Finding (and Exploiting) Vulnerabilities on IP Cameras: The Tenda CP3 Case Study

Dario Stabili[1](✉), Tobia Bocchi[2], Filip Valgimigli[2], and Mirco Marchetti[2]

[1] Department of Computer Science and Engineering, Alma Mater Studiorum - University of Bologna, Bologna, Italy
dario.stabili@unibo.it

[2] Department of Engineering "Enzo Ferrari", University of Modena and Reggio Emilia, Modena, Italy
{tobia.bocchi,filip.valgimigli,mirco.marchetti}@unimore.it

Abstract. Consumer IP cameras are now the most widely adopted solution for remote monitoring in various contexts, such as private homes or small offices. While the security of these devices has been scrutinized, most approaches are limited to relatively shallow network-based analyses. In this paper, we discuss a methodology for the security analysis and identification of remotely exploitable vulnerabilities in IP cameras, which includes static and dynamic analyses of executables extracted from IP camera firmware. Compared to existing methodologies, our approach leverages the context of the target device to focus on the identification of malicious invocation sequences that could lead to exploitable vulnerabilities. We demonstrate the application of our methodology by using the Tenda CP3 IP camera as a case study. We identified five novel CVEs, with CVSS scores ranging from 7.5 to 9.8. To partially automate our analysis, we also developed a custom tool based on *Ghidra* and *rhabdomancer*.

1 Introduction

This paper discusses the application of a classical methodology for conducting in-depth security analysis with a focus on consumer IP cameras. IP cameras are popular IoT devices, and their cybersecurity has been scrutinized by the scientific community in recent years. Related works [11,12] have already identified several vulnerabilities, but their analysis has mainly focused on network traffic. The discussed methodology takes an in-depth approach and requires physical disassembly of the device. While this attacker model may seem more powerful than the one used in related literature, we argue that it is more realistic and allows for the identification of a higher number of more relevant software vulnerabilities. Consumer IP cameras are inexpensive, making it easy for attackers to purchase one or more devices to experiment on. Moreover, our results demonstrate that this kind of analysis enables attackers to extract sensitive information

K. Minematsu and M. Mimura (Eds.): IWSEC 2024, LNCS 14977, pp. 195–210, 2024.
https://doi.org/10.1007/978-981-97-7737-2_11

and devise attack strategies that can be weaponized against other IP cameras of the same make and model, even without physical access. The adopted methodology comprises five main steps.

The first step involves gathering information from open and public sources related to the IP camera under analysis. This step does not require the camera itself and can be performed by checking public data records from the *FCC ID Search* web service [7]. This information proves to be extremely useful, as it often includes high-resolution pictures of the printed circuit boards, which can be used to identify diagnostic and programming interfaces. Additionally, valuable information can be obtained from firmware repositories. Depending on the camera manufacturer and model, it is often possible to download a copy of the firmware from official or unofficial repositories. This step may enable an attacker to focus on a limited number of IP cameras that are more likely to exhibit interesting vulnerabilities.

The second step requires physical access to a specimen of the IP camera under analysis. Within this step, the camera is disassembled to gain physical access to the inner printed circuit board. The primary objective is to visually inspect it and identify one or more vulnerability surfaces that enable direct and low-level interaction with the camera. In our experience, it is highly likely to identify internal USB ports that have no external connectors, as well as simpler debugging and diagnostic interfaces based on standard JTAG or UART protocols. Having physical access allows us to connect and probe these interfaces to confirm that they are active and explore the related attack surfaces. At this step, it is usually possible to interact with the bootloader and potentially gain access to a command-line interpreter as a privileged user.

The third step begins by extracting the firmware deployed on the IP camera. This task can be accomplished either by exploiting low-level read access to the memory of the IP camera through a diagnostic interface or by physically connecting an external reader to the memory chip soldered to the PCB. In some cases, a chip-off might be necessary, although it has not been required in our experience. The extracted firmware is then subjected to common static analysis procedures aimed at identifying relevant partitions, configuration files, scripts, executable files, and cryptographic material.

The fourth step complements the static analysis of the firmware with a dynamic analysis of the network behavior of the IP camera under test. The peculiarity of our approach is that, instead of applying a general approach for static analysis we focus on the main services exposed on the network by the device, thus considering its use-case scenario. By considering the classical usage scenario of the target device, we demonstrate how existing classical tools for static analysis can be tweaked to quickly identify vulnerabilities exposed by the target device, thus preventing the security researcher to manually analyze different potential sources of vulnerabilities that wouldn't lead to exploits.

