École Polytechnique Fédérale de Lausanne

Searching for vulnerabilities in Barco's ClickShare devices

by Bradley Mathez

Master Thesis

Approved by the Examining Committee:

Prof. Dr. sc. ETH Mathias Payer
Thesis Advisor

Alain Mowat
External Expert

Alain Mowat
Thesis Supervisor

EPFL IC IINFCOM HEXHIVE
BC 160 (Bâtiment BC)
Station 14
CH-1015 Lausanne

August 19, 2022

I’m not afraid of dying. I’m afraid of not living.
— Mike Horn

Dedicated to my parents and my sister who have always supported me in life,
specially through the EPFL’s studies,
and no matter what crazy choices I have made.

Acknowledgments

I would like to express my deep gratitude to my semester project's supervisor, Mr. Alain Mowat, for his expertise, very appreciated comments and enormous support during the whole research, without whom none of this would have been possible in such a short period. I have learned a lot since the beginning of this project, especially through Alain Mowat's guidance but also with the advices and help coming from the other SCRT's team members. The subject of this project was challenging and extremely interesting, thank you.

I am further grateful to HexHive at EPFL and Prof. Dr. sc. ETH Mathias Payer for making this semester project realizable. Pr. M. Payer was always available and helpful. Without him, this project would have been possible.

I wish to extend my special thank to Mr. Nicolas Oberli and Mr. Karim Sudeki without whom the eMMC's content extraction and analysis would have been done with much more trouble. In addition, their advice and expertise were more than needed.

Thanks should also go to the SCRT's team for their continuous support, the many fascinating discussions and infinite advice. The whole team is absolutely passionate about cybersecurity and they will always find a way to transmit it to others.

Thanks to all of you!

Lausanne, August 19, 2022 Bradley Mathez

Abstract

The ClickShare devices allow to share the screens of computers on a presentation system, such as a TV screen. Since they are becoming more common in the companies' infrastructures, their security against cyber threats is needed. In 2019, WithSecure published their analysis about ClickShare's devices cyber security and discovered fourteen vulnerabilities. In the present research, we evaluated the security improvements of ClickShare's devices two and a half years later. We also searched for novel attack methods. We assessed the security of the base unit and the desktop application in black box by exploiting techniques a that real attacker could use. Our analysis shows that ClickShare's devices strengthened their security since December 2019 and seem well hardened against black box attacks. As the devices' security has been fortified, we did not have access to their inner working. Thus, we were not able to test their security in white box.

Contents

Chapter 1

Introduction

In 2012, Barco, a Belgian technology company specialized in digital projections systems [13], launched ClickShare devices, a system able to share the screen of a computer on a presentation frame and aiming at increasing the collaboration in meeting rooms. The idea is to keep the design simple and effective. In December 2019, a cyber security company, WithSecure (formerly F-Secure) published an advisory with fourteen vulnerabilities assessing the security of the most important aspects [51]. Three principal elements compose a ClickShare ecosystem: the base unit, the desktop application running on the user's machine, and optionally, buttons connected to the user's computer.

ClickShare's teams correctly managed the safety of their firmware updates by encrypting them before releasing their packages on internet. The devices decrypt the computer code during the update process. However, it represented one of the main obstacles on our road to assess the security of the ClickShare equipment. Besides, we deduced from the desktop application binaries that they were written in C++ with classes. This implies several indirection during reverse engineering. Moreover, ClickShare's teams chose to harden the reverse engineering work by obfuscating and stripping their code. Lastly, as we wanted to evaluate the system security, we extracted the storage memory content from the base unit. It required material and expertise we did not have initially.

This project targeted the components' security evaluation of ClickShare's devices, as already performed two and a half years ago by WithSecure, and verify that the situation has improved since then. Even if WithSecure worked amazingly, we estimate that it is necessary to confirm the system's evolution to keep a high level of safety, especially for products aiming at companies as clients, and not individuals, since the impact is usually higher in case of cyber-attacks. For example, the Mirai botnet created in 2016 was only formed with *Internet of Things* (IoT) devices and was used in some of the most disruptive *Distributed Denial of Services* (DDoS) attacks [24]. Besides, it exists a wide range of specialized attacks on the IoT [26].

Our approach for this project was to imagine how a malicious cyber-actor would attack an organization through ClickShare systems, and what would be the potential impact of this assault to maximize it in our tries. We chose to act as a nefarious user to understand them and to better defend the infrastructures afterwards. For this purpose, we bought a ClickShare device (C-5 base unit) and attempted to attack it directly in black box, i.e., only with the public documents, our skills, and without inner workings details.

We began our evaluation by a reconnaissance of the components through the official documentation and WithSecure's report. This phase consisted in a visual examination of the material too, but also of network scans, and inspection of the exposed services. Secondly, we organized the attack surface, the targets and the potential incursions we wanted to endeavor. Then we launched the several assaults we thought about and adapted the plan gradually along our discoveries. The last stage was to physically disassemble the components to solder cables on them, extract, and observe the storage memory content.

With this document, we wish to verify that the security of the ClickShare devices has improved since WithSecure's report in December 2019 to better protect the end users.

We discovered that the systems have been well hardened against cyber-threats. However, there are still security improvements possible. We spotted a weak password utilization for the `root` account of the base unit as the one WithSecure's reported in 2019. There might also be potential buffer overflows in the desktop application, perhaps leading to end users' computers compromises. Moreover, we noticed that the video shared content between the desktop application and the base unit is lacking a key negotiation algorithm, as WithSecure already disclosed earlier.

We propose this document as participation to strengthen the safety of the ClickShare's devices by having evaluated it during six months and tried several techniques upon them to validate their security amelioration over the last two and a half years. We believed it necessary, as presentation systems are becoming more and more common nowadays. Furthermore, the discovered issues will be sent to the teams of ClickShare to contribute to the improvement of the security.

Chapter 2

Background

In this chapter, we will present an overview of the different parts forming a ClickShare environment. We will cover the interaction details later in chapter 3. Then, we will discuss the WithSecure's report about ClickShare's security [51] (published in 2019) and show the most important vulnerabilities among the fourteen they have discovered.

2.1 ClickShare presentation

As we have already referenced before, ClickShare ecosystems aim at sharing the monitor of a device, e.g., a laptop, on a presentation structure, e.g., TV screen, projector. Three main elements compose a ClickShare environment: the base unit, buttons, and desktop application. Figure 2.1 summarizes these three modules and shows how they are connected to each other. The warning signs depict the locations we have attempted to attack during this project, but the details will be developed subsequently in this document.

The base unit (yellow in Figure 2.1) is plugged into the presentation system. It represents the core of the environment, since the other components cannot work properly without a base unit. It proposes a Wi-Fi *access point* (AP) where the desktop application and the buttons can connect to, but the desktop application can also connect to the base unit through Ethernet. Moreover, the base unit exposes an administration web interface used to manage the base unit, and an *application programming interface* (API), on port 4003, satisfying the same purpose as the administration web interface, but eases the work when there are a lot of base units to set up. It also offers several services (they are not depicted on Figure 2.1 to simplify the scheme) such as AirPlay, Miracast MICE, Google Cast, Cloud administration, or automatic updates.

Multiple buttons can associate themselves to a single base unit. They only have to be paired with it. A button (gray in Figure 2.1) is attached by USB to the device wanting to share its screen.

