'
option download_url 'http://download.tplinkcloud.com/firmware
    /Tapo_C100v1_en_1.0.17_Build_201112_Rel.29622n_
    _1608109639905.bin'
option release_log 'Modifications and Bug Fixes: \n1. Fixed
    the bug that the auto-reboot feature does not take effect
    on certain dates .\n2. Optimized the SD card detection
    mechanism.'
option release_log_url 'undefined yet'
option location '0'
```

This is a section that is normally unpopulated. After being prompted by the app, the camera checks for the newest available update and stores it in this config. Afterward,

if the camera is told to start the update procedure, the update stored in the config is applied. The firmware is downloaded through http, but we will find out that it is signed, so it is not a problem. However, we can now download at least the newest firmware update for analysis, as we have the URL to it now. Unfortunately, there appear to be random numbers, so we cannot access any older firmwares.

Rest of the config

The rest of the config does not appear to contain anything we identified as relevant. There are different settings for the cloud connection with TP-Links servers, camera-related settings, and standard OpenWRT settings. The full dump of the config can be found in the appendix.

2.2.3 Analyzing the firmware

After making a backup of the camera's flash memory and downloading a few firmware updates over time, we have been ready to start analyzing and comparing the firmware files.

Identifying the sections

Using the standard binwalk tool, we took a look at one of the firmwares:

```
> binwalk Tapo_C100v1_en_1.0.10_Build_200519_Rel.66820
   n_1594610788996.bin
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------

25088         0x6200          LZMA compressed data, ...
66560         0x10400         LZMA compressed data, ...
1531392       0x175E00        Squashfs filesystem, ...
8062720       0x7B0700        gzip compressed data, ...
8127408       0x7C03B0        gzip compressed data, ...
```

This result is a little surprising, as usually there are more sections when analyzing a regular router firmware, which this should be. However, we have realised that binwalk does not support Lexra architecture, so it could not give us any other useful pieces of information. Thankfully, this output has indeed been correct and helpful in identifying the different parts of the firmware. The first and second "LZMA compressed data" are both bootloaders, "Squashfs filesystem" is the root filesystem, "gzip compressed data" are both configs, the first config is the fixed factory one while the second one is the user

config, in a regular overlayFS form used in stock OpenWRT. Since we already have access to the console, we can try checking the partitions with mtd:

```
cat /proc/mtd

dev:    size   erasesize  name
mtd0: 0001d800 00010000 "factory_boot"
mtd1: 00002800 00010000 "factory_info"
mtd2: 00020000 00010000 "art"
mtd3: 00010000 00010000 "config"
mtd4: 00010000 00010000 "boot"
mtd5: 00165c00 00010000 "kernel"
mtd6: 0054a400 00010000 "rootfs"
mtd7: 000f0000 00010000 "rootfs_data"
mtd8: 007a0000 00010000 "firmware"
```

and taking a look at the bootlog:

```
[    0.405000] 0x000000000000-0x00000001d800 : "factory_boot"
[    0.434000] 0x00000001d800-0x000000020000 : "factory_info"
[    0.464000] 0x000000020000-0x000000040000 : "art"
[    0.474000] 0x000000040000-0x000000050000 : "config"
[    0.485000] 0x000000050000-0x000000060000 : "boot"
[    0.495000] 0x000000060000-0x0000001c6000 : "kernel"
[    0.521000] 0x0000001c6000-0x0000006f0000 : "rootfs"
[    0.549000] 0x0000006f0000-0x000000800000 : "rootfs_data"
[    0.561000] 0x000000060000-0x000000800000 : "firmware"
```

This is indeed a standard OpenWRT layout, so we will not be going into it. The only interesting thing is the fact that there are two bootloaders, which is a little unusual, and the system is not aware of one of the bootloaders.

We were able to identify header offsets and confirm the previously identified offsets from these three files:

```
# End of the file contains partition sizes for flash
torchlight/product_config/ALL/buildroot.config
# TP_BOOT_MAGIC bytes in
factory_boot/rts3903_src/include/configs/rlxboard.h
# config constants
factory_boot/rts3903_src/bsp/RTS3903/rlxboard.h
```

The important bits turned out to be the fact the firmware info headers are 0x200 bytes in size. This meant that the firmware starts with 0x200 byte header, which is followed

by a 0x10000 bytes of a bootloader, then at the address 0x10200 (offset is because of the first header) is the second bootloader, after which at 0x176200 is the root filesystem.

Looking at the header, it is reminiscent of the traditional TP-Link header used in their routers (only part is shown as the rest are zeroes):

```
> xxd -c16 \
Tapo_C100v1_en_1.0.10_Build_200519_Rel.66820n_1594610788996.bin \
| head -n18
00000000: 0000 0100 55aa 4c5e 831f 534b a1f8 f7c9  ....U.L^..SK....
00000010: 18df 8fbf 7da1 aa55 0200 0000 0000 001f  ....}..U........
00000020: 1be5 a4a8 1b9f b7f3 0a1b e84c a735 ff67  ...........L.5.g
00000030: 0cc9 919f 9238 a84d c5d4 e28d 9277 45fc  .....8.M.....wE.
00000040: 0f6f a075 28cf 91ac 99fd 1b24 cf29 2fc7  .o.u(......$.)/.
00000050: 5b64 8928 7ba0 bcee 4f69 a044 40e4 700a  [d.({...Oi.D@.p.
00000060: 8029 8532 3b11 d281 3f55 a7f6 8d97 0003  .).2;...?U......
00000070: e2e6 61a6 bba3 1515 09fe e99c 2023 ad36  ..a.........  #.6
00000080: 33ba ea68 282e 8ded 2dbc 00e1 47e5 e29c  3..h(... -...G...
00000090: 9116 f24e 0185 3609 77ac eca3 e5a3 43aa  ...N..6.w.....C.
000000a0: 0001 0003 0002 c000 0000 0000 0000 0000  ................
000000b0: 0000 16f3 4fc2 8e5c 7cd4 2249 8e7e f0c4  ....O..\|."I.~..
000000c0: a69f 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 0000 4857 4445 5343 0000 0000 0000 0000  ..HWDESC........
000000e0: 0001 edcf 1d37 890c 9d55 4b59 4b8f 5679  .....7...UKYK.Vy
000000f0: 46d2 3bc1 ad10 721c 8926 36ab 9172 3c45  F.;...r..&6..r<E
00000100: f276 2667 1554 45da 1a81 806c 0c37 0ed4  .v&g.TE....l.7..
00000110: 8278 0000 0000 0000 0000 0000 0000 0000  .x..............
```

And one from a newer firmware:

```
> xxd -c16 \
Tapo_C100v1_en_1.0.16_Build_200929_Rel.65405n_1604655483096.bin \
| head -n18
00000000: 0000 0100 55aa 4c5e 831f 534b a1f8 f7c9  ....U.L^..SK....
00000010: 18df 8fbf 7da1 aa55 0200 0000 0000 001f  ....}..U........
00000020: 8f64 a691 f947 0e02 9618 d005 97e5 d4ba  .d...G..........
00000030: 02bd fc36 fd47 be10 2f1d 48d1 7e88 ce0c  ...6.G../.H.~...
00000040: d650 b40f ee74 de0b 1b73 ff2f af8a 2e1f  .P...t...s./....
00000050: a3b0 dc13 50f9 e278 e073 663c ba18 d663  ....P..x.sf<...c
00000060: d8df 8285 03d3 5924 7e27 9e72 a5cf a827  ......Y$~'.r...'
00000070: 6103 768b ae8e 35e1 04b0 cf06 f7cd a3bc  a.v...5.........
00000080: dfca 7c70 58a1 0a33 1bf5 489e 1e08 0114  ..|pX..3..H.....
00000090: 7c4d 8314 b09a f6be f3ac 7c0c f39d fa73  |M........|....s
000000a0: 0001 0003 0002 c000 0000 0000 0000 0000  ................
```