The fifth and final step builds upon the information gathered in the previous steps to perform a detailed reverse-engineering process of all executables that implement services available from the network. To partially automate this complex step, we developed a novel tool based on *Ghidra* [14] and *rhabdomancer* [13] that identifies the functions responsible for handling data received from network connections, builds the invocation sequence list and correctly identifies the thread responsible for each invocation sequence. This allows security researchers to quickly analyze the whole function call sequence of a target network handler to identify potential vulnerabilities in its code, thus helping them to demonstrate potential exploits and identify possible mitigation.

In this paper, we demonstrate the effectiveness of the proposed methodology by focusing on a popular consumer IP camera, the Tenda CP3, as a use case. Our analysis has already led to the publication of five new CVEs, two of which have a CVSS score of 7.5 and the remaining three have a CVSS score of 9.8.

1.1 Related Work

The scientific literature already contains several research papers focusing on the cybersecurity of IoT devices, IP cameras, and video surveillance systems. However, most of this work limits itself to network-based security analysis, either by sniffing and analyzing network traffic or by interacting with exposed network services. Notable contributions in this field include the analysis of the attack surface of IP-based surveillance systems [10], as well as wide-ranging analysis of IoT devices [12] and the exposed network services of IP cameras belonging to a given nation [3]. These papers demonstrate that connected IoT devices present many vulnerabilities; however, in many cases, the analysis is limited to relatively simple scanning and probing activities.

Other papers more related to our proposal focus on the in-depth analysis of network communications of a given IP camera [1,4]. In particular, the authors of [4] managed to exploit network-based attacks, such as Man-in-the-Middle, to eavesdrop on and interact with network communications between the IP camera (a TP-Link Tapo C200) and other devices on the same local network, thus identifying three novel vulnerabilities. We remark that these papers only based their analysis on network interactions, without identifying relevant vulnerabilities (such as remote code execution) that require an in-depth reversing of executables extracted from the firmware of IP cameras.

In relevant related work, Shwartz et al. [16] perform firmware extraction from 16 different IoT devices, with the final goal of extracting and cracking passwords that would allow remote access to the compromised devices. They demonstrate the effectiveness of their approach, as well as the widespread vulnerability related to password management of IoT devices, by creating a modified version of the Mirai botnet that utilized these passwords to compromise vulnerable devices. The authors discuss the possibility of conducting a more comprehensive static analysis and reverse engineering of the analyzed devices but do not undertake these tasks themselves.

On the other hand, this paper introduces a methodology for the security analysis of consumer IP cameras, demonstrated through the examination of the Tenda CP3 as a case study, and the development of a novel tool. To our knowledge, this is the first paper to delve into such an in-depth analysis of IP cameras.

1.2 Outline

The remainder of this paper is organized as follows. Section 2 presents the IP camera that we use to demonstrate the application of our analysis methodology, while Sect. 3 presents the detailed analysis of the firmware extracted from our device. Section 4 analyses and describes the two main programs responsible to handle all external connections to the camera, demonstrating a practical method to identify potential vulnerability and to design their related exploit. Finally, Sect. 5 highlights the main strengths of this work and describes future development.

2 Analysis of the Tenda CP3 IP Camera

In this section, we present the analysis of the Tenda CP3 connected camera and provide details of the hardware platform. Although we had full access to the IP camera during our analysis, our primary focus was on outlining the steps necessary for gathering as much information as possible using OSINT sources and firmware analysis. Consequently, we describe a methodology that could be applied not only to other IP cameras but also to various IoT devices.

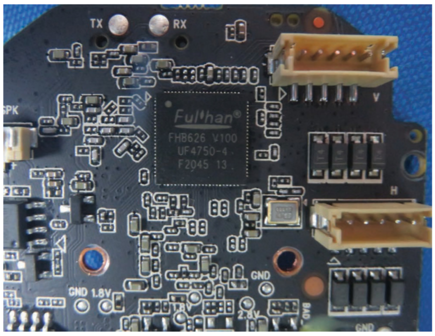
2.1 Hardware Analysis

The Tenda CP3 connected camera is based on a single PCB platform to which the lens and all extension PCBs are connected. We conducted an analysis of the camera's internals using its FCC ID, a unique identifier required for devices transmitting over radio frequencies in the United States. Upon locating the FCC ID on one of the labels attached to the camera, we entered its value into the FCC ID search form available on the Federal Communications Commission website. The FCC ID of the Tenda CP3 cameras is `V7TCP3`, with `V7T` representing the grantee code (i.e., Shenzen Tenda Technology Co., Ltd.) and `CP3` being the product code assigned by the grantee. By examining the internal pictures of the `V7TCP3` product (Fig. 1) we identified the main components of the main PCB.