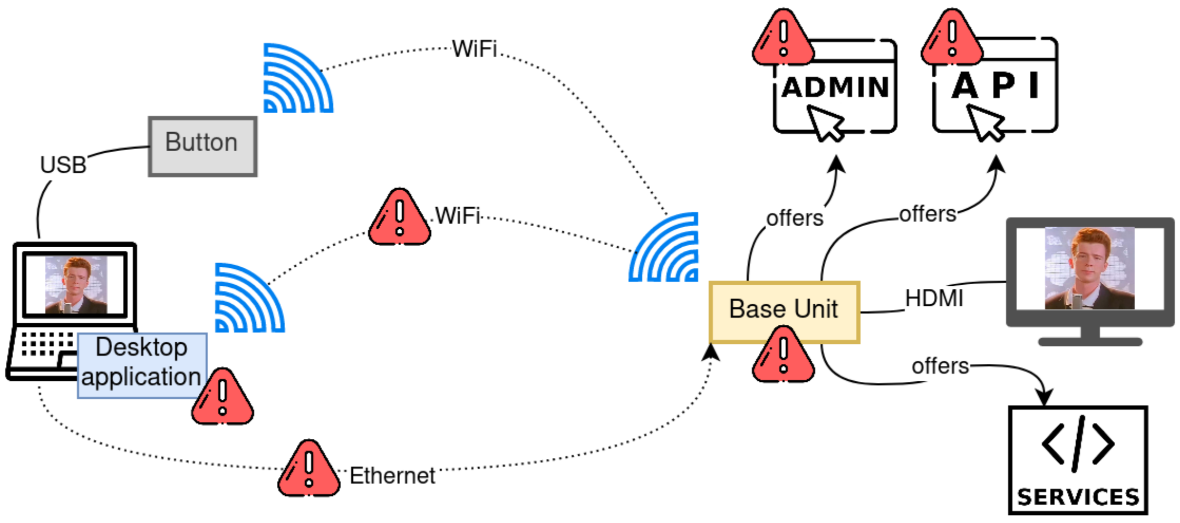


Figure 2.1: Simple representation of ClickShare environment. The warning signs denote the spots we have tried to attack in this document. The particulars are discussed later in this paper.

Its design is very simple and can be summarized as a button with a USB cable connecting to a base unit through Wi-Fi. The desktop application is needed to control the button, since it employs a proprietary communication protocol.

The desktop application (blue in Figure 2.1) is the program used to discuss with the base unit either through a button, or without it over Wi-Fi or Ethernet directly. It also detects the base units to which it can connect. We will detail these components later in this document.

2.2 WithSecure's report

During their investigation, the researchers at WithSecure initially attacked the base unit and noticed that two CVEs (check CVE-2017-7936 [32] and CVE-2017-7932 [31]) affected the processor (a customized version of the i.MX6 DualLite SoC manufactured by NXP [51]). These have allowed the security team to execute a modified U-Boot program and get full access to the device through the serial console. With these first vulnerabilities, WithSecure extracted the code running on the base unit and analyzed it to continue the attack [51].

Having full access to the base unit allowed them to identify new vulnerabilities on the Click-Share's environment, such as the following attacks on the buttons. The base unit and the computer employ a *USB dongle bridge* program to communicate with the button. This is a group of commands that can be sent to buttons to discuss with them. WithSecure discovered that some of them were prone to OS commands injections (check CVE-2019-18830 [5]). Thus, they used this issue as an initial foothold on the buttons to start an interactive console through the serial port with the privileges of the USB dongle bridge, i.e., *nobody* [51].

Moreover, some commands can only be accessed with a signed certificate through mutual peer authentication, e.g., changing device configuration or updating software components. However, the certificate chain verification during the USB authentication was vulnerable at two steps (check CVE-2019-18826 [1]). Firstly, in the certificate store initialization (employed to offer a set of trusted certificates), the implementation adds both the trusted *Certificate Authority* (CA) and the user-provided certificate. Hence, a self-signed root certificate crafted by the user will be unconditionally accepted as a trusted root CA. The second vulnerability concerns the certificate verification. It only verifies the certificate validity against the trusted certificates. Since it was possible to inject trusted root certificates, an attacker could always bypass the certificate verification and get access to the otherwise restricted commands [51].

Afterwards, WithSecure noticed that ClickShare's buttons were configured with a default `root` password, i.e., *root:letmein* (check CVE-2019-18828 [3]). They cracked its hash extracted from `/etc/shadow` and used it to elevate the privileges they had, through the serial console. Furthermore, since the passwords were stored in an immutable system partition, even the administrators could not modify it [51].

With these two execution paths, WithSecure had root accesses to both the base unit and the buttons. Moreover, WithSecure discovered that the *Joint Test Action Group* (JTAG) port was not disabled on the buttons (check CVE-2019-18827 [2]). This is linked to the choice of the *System on Chip* (SoC) used to build buttons, because the SoC did not provide an option to permanently neutralize the JTAG port, such as a microscopic *electronic fuse* (eFuse). Then, WithSecure abused that hardware flaw to obtain a different access and extract the *data encryption keys* (DEK) [51].

The security team has also revealed that the button's *system logs* (syslog) were transmitted in clear text through Wi-Fi to the base unit [51]. We will discuss it later, as we managed to verify it during this project.

Using the previous access to the base unit, WithSecure have discovered that there was a test certificate signed by the production CA with its corresponding private key (check CVE-2019-18831 [6]) [51]. It implies that the malicious actor could perform attacks such as *Man-in-the-Middle* (MitM) without the need of more information, except this test certificate and its private key.

Exploiting the previous flaw (with MitM attacks), WithSecure also divulged a vulnerability affecting the media stream (check CVE-2019-18833 [7]). They decrypted the TLS traffic between the base unit and the desktop application during the start of the content sharing and discovered that there was not any key negotiation mechanism to exchange the encryption key of the media stream [51]. In other words, the key was sent directly from the base unit to the desktop application. It implies that if an assailant could perform a MitM attack, then they could easily get the key, decipher the data stream, and see the currently shared content.

Moreover, the buttons exposed their file system through USB, more specifically, we can access the files of the application which the user needs to employ the buttons. WithSecure noticed that these were writable, and exploit this possibility to modify the application on the buttons with their own, but then the application was not signed anymore. However, they also noted that the desktop application loaded some missing *Dynamic-link library* (DLL). It implies that an attacker could drop a nefarious DLL at the path the application will search for a nonexistent DLL, which will be loaded by the desktop application (DLL-sideloading, check CVE-2019-18829 [4]). That way, the desktop application stays signed, but the code it executes may be malicious [51].

Nonetheless, they identified more vulnerabilities on the button and the desktop application that we will not detail in the report from WithSecure [51]. Table 2.1 summarizes the major weaknesses discovered by WithSecure that we will discuss in this paper.

CVE Identifier	Vulnerability name
CVE-2019-18826 [1]	Incorrect certificate chain verification during USB authentication
CVE-2019-18827 [2]	JTAG access is not permanently disabled
CVE-2019-18828 [3]	Weak, hardcoded credentials
CVE-2019-18829 [4]	Automatic execution & DLL sideloading
CVE-2019-18830 [5]	Multiple OS command injections in *dongle_bridge*
CVE-2019-18831 [6]	Test credentials signed by production CA
CVE-2019-18833 [7]	Insufficient protection for media streams

Table 2.1: Major vulnerabilities discovered by WithSecure that will be discussed in this document.

Chapter 3

Design

Here we detail the ClickShare's components interactions and its hardware. We then discuss the targets we have selected to attack and justify these choices. This chapter also clarifies the configurations of the attacks.

3.1 Objectives

The main objective of this project is to assess the security level of the ClickShare's solution and evaluate if improvements have been made since WithSecure's report in 2019 [51]. Our goals are to compromise the base unit and get access to the network it is connected to, the computer of someone using the desktop application, and to intercept the shared content between the desktop application and a base unit.

3.2 ClickShare's details

This section details the ClickShare hardware and software components and their interactions. We have decided to analyze a C-5 ClickShare base unit with the latest update of the desktop application, because the C-5 is part of the new models proposed by Barco after they patched the flaws pointed out by WithSecure's report and changed the hardware.