```
000000b0: 0000 16f3 4fc2 8e5c 7cd4 2249 8e7e f0c4  ....O..\|."I.~..
000000c0: a69f 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 0000 4857 4445 5343 0000 0000 0000 0000  ..HWDESC........
000000e0: 0001 edcf 1d37 890c 9d55 4b59 4b8f 5679  .....7...UKYK.Vy
000000f0: 46d2 3bc1 ad10 721c 8926 36ab 9172 3c45  F.;...r..&6..r<E
00000100: f276 2667 1554 45da 1a81 806c 0c37 0ed4  .v&g.TE....l.7..
00000110: 8278 0000 0000 0000 0000 0000 0000 0000  .x..............
```

We can see that some parts are the same and some change. The changed ones have a correct length to be MD5 hashes, but we cannot confirm that at this point without some further information. Let us continue with the filesystem.

Root filesystem

At first, we were not able to find out what version of the SquashFS filesystem is being used here. The GPL sources did not contain any script or configuration which would indicate how the filesystem was put together and we could not get it to unpack either. Fortunately, we were able to unpack it with squashfs-tools from the archlinux repository after a few hours of trying to do it with different libraries. While it might seem stupid after reading this, it was rather non-trivial to figure out. Other versions of the standard tools did not work and most of the information we were able to find was pointing to the fact that manufacturers do non-standard things in regards to SquashFS implementations. After some trial and error, the command and settings to repack the filesystem turned out to be:

```
mksquashfs new-squashfs-root new.sqsh -comp xz -b 256K \
-no-xattrs -all-root
```

Config partitions

The configuration partitions are standard OpenWRT setup, nothing appears to have been changed.

2.2.4 Analyzing programs

After unpacking the root filesystem, we were able to start analyzing the available programs. We have decided to use the open-source reverse-engineering tool Ghidra for exploring and decompiling the binaries. There were a few candidates for exploring, namely `tp_manage`, `uhttpd` and `slpupgrade`. In addition, there were a few scripts that did not belong to a standard OpenWRT install too.

Architecture

The unusual architecture of the system turned out to be our first problem when trying to analyze the available programs. Lexra does not seem to be supported by any modern software and as previously mentioned, more of an industry secret. While not perfect, since the architecture is still MIPS, just without a few instructions and some quirks, we were able to use the MIPS profile in Ghidra and go through the code. Since we were missing the standard libraries for this architecture many functions were not identifiable. Thankfully, there were some parts that we were able to make some sense of.

tp_manage

We started with `tp_manage`, however, this has proven to be a dead-end. It is a program that takes care of the communication with TP-Link's servers and there did not appear to be anything out of the ordinary.

uhttpd

We had uhttpd source code available since it is part of OpenWRT, but it has proven to be practically useless. TP-Link used a heavily modified version of this program. The server indeed handles all of the communication with the app, as we have suspected. It also handles the video streaming into the app, which is separate from the RTSP server on the camera. Unfortunately, we were unable to easily decode the video or audio stream. The community has also failed to do so, spawning only the previously mentioned proof of concept which can get a few frames in before falling apart. However, the server is running as root and there are quite a few exec calls that appear to indiscriminately insert parameters into format strings. We suspect that these are exploitable. Looking further, the server is also the one responsible for starting the firmware update procedure, leading us into the next program.

slpupgrade

The `slpupgrade` binary is directly executed by the uhttpd server. It accepts a single file path as a parameter. It takes the file at this path and it does approximately the following:

1. Check for the existence of the file
2. mmap the file
3. Validate the RSA signature of the firmware
4. Validate the checksums of the file

5. Validate the information such as the device type, revision, etc.

6. If everything was ok, write the firmware into the flash memory.

We are interested in the signature and checksum validation. The decompilation confirms that the changed values in the headers were indeed an RSA signature and MD5 checksums. We were able to gather the correct offsets and identify what some parts of the header mean. We would like to try and alter the firmware. We could do it by hand and write it into the flash, but we would like to use the slpupgrade utility to make it easier.

Since we now know the offsets, we can recalculate the checksums after changing parts of the firmware. The only problem that is left is the signature. Curiously, the program copies the signature into a string from the firmware file for validation. After it copies it, it then overwrites the spot at the header with zeroes. This is because the checksum is calculated with the header included and since the signature cannot be known beforehand, zeroes are put as a placeholder. However, the program then does not come back to write the signature back, so the signature is lost and the firmware is written into the flash memory with zeroes instead of the signature. This fact made it easy to modify the program to ignore the signature, as the signature check happens as the first thing, so the jump into the validation procedure can be easily replaced with a NOP instruction.

After playing around with this, our first attempt at modifying the firmware failed with a bricked camera - as we later found out, we have not calculated the checksums correctly. This led to a finding that the bootloader validates the checksums during the boot process. However, as mentioned, it does not check the signature, as it cannot because it is not there. After analyzing the bootloader, we found that it contains a whole recovery HTTP server, accessible through the hidden ethernet interface, which validates the signatures of submitted files. We have been able to recover our camera through this functionality.

Telnet

The original firmware does not contain a busybox distribution with a telnet client compiled into it. However, all of the subsequent firmwares appear to do. We have realised this only after a few people in the community have pointed out that they were trying to get it to work. We have no idea why did they decide to include this hidden functionality when it was not there in the original release.

The telnet server runs by default but is bound only to the localhost interface, so we cannot access it from the outside. With our acquired know-how, we were able to modify the firmware and change the service file so that telnet would accept all connections.

Other scripts

We were able to identify other custom scripts created by TP-Link's developers. Only one of them has proven particularly interesting - check_upgrade init service file.