The Tenda CP3 is equipped with a Fullhan `FH8626 V100` System on Chip designed for HD IP cameras, capable of multi-stream encoding in *H.264* format at 1080p resolution and 15 frames per second. Additionally, it features a Realtek *8188FTV* Network Interface Controller, providing support for *802.11b/g/n 2.4 GHz* connectivity. On the top side of the main PCB, we also identified a UART serial interface, indicated by the two *tx* and *rx* pads located at the top of the image, as well as a flash chip (partially visible on the left side of the picture).

2.2 UART Serial

After the initial hardware analysis, we decided to access the internals of the camera available to us to verify the information found via OSINT. Although the labels *tx* and *rx* were not printed on the PCB found inside our device, all the


(a) Top side of the main PCB.

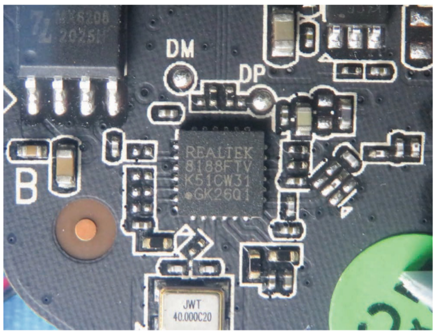

(b) Bottom side of the main PCB.

Fig. 1. Internal pictures of the Tenda CP3 accessed via the FCC ID public repository.

other components and their locations were exactly the same. We proceeded by connecting to the UART serial interface using a USB TTL adapter and minicom configured with a baud rate of 115200 (8N1).

During the first boot of the device we only connected to the *tx* pad of the UART interface with the *rx* pin of our USB TTL adapter to log the output of the system boot process. After analyzing the recorded log of the boot process, we identified several useful information about the bootloader (U-Boot 2010.06-dirty, with the possibility to interrupt the autoboot process by pressing `‘E’`), the OS (a Linux-3.0.8 ARMv7 Linux Kernel Image), the number and mapping of the partitions on the *spi_flash*, the name of some demons and applications started by the system (one of them being *telenetd*), and some configurations saved by the device, including the configured WiFi credentials printed in clear.

We then connected the *tx* pin of our USB TTL adapter to the *rx* pad of the PCB to interrupt the autoboot sequence, only to encounter a password-protected login prompt. Later, we will discuss how the password was recovered to access the U-Boot console.

CVE-2023-30354: Physical access and WiFi credentials disclosure - Tenda IP Camera CP3 does not defend against physical access to U-Boot via the UART; the Wi-Fi password is shown, and the hard coded boot password can be inserted for console access.

Base Score: **9.8 Critical**
Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

2.3 Firmware Extraction and Analysis

We extracted the firmware from the SOIC8 chip by directly connecting to the pins of the chip via a dedicated clip, as depicted in Fig. 2. We took care to isolate the processor from the flash memory to prevent any modifications during the firmware extraction procedure, thus enabling us to work with the corresponding image of the data found on the flash memory.

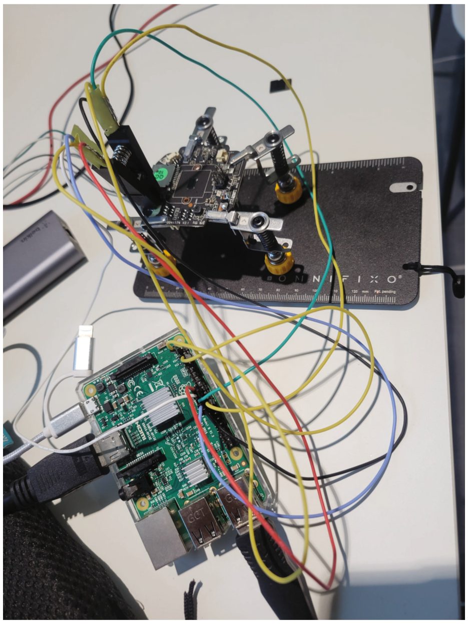


Fig. 2. Firmware extraction procedure.

We extracted the firmware image using the flashrom utility [8], and we modified the permissions on the saved image to be *read-only*. Additionally, we computed a fresh checksum of the saved flash image to ensure that we always had a clean image available.