3.2.1 ClickShare's hardware

The base unit possesses a power supply port, HDMI port, Ethernet port, USB-A port, USB-C port and a Wi-Fi card to connect with the outside world. Besides, when the plastic case is closed, it exposes a reset button (bottom yellow in Figure 3.1) and a main button with red and white LEDs (top yellow in Figure 3.1) to the user to interact with the base unit. Furthermore, we noticed an inaccessible button (light blue in Figure 3.1) when the plastic cover is closed. In red, we can observe three potential debugging ports, but we will discuss them later in this document.

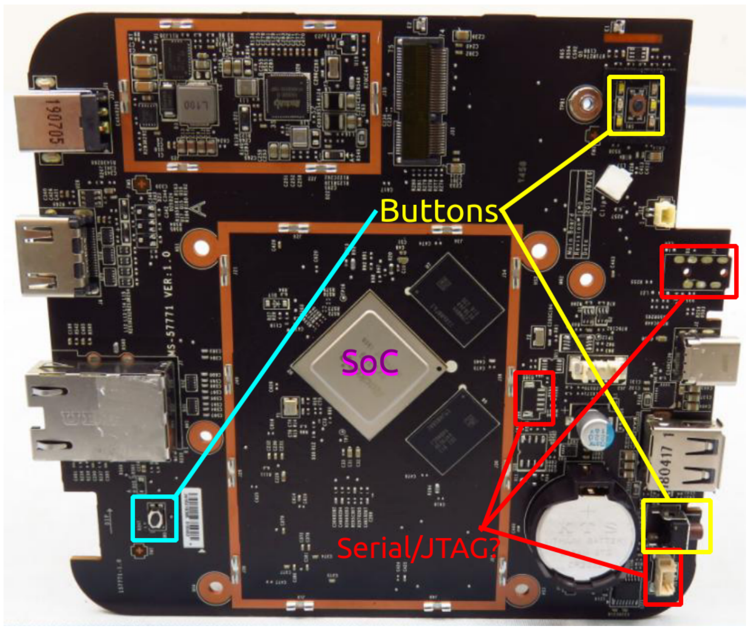


Figure 3.1: ClickShare C-5 base unit, recto. Red: possible debug ports (e.g., JTAG, serial, USB), yellow: accessible buttons when the plastic case is closed, blue: inaccessible button when the plastic cover is closed. Source: https://fccid.io/2AAED-R9861511/Internal-Photos/Internal-photos-4504760.pdf

The ClickShare's base unit is built using a custom board. Figure 3.1 represents a picture of the ClickShare's base unit's internal components. The upper left orange rectangle takes care of the power supply. The bigger orange rectangle contains the *System on Chip* (SoC) and the *Random Access Memory* (RAM). The current SoC is a RockChip RK3399. Figure 3.2 depicts the other side where there is an *embedded MultiMediaCard* (eMMC) serving the purpose of a small flash memory. The model is `SDINBDG4-8G` (8 Gigabytes) from SanDisk. The eMMC stores the base unit's *Operating System* (OS).

3.2.2 Components' interactions

In this section, we describe in detail the interactions between the ClickShare's components while browsing the diagram in Figure 3.3. The warning signs represent the elements we have attacked in this project. The choice of the targets is explained in section 3.3. Regarding the base unit, we already have noticed that it exposes an API and an administration web interface in section 2.1. We will portray them here. Afterwards, we describe the desktop application's components and finish with the interactions between the desktop application and the base unit.

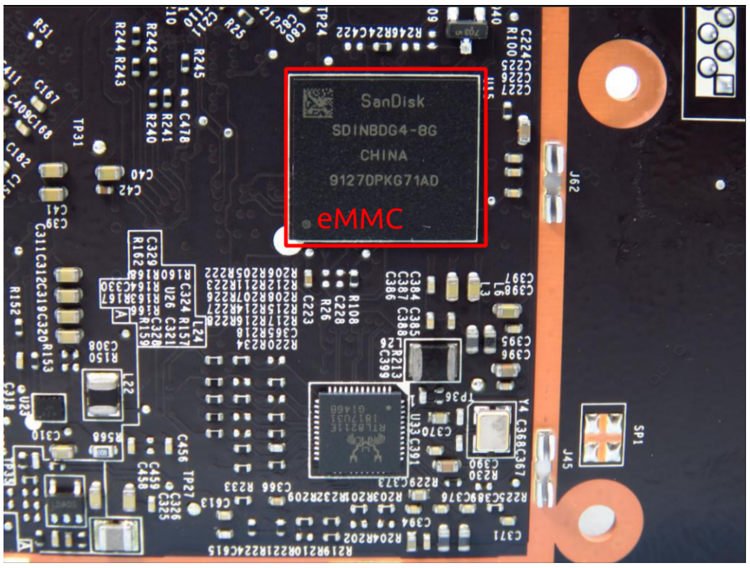


Figure 3.2: ClickShare C-5 base unit, verso. Red: SanDisk eMMC SDINBDG4-8G memory flash (8Gb). Source: https://fccid.io/2AAED-R9861511/Internal-Photos/Internal-photos-4504760.pdf

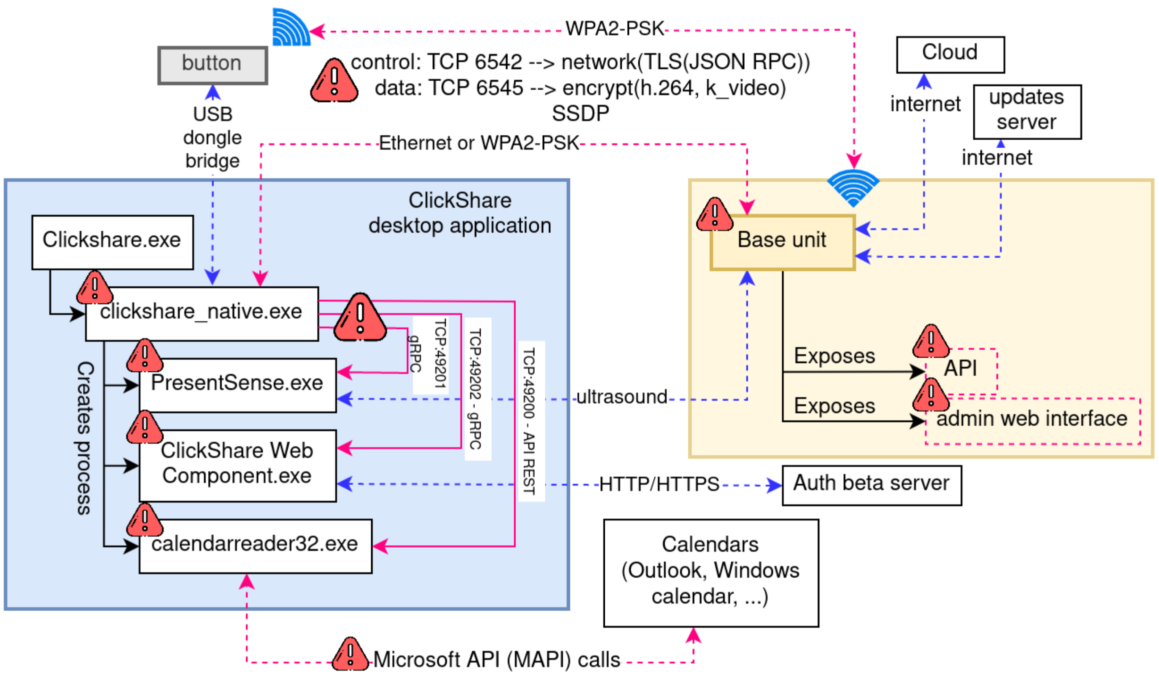


Figure 3.3: Components interactions details. The warning signs with pink links represent the components we have attacked in this document. Blue box: desktop application, yellow box: base unit, grey box: button

Base unit

The yellow rectangle represents the base unit in the Figure 3.3. The ClickShare's base unit can be administrated through the administration web interface, or the API it exposes. They have almost the same functionalities and they both require to be authenticated, but their implementations vary. The API uses *HTTP-basic-access-authentication* while the administration web interface uses cookies containing randomly generated identification values. An administrator can, for example, alter the settings of the base unit, upload wallpapers to display on the presentation system, upload XML configuration files, enable or disable the different services, or modify the network settings.