```
> cat /etc/init.d/check_upgrade
#!/bin/sh /etc/rc.common

START=13

start() {
        if [ ! -d "/tmp/sdcard" ]; then
                mkdir -p /tmp/sdcard/
        fi

        if [ -b /dev/mmcblk0p1 ]; then
                mount -t vfat /dev/mmcblk0p1 /tmp/sdcard/
        else
                exit 0
        fi

        echo "check firmware upgrade"

        if [ -e /tmp/sdcard/factory_up_boot.bin ]
        then
                echo "start firmware upgrade ..."
                slpupgrade -n "/tmp/sdcard/factory_up_boot.bin"

                while true
                do
                        sleep 10
                done
        else
                umount /tmp/sdcard
        fi
}
```

As we can see, it is a simple script. When the device starts up, it checks whether the SD card partition is available. If yes, it checks if there is a particularly named file. Upon finding this file, it starts the upgrade process with this file. This is a completely undocumented functionality, on its own, not that useful. It cannot be used to recover from a brick, it could be used to update the device offline, but once again, undocumented. However, due to the way the slpugprade works, we are able to

downgrade firmwares with this. slupgrade does not check whether the firmware version is higher, it just checks the signature, checksums, and various other things such as camera type, revision, etc. but not the firmware version. This has proven quite useful, as we were able to downgrade to older firmwares to explore them while in action.

Chapter 3

Security evaluation

In this chapter, we evaluate our findings, and if any problems were found, we propose ways in which they could be resolved. We will evaluate it from two viewpoints, from the viewpoint of a remote attacker and from the viewpoint of an attacker with access to the device at various points in time.

3.1 Remote attacker

We must assume that the camera will be connected directly to the Internet - there have been many cases of even home cameras with default credentials being directly accessible. From our exploration, it appears that there is some viable attack surface. The HTTP server, which appears to be a heavily modified version by the manufacturer, employs direct exec calls with unsanitized formatting strings (calling other processes through ubus or directly setting things into the configuration). After we shared our findings with the community, there has already been one RCE exploit found exploiting this error. The RTSP and ONVIF are other possible points of ingress, but they appear to have a lower attack surface. While they are a little more complicated programs, dealing with audio-video streaming, their authentication appears to be set up correctly. When the "third account" is not set up, the servers seem to be well secured, so the only point of attack could be some buffer used during the authentication, but that does not seem to be the case. The only other point is the telnet server. Since it is not used, we do not see any reason why it should be running but is bound to localhost and hence it does not respond to anything. However, everything is running as root, so any single exploit can take over the whole device without the user noticing.

3.2 Attacker with physical access