After obtaining the firmware image, we manually extracted the partitions described in the boot process, resulting in 6 different files (one for each partition). Here, we provide a high-level analysis of the content of the partitions extracted from the flash image:

- **bootstrap** and **uboot-env:** partitions containing the configuration of the bootstrap and uboot environment;
- **uboot:** partition containing the uboot bootloader of the device;
- **kernel:** partition containing the root file system of the device. The root file system is stored in a compressed format.
- **data:** partition containing the user data used by the applications being executed on the device. This partition is formatted in a *jff2* file system.
- **app:** partition containing the file system on which all scripts, applications and configuration file of the Tenda CP3 IP camera. This partition is formatted in a *squashfs* file system

3 Detailed Analysis of the Tenda CP3 System

The detailed analysis of the Tenda CP3 camera combines both dynamic and static analysis on the content of the *app* partition of the extracted firmware image. Specifically, we are primarily interested in mapping all services being executed on the camera that accept any incoming traffic, and in enumerating interesting files found in the *app* partition of the camera via the firmwalker utility [6]. These files include UNIX configuration files (such as `/etc./passwd` and `/etc./shadow` files), bash scripts, and all executable files found in the partition.

In this section, we present the results of the analysis of the UNIX configuration files, the service mapping, and the bash scripts responsible for the initialization of the system. In Sect. 4, we provide a detailed analysis of the main programs found active on the camera.

The scripts used the steps outlined below are available on GitHub [2].

3.1 UNIX Configuration Files

We identified several different configuration files available in the *app* partition of the extracted file system. While some configuration files are related to the behavior of the applications executed by the system at start up (more on this in Sect. 3.3), by examining certain well-known strings in the text files, we were able to identify two extremely interesting files.

The first file is `ap_mode.cfg` (located at the root directory), which contains the default configuration for the access point exposed by the device while in configuration mode. This includes the interface label, the SSID type and prefix, the default IP address (`192.168.55.1`), DHCP range, and default password in plain text.

The second file is `shadow` (also located at the root directory), which serves as a modified version of the `/etc./shadow` file found in all UNIX systems that will be copied in the `/etc.` folder at start up (see Sect. 3.3). The *shadow* file contains a single user (`root`) and the hash (generated with *descrypt*) of the password associated with the root user. Since the length of the hash is limited to only 13 bytes, we attempted to reverse-engineer the password via a brute-force attack, eventually succeeding in recovering it. The recovered password has a length of 8 characters and is likely generated following a pre-defined scheme used by the vendor. Upon successful authentication with the recovered credentials via UART shell, we were also able to access the U-Boot shell after a couple of easy and intuitive modifications to the password.

3.2 Service Mapping

We conducted a network mapping on our device to identify all services accepting incoming traffic from external hosts. This mapping was performed from both an external attacker's perspective (e.g., by mapping the ports from a host connected to the same network as the device) and by accessing the internal shell of the camera (made possible by the recovered password found in Sect. 3.1) to identify the services handling incoming traffic.

External Mapping Report. We performed the external mapping using the *nmap* utility [9] to identify all listening ports accepting both TCP and UDP connections. The results of the network mapping procedure indicate that the camera is accepting *telnet* connections on port 23/TCP, *rtsp* (Real-Time Streaming Protocol) connections on port 8554/TCP, and exposes other unknown services on ports 843/TCP, 1300/TCP, 6688/TCP, 8699/TCP, 9876/TCP, 3702/UDP, 5012/UDP, 5683/UDP, and 19966/UDP.

We were able to access the camera via the *telnet* protocol using the `root` credentials recovered in the previous step, and via *rtsp* using the default credentials associated with the service, stored in plain text in the `ap_mode.cfg` configuration file. It is worth noting that these credentials are hard-coded and identical in all IP cameras from the same vendor, and cannot be modified via the corresponding smartphone application. Therefore, by accessing a network where a camera from the same vendor as the one used in our analysis is installed, anyone can access its live video stream using the same set of credentials.

CVE-2023-30351: Remote access via hard-coded credentials - Tenda IP Camera CP3 was discovered to contain a hard-coded default password for root which is stored using weak encryption. This vulnerability allows attackers to connect to the TELNET service (or UART) by using the exposed credentials.

Base Score: **7.5 High**
Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

CVE-2023-30352: RTSP feed access via hard-coded credentials - Tenda IP Camera CP3 was discovered to contain a hard-coded default password for the RTSP feed.

Base Score: **9.8 Critical**
Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

Internal Mapping Report. Following the mapping of services accepting incoming connections from outside the device, we exploited the previously found access via telnet to discover other potentially interesting services running on the device that could lead to further vulnerability findings. Initially, we analyzed the processes active on the device after gaining root access through the exploited telnet service. We found that only slightly more than 50 processes were active on the device, many of which were observed starting as system processes via the UART serial interface logs.