However, the administration interface and the API differ in their inputs sanitization processes when uploading wallpapers. Conducting this action through the administration web interface, it only accepts file name ending with `*.jpg` or `*.png` and verifies the document's header to match its extension (otherwise it returns an error and discards the picture). The API endpoint of wallpaper upload performs the same check, but the API endpoint of the XML configuration upload only validates that the file name ends with `*.jpg` or `*.png` without checking the file's content. Due to this flaw, the base unit administrator (the API is authenticated) can upload a document containing arbitrary data. We will discuss the attack and its results later in this document.

Moreover, the base unit can connect to a cloud and the supervisors can administrate the base unit remotely on multiple locations at the same time. It is also possible to connect the base unit to internet and get automatic updates from the related servers.

Desktop application

We analyzed the Windows version of the desktop application, since it is only available for Windows and MacOS. As we have seen before in this document, the desktop application (represented in the blue rectangle in Figure 3.3) connects the laptop to a base unit. Five principal executables constitute it. The first two, i.e., `Clickshare.exe` and `clickshare_native.exe`, are in charge of loading the last application version. `clickshare_native.exe` should load the various components as well, while being the central binary that collects and analyzes the results before preparing them for display to the user across its *Graphical User Interface* (GUI). ClickShare's simplicity idea is felt in the design of the GUI too. It can be summarized as a black window with only one fillable text field waiting for a base unit's IP address or a host name to connect to. The GUI also proposes a list of base units it managed to detect through *Simple Service Discovery Protocol* (SSDP).

`PresentSense.exe` offers a *gRPC Remote Procedure Call* (gRPC) server that can communicate employing *protocol buffers* (protobufs) [29]. Its objective is to listen to the ultrasound by using the computer's microphone to discover the existence of base units around it and deliver the results to `clickshare_native.exe` when it asks for it through gRPC on port 49201.

We did not fully understand the purpose of `ClickShare Web Component.exe`, but we observed that it is used to connect to the beta server authentication to verify the identity of someone wanting to get access to the beta program. Since we can create an account and get an admission to the beta without restrictions, we did not investigate this field more than necessary. However, we noticed that `clickshare_native.exe` communicates with it through gRPC on port 49202.

Regarding `calendarreader32.exe`, its objective is to read the planned events saved in the various computer's calendars, e.g., Outlook from Microsoft Office suite, Windows calendar, and extract meetings containing links to video conferences. *Microsoft Application Programming Interface* (MAPI) allows to interact with the Windows calendars and the Outlook calendar. The events go through a *regular expression* (regex) search. If at least one of the regexes matches, then the meeting is marked as interesting and is sent to `clickshare_native.exe` if it asks for it over an API request. The response is returned in *JavaScript Object Notation* (JSON) format.

Communications between desktop application and base unit

As we already discussed, the base unit opens a Wi-Fi AP for the buttons to connect to, but it can also be employed by the desktop application or by a base unit administrator's computer to modify the device's settings. The Wi-Fi communicates across WPA2-PSK (i.e. *Pre-Shared Key*). The key is either shared to the button by associating the button to the base unit through USB, or written by the user to connect to the Wi-Fi AP. The desktop application can also connect to the base unit by using the computer's microphone to detect and associate to it, as it can produce ultrasound. Additionally, it is possible to connect to the base unit over Ethernet as well.

We can split up the information exchanged between the base unit, and the desktop application in three parts. The first is the easiest one since it is only SSDP to discover the base units over the network. The second is the control flow, and it manages the connection. For example, it establishes the connection between base unit and the desktop app, notify the base unit if the user wants to disconnect, share their screen or observe the currently shared content. The communication uses a custom version of the JSON-RPC protocol[1] with a small header containing the dimensions of the JSON-RPC packet, but also other details such as the size of the pictures that can be transported aside the JSON-RPC. The last one is the data flow and includes the video stream encrypted with a key sent over the control flow.

[1]another variant of RPC employing JSON and created in 2010

3.3 Attack design

We decided that the purposes of an effective offensive on the ClickShare's ecosystem were to compromise either the network to which the base unit is connected, or the computer of users through their button or desktop application utilization, or visualize the currently shared content.

Now that we understand the various components more specifically and the ambitions of a successful attack, we will clarify the selection of targets, but also how we targeted them and why we chose these types of assaults.

3.3.1 Choosing targets

We highlighted the targets in Figure 2.1 and Figure 3.3 with warning signs. Since the latter is more detailed, we will pursue the explanation with this scheme. As we have already seen, the ClickShare's ecosystem contains three main elements and we will split up the arguments using this separation. We will start with the desktop application, then continue with the communications between the previous part and the base unit, and finish with the base unit. This represents the sequence in which we targeted them with the only exception that we began with the base unit's API and administration web interface first since it is usually the attacking spots demanding the less effort. However, to keep this document organized, we decided to present them in the order we initially exposed.

Desktop application

As discussed before, multiple binaries compose the desktop application and they converse with *Inter-Process Communications* (IPC) such as gRPC and APIs. The objective of attacking this element was to better understand how computers communicate with base units and we were looking for possibilities to compromise ClickShare's users. Since the first binary, i.e., `ClickShare.exe`, only starts the execution chain but does not take care of the transmissions or any other features, we focused on the rest of the chain. `clickshare_native.exe` is the primary binary (See section 3.2.2), then it was a central target as well. The child processes, i.e., `PresentSense.exe`, `ClickShare Web Component.exe`, and `calendarreader32.exe`, were also interesting to investigate, since they communicate with `clickshare_native.exe` and interact with the real world. Thus, an attacker might control inputs and influence the application's comportment.

We also selected the IPC as objectives, because we can intercept the exchanged messages and modify their content to impact the program's behavior. We chose MAPI calls, because we wanted to question the compromise of a target through a simple meeting invitation. As discussed in section 3.2.2, the desktop application reads the calendars' events and notifies the user of the close ones over its GUI. Hence, the desktop application must examine the events content before displaying them to the user. The focus was put on the event parser and the objective was (as always) a *Remote Code Execution* (RCE).

Communications

Besides the binaries running on the user's computer and the ones on the base unit, the communications between them are very attractive as long as we wish to monitor and look at the shared content. Note that the information of the transmissions are independent of the link they go through and are composed of three pieces. We did not select the base unit discovery with SSDP as an objective, because the impact of a base unit impersonation is limited to flooding the ClickShare's base units proposition list.

The other parts represent the control and the data flows. As explained in section 3.2.2, the data flow contains the shared content. Consequently, it is a target by design according to our objectives[2]. The last flow takes care of the connection's control (See section 3.2.2). Then, it seemed logical to aim for it.

Base unit

In section 3.2.2, we explained that the base unit offers an API and an administration web interface. Normally, only an authenticated user can interact with them and modify the base unit's settings. Therefore, they are particularly interesting to target and search for vulnerabilities or logical bugs that could lead to a *Local Privileges Escalation* (LPE) or a RCE. They were our first goals since we do not need much set up to assess their security and it could have given us a quick internal access.

Finally, the base unit itself, as a hardware component, was also a target. It has been presented in subsection 3.2.1 earlier in this document, yet the objective was to obtain an initial foothold to gather more information about the base unit's inner workings. To achieve our ambition, our first targets were the potential debug ports (in red on Figure 3.1), i.e., JTAG, serial, USB. The next focus was the eMMC (in red on Figure 3.2), because it may contain the firmware, the programs running on the base unit, some hard-coded keys, or other fascinating details.

3.3.2 Testing targets

In subsection 3.3.1 we considered the targets. In the present passage, we will describe the attacks we planned to perform over the objectives. However, we began by running the desktop

[2]See section 3.3: "We want to access the shared content"

application and tried to understand the purpose of each component when they are connected together. Like the previous section, this one will be segmented in the three same parts.