An attacker which is in the possession of the camera has a few options. He can downgrade the device with an SD card to a vulnerable firmware - this is usable as there already exists an RCE exploit which has been fixed just recently (February 2022). Otherwise, opening the device without any signs of tampering is trivial, even without turning the device off. After popping the cover off, the serial UART pads are directly accessible, on all of the models. This is not bad, on the contrary, we would like to praise the physical design, as everything is easily accessible. However, since the console is enabled, it allows the attacker to compromise the device if left alone for a few minutes.

3.3 Severity and Recommendations

Let us list and try to assign vulnerability scores based on the Common Vulnerability Scoring System (CVSS), version 3.1:

Attacker modifying the firmware in a supply chain

Base Metrics

Attack Vector - Physical - the attacker needs to be in physical possession of the device

Attack Complexity - Low - There are no special conditions

Privileges Required - None - the attacker just has to authenticate with the default known password

User Interaction - None

Scope - Unchanged - the attacker cannot influence manufacturer's servers and already runs as root

Impact Metrics

Confidentiality - Complete - the attacker has full control

Integrity - Complete - the attacker has full control

Availability - Complete - the attacker has full control

This attack comes out to a 6.8 score. We are in the process of submitting it. Recommendations for alleviating the issue are validating the signature during the boot process. However, the manufacturer should also provide a firmware that disables the signature check, both during boot and during an update, for hobbyists and researchers.

Attacker downgrading the firmware

Base Metrics

Attack Vector - Physical - the attacker needs to be in physical possession of the device

Attack Complexity - Low - there are no special conditions

Privileges Required - None - just insert the SD card

User Interaction - None

Scope - Changed - the attacker can exploit a vulnerability in the older firmware

Impact Metrics

Confidentiality - Complete - the attacker has full control

Integrity - Complete - the attacker has full control

Availability - Complete - the attacker has full control

This comes out to a score of 7.6.

This can be solved by checking if the version of the firmware is higher in the update program.

Remote attacker

An RCE vulnerability CVE-2021-4045 regarding the C200 model has been evaluated as having a score of 9.8.

The severity of this could be lowered by not running all of the processes as the root user and having better sanitisation practices when using formatting strings for exec commands.

Bibliography

[1] Discussion regarding the camera on github. https://github.com/nervous-inhuman/tplink-tapo-c200-re/issues/1. Accessed: 2022-05-09.

[2] drmnsamoliu's blog. https://drmnsamoliu.github.io/telnet.html. Accessed: 2022-05-09.

[3] Firmware layout of tp-link firmware - firmware-mod-kit. https://gitlab.com/kalilinux/packages/firmware-mod-kit/-/blob/a98105cfc1ab98ebd157f52b1458b129e2bcad45/src/tpl-tool/doc/Image_layout. Accessed: 2022-05-09.

[4] Firmware tool for tp-link firmwares with the version 3 header. https://github.com/xdarklight/mktplinkfw3. Accessed: 2022-05-09.

[5] Lexra. https://www.linux-mips.org/wiki/Lexra. Accessed: 2022-05-09.

[6] Lexra story. https://www.probell.com/lexra. Accessed: 2022-05-09.

[7] Lights, camera, hacked! an insight into the world of popular ip cameras. https://research.nccgroup.com/2020/07/31/lights-camera-hacked-an-insight-into-the-world-of-popular-ip-cameras/. Accessed: 2022-05-09.

[8] Tapo brand announcement. https://www.tp-link.com/en/press/news/17086/. Accessed: 2022-05-09.

[9] Tapo c200 rce. https://www.hacefresko.com/posts/tp-link-tapo-c200-unauthenticated-rce. Accessed: 2022-05-09.