Upon observing the list of active threads on the device, we discovered that the vast majority were spawned from two different applications, namely *noodle* and *apollo*, which were also found in our copy of the flash memory in the *app* partition.

We then proceeded with the identification of processes associated with the open ports found in the external network mapping. The results of the `netstat` command highlighted that the *telnet* connection is managed by *inetd* (as expected), while all other connections are managed by either *noodle* or *apollo*. Specifically, *noodle* listens on ports 843/TCP, 1300/TCP, and 5012/UDP, while *apollo* serves all other ports, including 8554/TCP (RTSP). Since these two binaries serve as the main entry points for any external connection (including communication with the vendor's servers and the vendor's application), we will focus on their analysis in Sect. 4.

3.3 Bash Scripts

We analyzed the content of the bash scripts found in the `/etc./init.d` folder, which contains 5 different scripts:

- **S01udev:** responsible for creating some system folders and run the `udevd` demon and to start the *udevstart* program;
- **S02init_rootfs:** responsible for mounting the *data* file system partition;
- **S03network:** responsible for configuring the network interface;
- **S04app:** responsible for mounting the *app* file system, initializing different applications, and starting the *noodles* application;
- **rcS:** responsible for executing all the scripts found in the `/etc./init.d` folder starting with the `S[0-9][0-9]` regex expression.

Since the *S04app* script is the most interesting initializing script, we decided to analyze it in detail to recreate the entire initialization process of the applications being executed on the device and to find any potential vulnerabilities. The *S04app* script invokes another script (`chk_ver.sh`) that is responsible for updating any script found in the `/usr/bin` folder with newer versions found in the `/app` folder, if available. While it is possible to exploit this script to overwrite system applications with another file with the same name placed in the `/app` folder, it is worth noting that obtaining access to the system is necessary to perform this exploit. Given that the only user available on the device is the *root* user, the process of modifying a script is trivial and of little interest in our case study.

The *S04app* then proceeds to invoke three hard-coded scripts (`patch.sh`, `sys_init.sh`, and `app_init.sh`) from the `/app` directory, if available. Unfortunately, in both our device and flash image, we were unable to locate these scripts, as they are most likely related to the installation of a patch downloaded from the vendor's website.

After executing the *noodles* application, the *S04app* script identifies and mounts the SD card on the `/mnt/sd` mount point. If the SD card is mounted correctly, it then executes another script (`iu.sh`), which appears to be an update script. Upon further analysis, the `iu.sh` script copies the content of a file named *Flash.img* found in the root directory of the SD card to a temporary working folder (*/home*) and proceeds by copying its content to replace the entire flash memory. It is worth noting that since the entire system boot process does not check the integrity of the loaded flash image, it is possible to overwrite the system simply by inserting a properly formatted SD card with the malicious firmware image, without the necessity of accessing the device through the network.

CVE-2023-30356: Missing support for Integrity Check - Tenda IP Camera CP3 was discovered missing Support for an Integrity Check, allowing attackers to update the device with crafted firmware.

Base Score: **7.5 High**
Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:H/A:N

4 Binary Analysis: Noodles and Apollo

In this section, we present a detailed analysis of the two main binaries responsible for managing network connections on the Tenda CP3 camera, namely *noodles* and *apollo*. The primary objective of the analysis presented in this section is to outline the methodology we adopted in identifying vulnerabilities that could potentially allow for remote code execution on the device.

Our analysis is based on the reverse-engineered representation of the two binaries obtained with Ghidra [14], upon which we applied a customized version of the *rhabdomancer* script [13] to identify insecure functions handling external connections (`recv`, `recvfrom`, and `recvmsg`). Rhabdomancer is a Ghidra script designed to assist with vulnerability research tasks based on a candidate point strategy against software written in C/C++. It locates all calls to potentially insecure functions (the candidate points), which can be used to find insecure input access to the process. Additionally, we developed a tool on top of rhabdomancer that automates the entire process of reconstructing the function call sequence from the `main` function to the specified entry points [2], enabling us to easily map all the threads of the two processes to the different ports they use. Subsequently, we proceeded with a manual analysis to identify potential security vulnerabilities and design exploits to achieve remote code execution on our device.

4.1 The *Noodles* Binary

By executing our modified version of the rhabdomancer script, which specifically targeted functions relevant to our analysis (primarily *recv*, *recvfrom*, and *recvmsg*), we identified three candidate points in the decompiled binary. These points were labeled as `FUN_00014e68`, `FUN_0001fc14`, and `FUN_00012b7c`.