Desktop application

As mentioned before, one of our main objectives was to compromise the user's machine that runs the desktop application. Since we had the binaries installed on our computer, the first attack phase was the reconnaissance. We wanted to perceive how the desktop application was conversing with the other elements. We will assess this primary stage by running the executable, and actively monitoring the communications channels and the relation between the desktop application components to better understand their respective goals. The second phase was the binaries' static analysis while targeting the user's inputs to comprehend their paths in the executables and observe possible dangerous function calls where a user can control some parameters, e.g., `snprintf`, `strncat`, `memcpy`, or `fread`. We also wanted to extract the protobufs from the binaries to discuss with the gRPC servers and perceive their conversations.

Communications

Concerning the communications, our objective was to intercept them and decipher the shared content. We began with a reconnaissance stage allowing us to discover that the transmissions between the desktop application and the base unit took place through multiple connections (as we can understand in Figure 3.3 and we explained it in subsection 3.2.2). We noticed that the control flux was encapsulated in TLS, and we did not recognize the pattern of the data flow. Thus, the attack we planned was a MitM on the control flow to collect more information about the data flow, as the WithSecure's report described [51].

Base unit

Recall that our last objective was to compromise a base unit and directly access the network it was connected to. This mission began, like the others, with a reconnaissance phase. We first planned to use and configure the base unit to grasp the overall idea of its use, detect the potential entry points, then scan the base unit for open ports, and gather details about the services running behind these ports. Subsequently, we wanted to continue with a hardware visual inspection step, infer the components' identity, and their purpose. Afterwards, we aimed to conduct hardware attacks to glean internal data for analysis.

We also intended to collect information about the administration web interface and the exposed API. Since they both are used to manage the base unit, we scheduled to test various web attacks. We will expand the details in chapter 4.

Chapter 4

Implementation

This chapter covers the implementation of the attacks described in chapter 3. It will be split up in three parts as we already did earlier in this document, the desktop application, the communications between it and the base unit, and the base unit itself.

4.1 Desktop application

To start the attacks and verify their efficiency, we firstly had to disable the *Address Space Layout Randomization* (ASLR) on Windows. However, since the ASLR global deactivation did seem to affect the binaries' behavior we also had to modify their headers by flipping a bit to deactivate ASLR. For this purpose, we used *PESecInfo* script from OsandaMalith [40]. Now we could investigate desktop application crashes and set breakpoints to understand its behavior with *WinDBG* [54] and *gdb-peda* [37].

Then, we began with a dynamic analysis with *gdb-peda* [37], *APIMonitor* [12], and *ProcMon* [53], combined with a static analysis with *Ghidra*[1] [8]. It led us to a better understanding of each component. We continued to reverse engineer the desktop application, binary by binary to figure out their goals. We focused on `calendarreader32.exe` to retrieve its API endpoints and parameters (recall the Figure 3.3). By combining the reverse engineering process with dynamic analysis, we managed to find the API end points and their parameters. Remember that the server waiting for API calls is running on the user's computer and listening for connections on `127.0.0.1`, on port 49200. Thus, the contacts may only come from the *localhost*. However, we assumed that an attacker could send packets on this interface by other means, e.g., a *Cross-Site Request Forgery* (CSRF). On `calendarreader32.exe`, we also examined the MAPI calls and the event processing after retrieving them. We thought that it could be amazing to compromise

[1] "A software reverse engineering (SRE) suite of tools developed by NSA's Research Directorate in support of the Cybersecurity mission" — Ghidra

Tools	Windows	works with stripped binaries
WinAFL + Intel PT mode	✓	
WinAFL + DynamoRIO	✓	✓
WinAFL + sygyzy	✓	
NetAFL	✓	
Qiling	✓	✓
QEMU	✓	✓

Table 4.1: Summary of tools we considered for the desktop application analysis.

any user employing the ClickShare's desktop application through a simple meeting invitation. *Ghidra* helped us for this task as well.

We continued the investigations with the gRPC servers and their endpoints. gRPC is a protocol that minimizes the packet's size by establishing a communication protocol on every conversation's extremity. This protocol depends on the project since it needs to be compiled for each design. The files produced by the compilation are called *protobufs*. These are sort of dictionaries allowing both communicating sides to encode and decode the packets to reduce their sizes. Without the *protobufs*, it is complex to fully understand the transmission purposes. We tried different methods to extract the protocol buffers, but the one that worked was the usage of *pbtk*, a simple tool to retrieve protobufs from binaries [39], combined with *BloomRPC*, a tool created to work with protobufs and target the gRPC servers' endpoints [19].

Afterwards, we wanted to find dangerous functions calls in the desktop application's components, mainly in `clickshare_native.exe`. A dangerous function call is when a malicious actor can control the parameters and which can produce faults in the current running program. For example, `memcpy(dest, src, n)` where the `dest` is of fixed size and where the attacker can impose `src` and `n` (it could lead to a buffer overflow). We performed the research of these function calls with automated tools, such as *Joern* [28], *BinAbsInspector* [34], and *Ghidra's scripts*[2] [47]. The binaries are stripped C++ with some obfuscated parts and classes which implies a lot of indirections.

Thereafter, we desired to continue the analysis with the emulation and fuzzing of the various executable files. We planned to test fuzzers, emulators and instrumentation programs to find the most suitable ones. According to their descriptions, we build the Table 4.1 summarizing their potential usage in our case. We want them to be able to work with stripped *Portable Executable 32 bits* (PE32) where we do not possess the source code, i.e., no compilation is possible.

Finally, we considered *DLL hijacking* attack. The concept of this attack is to identify the DLLs the programs try to load, but that are not directly found. The idea is to develop a nefarious DLL that will be loaded before the genuine DLL by choosing a directory that will be searched for DLL before the real one, or simply that the legitimate DLL is missing [35]. We created the malicious DLL using *msfVenom* from the *MetaSploit-Framework* [42].

[2]We used modified versions of public custom scripts to fit our study case

4.2 Communications

We started to intercept the communications between the base unit and the desktop application going through port 6542, the control flow (see Figure 3.3). We began with the well known *Burp proxy* [41], then we tried some other tools such as *Ettercap* [27], *Bettercap* [18], *MITMProxy* [50], and *TProxy* [46], but they kept saying that there was no cipher suites in common between the client and the server. We investigated this problem with *Wireshark* [25] and noticed that the client (here the desktop application) only proposes the cipher suites: *TLS_ECDHE_EDSA_WITH_AES_256_GCM_SHA384* using the deprecated *sect283k1* elliptic curve for the key exchange algorithm. This was specified in the security whitepaper published by ClickShare [22]. Thus, we fixed this issue by working with *OpenSSL* and *FIFOs* as proxy, because we can specify the cipher suites in use, and *OpenSSL* has several cryptographic functions directly implemented (even some deprecated ones)[3] [33].

Nevertheless, we still had to bypass a certificate verification performed by the desktop application. In other words, the desktop application validates the server's identity, i.e., the base unit, employing the server's TLS certificate before establishing a TLS connection with it. We extracted the certificates stored in it with *Binwalk* [44], analyzed their content, and created our certificate chain as close as the real one as we could, to match certificates' size. Subsequently, we patched the desktop application's binaries by replacing the initial certificate root chain with our own and redirected the traffic to our *OpenSSL* proxies which are using certificates signed by our fake root certificate chain. This allows the desktop application to believe that our *OpenSSL* proxy is legitimate and establish a connection with it. We then had access to the control flow's communications in clear text.

4.3 Base unit

To assess the base unit's security, we began with a reconnaissance phase by scanning its open ports. The well-known port scanning tool, *nmap* [38] assisted us during this information gathering stage. We continued with visual reconnaissance to locate the various potential debugging ports as explained in subsection 3.2.1. Then, we proceeded with the administration web interface and the API. Since they both serve the same purpose, to administrate the base unit, with almost the same functionalities, we searched for the same type of vulnerabilities on both.

We started by understanding their features, then we looked for feasible ways to upload files. The idea was to upload a malicious file, execute it, and get an initial access. The uploading endpoints were common to both services. They both possess the possibility to upload wallpapers and XML configuration files. On the former deciphered firmware update, the *PHP* rendering engine was disabled in the wallpapers folder. We can imagine that Barco kept that feature for

[3]Check the references to understand how to design proxies with OpenSSL and FIFOs

their newer firmware versions. We planned to attempt various attacks on the upload functionality to inject a malicious file, such as changing its extension, multiple extensions, modifying the magic numbers, usage of special characters in the file name, path traversal to write the file in a selected directory, and extremely big content to cause overflows. Besides, we tried to read the internal files through *Local File Injections* (LFI) as well. The following attacks are specific to the wallpapers upload feature, code injection via picture metadata and image Trajick [49]. Regarding the XML configuration upload, we also wished to test *XML External Entity* (XXE) and set nonexistent parameters. During our attack simulation, we monitored the base unit's behavior while executing commands on both administration panels. Concerning the API, we particularly checked the HTTP response and we wanted to verify that no possible reflected *Cross-Site Scripting* (XSS) flaws hid in there. For both panels, we also attempted to use their functionalities without being authenticated. For this step, Burp [41] was the predestined tool. Moreover, the administration web interface has many text fields used to configure the base unit, e.g., set its name, SSID, welcome message. We attempted to perform several attacks on this such as buffer overflows, integer overflows or underflows, XSS, *Server-Site Request Forgery* (SSRF).

Since the administration panels were the only software access point, we will continue with physical attacks. During the visual reconnaissance phase, we noticed that a button was inaccessible when the plastic case is closed (in light blue in Figure 3.1). We wanted to understand what it does and scheduled to test buttons combinations at different stages of the booting process and when using the base unit. We noticed some potential debugging ports (in red in Figure 3.1) as well. They will be connected to cables to verify if they are debugging ports or something else. We also planned to extract the eMMC flash memory (in red in Figure 3.2) to isolate the code running on the base unit, however, according to the ClickShare's official documentation, they stored the data encrypted. Hence, the strategy was to understand how the decryption was performed and exploit this knowledge to decipher the base unit's data. We wanted to compare the implementation with the old firmware update we had and observe the differences too.

Finally, the hardware attacks were also considered, because it changed between the ancient ClickShare and the current ones. The modifications occurred right after the WithSecure's report. The previous ones were using a vulnerable SoC whereas the new ones are using *RockChip RK3399* as SoC. This model of processor possesses a specific processor flashing mode, and we thought that it will be accessible through a particular buttons combination. Our strategy was to perform several buttons combinations at different stages of the booting procedure and monitor the various ports we noticed on the base unit, i.e., the potential debug ports (in red in Figure 3.1), USB-A, and USB-C ports.

Chapter 5

Evaluation

This section shows the results, and the rabbit holes we dug into. To maintain the same presentation's organization, this chapter will also be fragmented in three parts, the desktop applications, the base unit and the communications between them.

5.1 Desktop application

This section will foremost develop the reverse engineering, followed by the inter-process communications, and finally, the DLL hijacking.

5.1.1 Reverse engineering

The reverse engineering was harder than we expected, because the desktop application is stripped[1], and sometimes obfuscated, C++ with classes, which implies a lot of indirection while reversing it due to the pointers on the functions. As described in chapter 4, we employed automated tools to assist us. Nevertheless, we obtained different outcomes, yet they were unusable. *BinAbsInspector* and *Joern* never ended their executions. *Ghidra*'s scripts terminated, but their results were worthless, due to their small precision, i.e., we obtained a huge number of false positives.

Finally, using *Ghidra*, we searched for dangerous function calls, e.g., `memcpy`, `sprintf`, or `fread`, where we control the parameters directly. These functions are considered as unsafe as they can be employed to perform buffer overflows, read data from the stack, or crash the process to cause a denial of service. We managed to find a call to `wsprintfW`[2] where the buffer has a

[1]The symbols have been deleted from the binary

[2]A Windows version of `sprintf`.

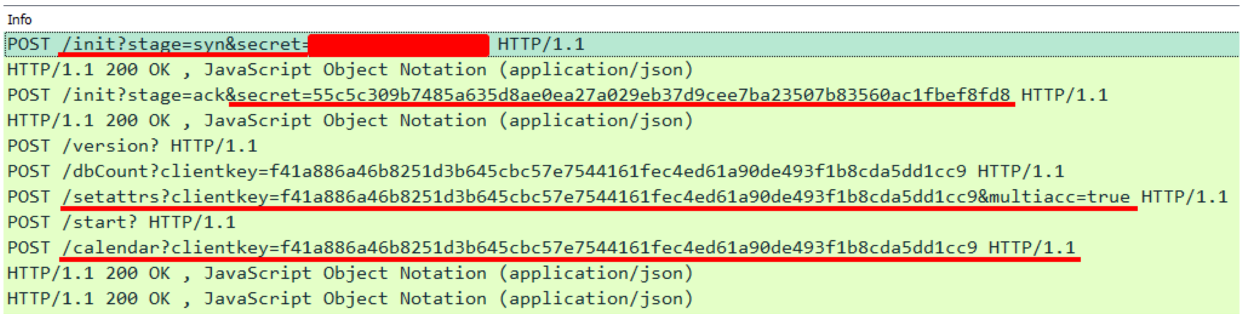
Listing 5.1: Potential buffer overflow in the desktop application
```
// Our 2048-bytes buffer
WCHAR big_buffer_start;
undefined local_806 [2046];

// The strings array
PTR_string_array
    |--> "Function reentry is not allowed:[...]"
    ...
    |--> "Login rejected. User name = %s, Password = %s"
    ...

// Dangerous call to sprintf
wsprintfW(&big_buffer_start,
    (LPCWSTR)(&PTR_string_array)[uVar6 * 2],
    piVar15, piVar13);
```

fixed size of 2048 bytes. Then, we thought that we could control the input buffer, i.e., its size, since the number of copied bytes is linked to the dimension of the input buffer. The input buffer is an element of a string array, i.e. a string, and one of the strings could be imposed by an attacker as it is a format string. Listing 5.1 illustrates this paragraph by showing the situation. Unfortunately, we did not manage to exploit this vulnerability. We followed back the execution chain from the dangerous call, but it is part of a functions list of some C++ classes. Trying to reverse it was a nightmare and it did not end. Recall that the desktop application is obfuscated C++ with classes. Nevertheless, the reverse engineering process taught us a lot about its inner workings and allowed us to verify some behaviors we imagined during this project. For example, the regexes matching on the events collected by `calendarreader32.exe`, or the *Inter-Process Communications* (IPC) endpoints mechanisms.

5.1.2 Inter-Process Communications

The gRPC's protocol buffers extractions worked well (we obtained the protobufs). We tried to modify the delivered packets by changing their sizes, contents, or both, but we did not discover any vulnerability or crash. It discarded the malformed packets.