`FUN_00014e68`. This function is referenced in 5 other functions within the *noodles* binary. We will refer to these different invocations using the memory addresses of our reversed binary for simplicity.

1. 0x00011b04. The first reference is inside the `main` function, and is related to a socket listening on port 1300 (one of the ports already identified in the previous analysis step). In this invocation, `FUN_00014e68` is used to receive commands from the client, as confirmed by the string `receive_cmd from client<%d>: <%s> len = %d \n` found a couple of instructions later. The available commands (hard-coded in the binary) through this interface include some suspicious strings like `ELFEXEC`, `DOWNLOAD`, and `SYSTEM`.

2. 0x000123bc. The second reference to our target function is inside the `FUN_00012110` function, which is referenced by 3 other functions: `FUN_000128a0` (invoked by the `main` function after the `ELFEXEC` command is received), `FUN_00014674` (invoked by the `main` function after the `DOWNLOAD` command is received), and `FUN_000147ac` (invoked by the `main` function after the `UPGRADE` command is received). Upon further investigation, we confirmed that these references are all related to the previously discussed functionalities, and they can be used to trigger the execution of different scripts available on the camera.

3. 0x0001272c. The third reference is found inside two different functions: `FUN_000146e4` (invoked by the `main` function after the `UPLOAD` command is received) and `FUN_00014748` (invoked by the `main` function after the `FLASHDUMP` command is received). While via the former invocation it is possible to upload a specific file to the camera, by exploiting the latter invocation we demonstrated that it is possible to remotely upload a modified version of the firmware that will be copied to the flash memory on the next restart. This enables the exploitation of the vulnerability identified in Sect. 3.3 (**CVE-2023-30356**) without requiring physical access to the camera to upload a modified firmware version on the SD card.

4. 0x00013cbc. The fourth reference is found inside the function `FUN_00013c30`, which happens to be the *policy_thread* spawned by the function `FUN_00013df4` (directly called in the `main` function). The *policy_thread* is listening on port 843, accepts a fixed string (`policy-file-request`), and responds with a fixed XML structure.

5. 0x000144f8. The final reference is found inside `FUN_000143c0`, which is another function associated with the management of external commands accepted by *noodles*. Specifically, this function is called upon the reception of the `SYSTEMEX` command, which has already been exploited by other researchers in **CVE-2023-23080** to achieve remote code execution.

`FUN_0001fc14`. The second function containing a `recv` invocation is referenced only once in function `FUN_0001d2c8` in the *noodles* binary. However, this latter function is referenced three times.

The first reference (`FUN_0001d2c8`) is related to some WiFi connection tests and extends up to the `main` function (invoked after receiving a `SYSTEM` command containing the *STATUS* keyword). We analyzed the entire activation graph of this invocation, composed of 6 different functions, and established that this set of functions is used to test the status of a known WiFi SSID by the camera on system boot.

The second reference (`FUN_0001d1f8`) is also related to WiFi communication and is also invoked indirectly by the `main` function after a `SYSTEM` command is received. However, this second function accepts a different system command (*SCAN*) and performs a network scan on the `wlan0` (hard-coded) network interface.

The third reference (`FUN_0001d3b0`) is once again related to WiFi communication and invoked via the `SYSTEM` handler of the `main` function. This time, the function accepts a *SCAN_RESULTS* command and prints the output of the last saved network scan on the serial interface. We remark that all three of these handlers return the same value (`<SYSTEMEX_ACK>ok</SYSTEMEX_ACK>`), which is the default response for any successful connection via the `SYSTEM` command. Additionally, we note that all of these commands are vulnerable to remote code injection, and that any received command is executed on the device with root privileges without requiring authentication.

`FUN_00012b7c`. The third function containing a `recvfrom` function is actually the *multicast_thread* spawned by the `main` function and attached to port 5012/UDP. The *multicast_thread* is configured to accept two commands, namely `YGMP_SVR` and `YGMP_CMD`.

Upon reception of the former command, *noodles* opens different configuration files to read the current settings of the device, which are then returned as an XML structure to the caller. These settings include the *IP* and *MAC* addresses of the camera, its *serial number*, the value encoded in the *QR Code*, the *hardware version*, and other information.

The latter command (`YGMP_CMD`), however, is far more interesting in the scope of our work, as it allows unauthenticated remote code execution on the camera by sending a formatted XML payload. In particular, the payload accepts 3 different tags for parsing: *TARGET*, *MAC*, and *CMD*. Although the content of both *TARGET* and *MAC* are apparently not used except in some printing functions, the content of the *CMD* tag is compared to the *reboot* string. If the `strcmp` returns 0, the `FUN_00016ea8` function is called with the argument `/app/bin/cmd reset`; otherwise, the content of the *CMD* tag is passed directly to the same function.