Concerning the `calendarreader32.exe`'s API, things are more interesting. Through network communications observations and reverse engineering, we originally noticed that the packets were sent over HTTP without encryption (see Figure 5.1). Moreover, the first transmitted packet contains a secret, the latter secrets are derived from it, the primary secret parameter has no constraint on its value, and the `/init` endpoint initializes the API every time it gets hit. Furthermore, the initial secret used by `calendarreader32.exe` is the same for all desktop application installations and is extremely easy to obtain. Thanks to this design, any user that can communicate with `calendarreader32.exe`'s API can initialize the API with the secret they want to, and then make calls to the API without restrictions.



```
Info
POST /init?stage=syn&secret=[redacted] HTTP/1.1
HTTP/1.1 200 OK , JavaScript Object Notation (application/json)
POST /init?stage=ack&secret=55c5c309b7485a635d8ae0ea27a029eb37d9cee7ba23507b83560ac1fbef8fd8 HTTP/1.1
HTTP/1.1 200 OK , JavaScript Object Notation (application/json)
POST /version? HTTP/1.1
POST /dbCount?clientkey=f41a886a46b8251d3b645cbc57e7544161fec4ed61a90de493f1b8cda5dd1cc9 HTTP/1.1
POST /setattrs?clientkey=f41a886a46b8251d3b645cbc57e7544161fec4ed61a90de493f1b8cda5dd1cc9&multiacc=true HTTP/1.1
POST /start? HTTP/1.1
POST /calendar?clientkey=f41a886a46b8251d3b645cbc57e7544161fec4ed61a90de493f1b8cda5dd1cc9 HTTP/1.1
HTTP/1.1 200 OK , JavaScript Object Notation (application/json)
HTTP/1.1 200 OK , JavaScript Object Notation (application/json)
```

Figure 5.1: `calendarreader32.exe`'API simple communications interception with Wireshark. In red, we can see the secrets derived from the one sent as a parameter during the first request. The primary secret has been redacted.

The secret derivation mechanism is also straightforward. It consists of string concatenations and SHA-256 hashing function. We illustrate it with some lines of code in Listing 5.2.

However, the impact is limited, because the endpoints are not interesting except `/quit` which could be employed to perform *Denial of Service* (DoS) attacks, and `/setattrs?freq=xxx` where `xxx` can be replaced by an integer that will represent the number of seconds the application will wait between two events of polling events from the calendars which could be used to cause a DoS if the integer is enormous (it polls the events only every `xxx` seconds, where `xxx` is gigantic). `calendarreader32.exe` is normally running with the same privileges the user has. Hence, even if a malicious actor could execute commands with the privileges of `calendarreader32.exe`, they should not be able to elevate their privileges.

Additionally, `calendarreader32.exe`'s purpose is to retrieve the events stored in the user's calendars that begin in less than 15 minutes or have already started, then filter them to keep only the ones containing a video conference invitation link and supply these results to `clickshare_native.exe` when it asks for it. While we reverse engineered `calendarreader32.exe`, we noticed that the filtering process was implemented with basic *regular expressions* (regexes) matching. The regexes we extracted were too simple and did not provide enough filtering. An attacker can easily design links to deceive the URL parser. We created an example in Figure 5.2. If the program related to the invitation link written in the meeting's body is detected through regexes, then, a joining button appears next to the event. It does not seem to be possible to use another protocol than HTTP(S) as an invitation link to bypass the filtering process. The link is overwritten with `"https://"` at the beginning of the string or it is discarded. However, it could be employed as a phishing vector to mislead users. For example, we created an event observed as a Zoom[3] link displaying a genuine link on the desktop application GUI (write

[3]Video conference application, see https://zoom.us/

Listing 5.2: `calendarreader32.exe`'s API secrets derivation function

```
# Init connection; secret1 = toto
curl -vi -X POST \
        -d 'stage=syn&secret=toto' \
        localhost:49200/init;

# ==> server's response1 = SHA256(secret1 + "servicesynack")
# secret2 = SHA256(response1 + "clientack")
curl -vi -X POST \
        -d 'stage=ack&secret=
        9972b7755c85e843d1ea00854d27b415
        f0b9f9b42bf1b82648de24225b223ef7' \
        localhost:49200/init;

# ==> server's response2 = SHA256(secret2 + "servicegenapikey")
# clientkey = SHA256(response2 + "ok")
# Set event polling frequency (e.g. freq=1 --> poll every seconds)
curl -vi -X POST \
        -d 'clientkey=
        5c6887f1c44aa86a1de64a24e5c9c1af
        9cb117c7137bd0dcf5a38ea2eda48b46
        &freq=1' \
        localhost:49200/setattrs;

# Get the events containing a video conference
# link and starting in less than 15 minutes
curl -vi -X POST \
        -d 'clientkey=
        5c6887f1c44aa86a1de64a24e5c9c1af
        9cb117c7137bd0dcf5a38ea2eda48b46' \
        localhost:49200/calendar;
```

something legitimate as the event's location), but redirect the user to our malicious website, i.e., `https://attacker.zoom.scrt.ch/j/malicious` (put this in the event's body). This technique also works with the other video conference applications detected by the desktop application, e.g., Microsoft Teams, Meet Lync, Webex.

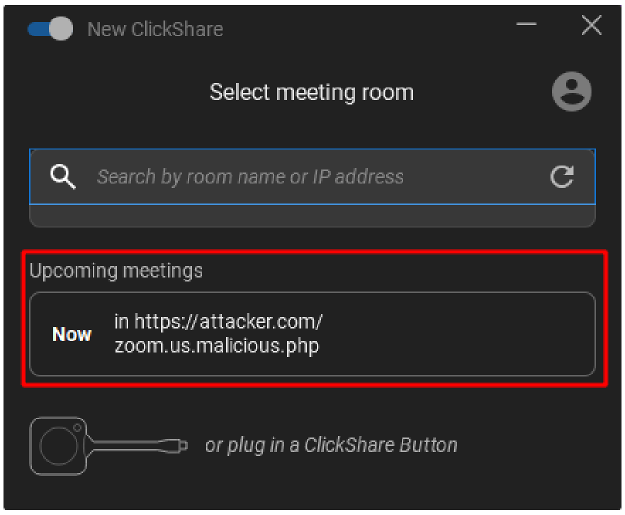


Figure 5.2: `calendarreader32.exe`'s URL filtering bypass

5.1.3 DLL hijacking

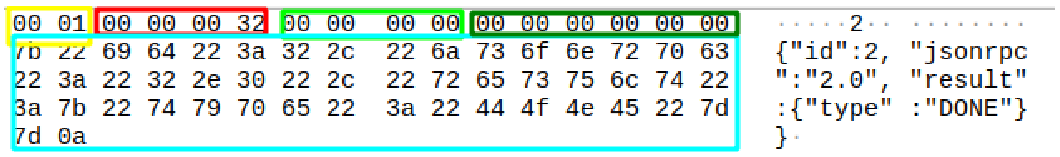
DLL hijacking is deceiving a trusted application into loading an arbitrary DLL [30] [35]. In our context, with *Process Monitor* (ProcMon), we noted that the desktop application tries to load some DLLs without finding them directly. We described the issue in scenario 2 in [35], i.e., the DLLs exist, but are not in the folder that Windows searches first. Thus, the desktop application should be vulnerable to *DLL sideloading*. We experimented this attack by creating malicious DLL with *msfVenom* [42] without getting any success. The desktop application does not even load our nefarious DLL. Nevertheless, its impact would be limited since no executable is running with different rights from the user ones. Thus, it could not normally be employed as a local privileges escalation.

5.2 Communications

Regarding the communications between the desktop application and the base unit (see Figure 3.3), we already mentioned that the deployed ciphers were not conventional, and that we had to bypass the server's identity verification before having access to the transmissions in plain text (see section 4.2). The underlying protocol uses JSON-RPC[4] encapsulated in a custom wrapper.

[4]Another RPC variant

We depicted the packet structure in Figure 5.3. With this knowledge, and after cleaning up the collection of captured packages, we replayed the packets with a simple python script by establishing a TLS connection to one of the communication sides to demonstrate the replay feasibility. It worked, then we should be able to recreate a desktop client based on these exchanges. We also tried to modify the packets' header (yellow, red and green parts in Figure 5.3), their content (blue in Figure 5.3), or both, to produce errors or crashes with some buffer overflow for example. Unfortunately, the packets we crafted in this manner were discarded. In addition, we looked for insecure callable methods through the JSON-RPC protocol, but we did not discover anything interesting.