Upon further inspection, we verified that the `FUN_00016ea8` function is a simple wrapper for the `system` function, with the arguments passed to the function being forwarded directly to `system` without proper sanitization. This allows unauthenticated remote code execution on the camera by simply passing a command different from *reboot*.

CVE-2023-30353: Unauthenticated RCE - Tenda IP Camera CP3 allows unauthenticated remote code execution via an XML document.

Base Score: **9.8 Critical**
Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

We emphasize that by following the described methodology, which is based on the analysis of the `recv` functions and their activation path, we were able to find all handlers for incoming connections managed by the *noodles* application. Furthermore, we demonstrated how to exploit these handlers to obtain unauthenticated RCE with root privileges on the device.

4.2 The *apollo* Binary

We employed the same methodology applied on the *noodles* binary to analyze the *apollo* binary. In particular, we want to remark that the vanilla rhabdomancer script on the whole *apollo* binary resulted in more than 128000 candidate points, while our tool returned less than 100 points, which we further reduced with a simple duplicate removal of different points on the same function calls.

The *apollo* binary utilizes 65 different threads to perform various tasks based on commands received from 7 different ports. Each of these threads eventually leads to one or more of the 25 `recv` functions. These functions containing the `recv` calls are solely responsible for managing incoming data, while the parsing of the received data structure is handled by the calling functions.

After thorough analysis, we successfully mapped each thread to a specific port listened to by the *apollo* process and identified the exposed functionalities.

- **3702/TCP** exposes ONVIF [15] *discovery*, *notification* and *hello* threads as required by the ONVIF Core Specification [3 threads];
- **6688/TCP:** exposes an HTTP server [3 threads];
- **8554/TCP:** exposes the *RTSP* service, which is used to access the camera video and audio [2 threads]
- **8699/TCP:** exposes a set of threads related to manage different functionalities of the camera [52 threads];
- **9876/TCP:** exposes the *yserver* TCP handler for incoming connections [3 threads];
- **5683/UDP:** exposes the *COAP* (COnstrained Application Protocol [5]) functionalities [1 thread];
- **19966/UDP:** exposes the *yserver* UDP handler for incoming connections [1 thread].

We proceeded with a more in-depth analysis of the main handler of port **8699/TCP** (`ut_cmd_server_init`) due to the significant number of threads directly associated with this port. We discovered that this connection is utilized for direct communication with the user application within the same network or via cloud services when in a different network. It accepts all possible commands that the user can provide through the application interface.

The incoming packets are handled by a dedicated thread (`ut_rcmd_server_proc`) and then passed to a parser function to identify the received command (`FUN_0007cb00`). In this function, the parser only checks that the received command starts with the character `!` and then proceeds to compare the remainder of the received string with hard-coded commands. If the received string matches one of the commands, then a thread is spawned and the desired function is executed on the camera. We identified 148 different commands (147 plus the `help` command, which returns the description of all the other commands), which are mapped to 50 different threads. As an example, the `audio_output_proc` manages various functionalities related to recording audio from the microphone (e.g., *audio_vol_in*, used to set the audio input volume, or *capture_audio*, used to capture audio from the microphone) and setting different speaker parameters. Meanwhile, functionalities related to playing audio on the camera are managed by the `audio_output_proc` (e.g., *loop_audio* and *play_audio*). However, if the string received by the `ut_rcmd_server_proc` thread and parsed by the `FUN_0007cb00` function is not recognized as one of the commands directly managed by the process, the `system` function is executed by passing the remaining string as the only parameter. By supplying a correctly formatted string to the service, we were able to achieve another instance of unauthenticated remote code execution on the camera.

Finally, we note that we also identified potential exploits on the other ports handled by the *apollo* process. While the methodology presented in this paper has proven effective in identifying vulnerabilities on our device, we emphasize that the process of behavior reconstruction (i.e., the detailed analysis presented on the *noodles* binary) and the subsequent crafting of exploits to verify a security vulnerability still heavily rely on human skill and experience.

5 Conclusions

This paper analyzes the process of vulnerability discovery on a consumer IP camera to demonstrate the effectiveness of the approach in the identification of security vulnerabilities. The methodology comprises five steps:

1. gathering relevant information from open sources;
2. physical access to a IP camera specimen aiming at identifying low-level attack vectors;
3. firmware extraction and static analysis of the whole file system, including configuration files, scripts and executables;
4. dynamic analysis of the network behavior, aiming at identifying all remote attack surfaces (such as open TCP and UDP port) of the connected IP camera;
5. in-depth reversing of all executables that implement network-facing services.

We provide a detailed example of the application of the proposed methodology by applying it on the widespread Tenda CP3 IP camera as a relevant use case. Our methodology allowed us to identify five new CVEs with a CVSS score ranging from 7.5 to 9.8.

We remark that the proposed methodology differs from the approaches that are commonly proposed in many related works, which only perform network-based analysis and fall short from executing a complete reversing of relevant executables.

To partially automate our approach we also developed a novel tool [2] based on Ghidra and rhabdomancer that is able to identify the functions managing incoming connection, to reconstruct their call tree from the *main* function for a fast identification of critical points in large binary executables.

Responsible Disclosure

Before publication of this work we contacted Shenzen Tenda Technology Co., Ltd. (May 2023) and disclosed our initial findings to them. We informed their representative that we had discovered several vulnerabilities in one of their products and mutually agreed to proceed with the responsible disclosure procedure once the CVE IDs for the vulnerabilities were assigned. We obtained 5 CVE IDs (out of the 7 initially requested, with 2 CVE IDs covering two vulnerabilities each) at the beginning of June 2023, and promptly reached out to their representative. As of time of submission of the camera ready of this work (July 2024), all details of our findings have been shared with the Tenda representative, who also assured that a patch resolving all these vulnerabilities is currently being developed and will be available in the next months.

Acknowledgments. This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

References

1. Abdalla, P.A., Varol, C.: Testing IoT security: the case study of an IP camera. In: 2020 8th International Symposium on Digital Forensics and Security (ISDFS), pp. 1–5 (2020). https://doi.org/10.1109/ISDFS49300.2020.9116392
2. ACES - Automotive, Cyber-Physical, Embedded and Security research laboratory. IWSec 2024 - Scripts and materials. https://github.com/SECloudUNIMORE/ACES/tree/master/IWSec2024. Accessed April 2024
3. Bathich, P., Malli, M., Hazimeh, H.: Exploiting vulnerabilities of IP cameras: lebanon case study. In: 2021 9th International Symposium on Digital Forensics and Security (ISDFS), pp. 1–6 (2021). https://doi.org/10.1109/ISDFS52919.2021.9486324
4. Biondi, P., Bognanni, S., Bella, G.: Vulnerability assessment and penetration testing on IP camera. In: 2021 8th International Conference on Internet of Things: Systems, Management and Security (IOTSMS), pp. 1–8 (2021). https://doi.org/10.1109/IOTSMS53705.2021.9704890
5. COAP: COnstrained Application Procotol. https://datatracker.ietf.org/doc/html/rfc7252. Accessed March 2024
6. craigz28: Firmwalker. https://github.com/craigz28/firmwalker. Accessed April 2024
7. Federal Communications Commission: FCC ID Search. https://fccid.io/. Accessed March 2024
8. Flashrom: Flashrom. https://flashrom.org/. Accessed March 2024
9. Fyodor: NMAP - Network Mapper. https://nmap.org/. Accessed April 2024
10. Kalbo, N., Mirsky, Y., Shabtai, A., Elovici, Y.: The security of IP-based video surveillance systems. Sensors **20**(17) (2020). https://doi.org/10.3390/s20174806, https://www.mdpi.com/1424-8220/20/17/4806
11. Li, J., Li, Z., Tyson, G., Xie, G.: Your privilege gives your privacy away: an analysis of a home security camera service. In: IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, pp. 387–396 (2020). https://doi.org/10.1109/INFOCOM41043.2020.9155516

12. Li, Y., Mandalari, A.M., Straw, I.: Who let the smart toaster hack the house? An investigation into the security vulnerabilities of consumer IoT devices (2023)
13. Marco Ivaldi: Rhabdomancer. https://github.com/0xdea/ghidra-scripts/blob/main/Rhabdomancer.java. Accessed April 2024
14. National Security Agency: Ghidra SRE. https://ghidra-sre.org/. Accessed April 2024
15. ONVIF: Open Network Video Interface Forum. https://www.onvif.org/. Accessed April 2024
16. Shwartz, O., Mathov, Y., Bohadana, M., Elovici, Y., Oren, Y.: Opening Pandora's box: effective techniques for reverse engineering IoT devices. In: Eisenbarth, T., Teglia, Y. (eds.) CARDIS 2017. LNCS, vol. 10728, pp. 1–21. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75208-2_1