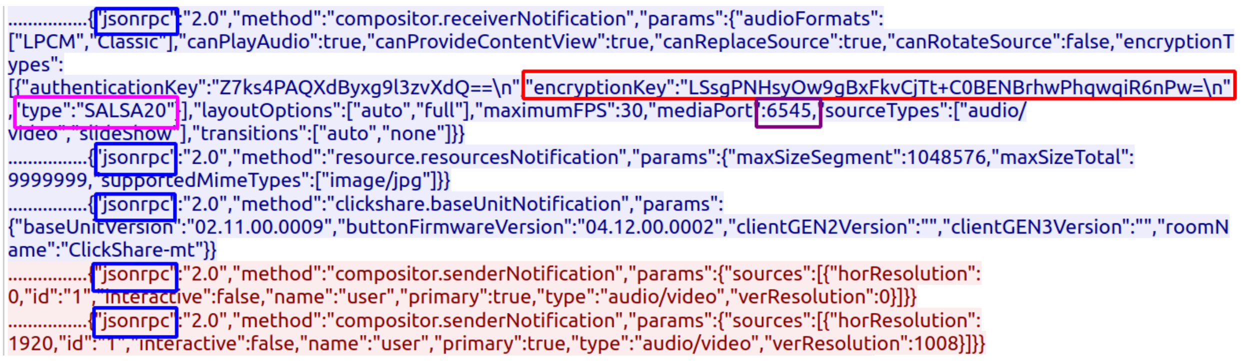
```
00 01 00 00 00 32 00 00  00 00 00 00 00 00 00 00   .....2.. ........
7b 22 69 64 22 3a 32 2c  22 6a 73 6f 6e 72 70 63   {"id":2, "jsonrpc
22 3a 22 32 2e 30 22 2c  22 72 65 73 75 6c 74 22   ":"2.0", "result"
3a 7b 22 74 79 70 65 22  3a 22 44 4f 4e 45 22 7d   :{"type" :"DONE"}
7d 0a                                              }.
```

Figure 5.3: JSON-RPC packets, sent between desktop application and base unit, chunks splitting. yellow: version; red: blue packet's size (json-rpc); light green: size of the picture (here: no picture sent $\implies$ size is 0); dark green: unknown (often zeros); blue: JSON-RPC data. It may contain pictures after the blue packet, but it is optional.

Notwithstanding, we delved into the JSON-RPC content. We noted in the exchange that the video stream encryption key is transmitted without key negotiation algorithms, but as plain text (as long as the TLS can be deciphered). Figure 5.4 illustrates it. WithSecure already reported the same issue under CVE-2019-18833 [7][51]. The only difference is the exploitation path. WithSecure went through a more straightforward one, since they managed to get a test certificate (and its private key) signed by the real root certificate chain used in production (CVE-2019-18831 [6]) [51]. Thus, the MitM attack was easier for them to conduct, but the results are identical. As long as a malicious actor can perform a MitM and decrypt the TLS communications, he can intercept the video stream encryption key and decipher the shared content. Since WithSecure already reported the same vulnerability and proved that the the video stream decryption was possible with this key [7][51], we did not implement any *Proof of Concept* (PoC), and we let the reader create their own. Knowing how to perform the attack to obtain the decryption key and that the video stream uses SALSA20 as an encryption algorithm eases the work. This weakness has been fixed by the security team of ClickShare by deleting the test certificate only. Thus, the initial foothold has been corrected, but the issue is still present and feasible. It is just harder to exploit since we need to patch the binary to conduct a MitM attack (as described in section 4.2), which implies to break the executable signature.

5.3 Base unit

This part will start by explaining the administration web interface and API results, followed by the hardware attacks, discussing both the potential debugging ports and the eMMC extraction



```
................{"jsonrpc":"2.0","method":"compositor.receiverNotification","params":{"audioFormats":
["LPCM","Classic"],"canPlayAudio":true,"canProvideContentView":true,"canReplaceSource":true,"canRotateSource":false,"encryptionT
ypes":
[{"authenticationKey":"Z7ks4PAQXdByxg9l3zvXdQ==\n","encryptionKey":"LSsgPNHsyOw9gBxFkvCjTt+C0BENBrhwPhqwqiR6nPw=\n"
,"type":"SALSA20"}],"layoutOptions":["auto","full"],"maximumFPS":30,"mediaPort":6545,"sourceTypes":["audio/
video","slideShow"],"transitions":["auto","none"]}}
................{"jsonrpc":"2.0","method":"resource.resourcesNotification","params":{"maxSizeSegment":1048576,"maxSizeTotal":
9999999,"supportedMimeTypes":["image/jpg"]}}
................{"jsonrpc":"2.0","method":"clickshare.baseUnitNotification","params":
{"baseUnitVersion":"02.11.00.0009","buttonFirmwareVersion":"04.12.00.0002","clientGEN2Version":"","clientGEN3Version":"","roomN
ame":"ClickShare-mt"}}
................{"jsonrpc":"2.0","method":"compositor.senderNotification","params":{"sources":[{"horResolution":
0,"id":"1","interactive":false,"name":"user","primary":true,"type":"audio/video","verResolution":0}]}}
................{"jsonrpc":"2.0","method":"compositor.senderNotification","params":{"sources":[{"horResolution":
1920,"id":"1","interactive":false,"name":"user","primary":true,"type":"audio/video","verResolution":1008}]}}
```

Figure 5.4: Deciphered JSON-RPC packets transmitted from the base unit to the desktop application. Red: video stream encryption key sent without key negotiation algorithms; blue: json-rpc protocol; pink: video stream encryption algorithm; purple: video stream port.

and analysis.

Firstly, we wanted to specify that the initial *nmap* scan of all TCP ports did not reveal any special opened ports different from the ones announced by ClickShare [23].

5.3.1 Administration panels

We began with a simple test. The API authenticates the users with HTTP basic authentication, but the administration web interface use cookies. On both services, we tried to use the application without being authenticated, and with manually crafted authentication. However, our attempts have been unsuccessful.

We already explained in section 3.2.2 that it was allowed to upload files via the administration web interface and the API (listening on port 4003). The malicious file injections were not helpful. On both wallpapers upload endpoints, it was not conceivable to upload a file type different from `*.jpg` or `*.png`, because its header and filename are verified. Nonetheless, we managed to inject a file with an arbitrary content through the API's "XML configuration upload" endpoint, because its header is not validated, but its filename is. With this technique it is possible to upload a file with an arbitrary base64 encoded content into the wallpapers' folder, but we cannot choose its extension. Obviously, we endeavored to circumvent these restrictions by modifying the filename, e.g., "`toto.png`" became "`toto.php.png`", "`toto.png%00.php`", or "`toto.png?shell.php`", yet the applications blocked them. We also tried to write the file in a different directory with a path traversal, with special symbols too, but the filename is sanitized and unconventional characters are replaced by "_". However, we observed on the old decrypted firmware update we had that ClickShare's teams deactivated the *PHP* rendering in the wallpapers' folder. Then even if we managed to bypass the protections and upload a PHP file, we will not be able to execute it as a PHP script file.

Moreover, the administration's panels possess text fields that might be filled to configure the base unit. We experimented various types of attacks on them, such as *XML eXternal Entity* (XXE), *Cross-Site Scripting* (XSS), *Cross-Site Request Forgery* (CSRF), *Local File Injection* (LFI), buffer overflows, integer over/underflows, path traversal, and special characters injection. We even used payload lists such as the one proposed by Burp Intruder and its active scans. Unfortunately, every attack failed.

Furthermore, we noticed that the API responses' content-type is (almost) always *application/json* (except two that are *text/xml*). It prevents the possibility of sending to a targeted user an XSS payload firing a CSRF payload that could impact the base unit through its administration panels.

Besides, both administration panels propose a way to download the base unit's logs. We performed several tasks on the base unit and downloaded the logs. Then, we went through every file. Regrettably, we did not observe any sensitive information leaking in the downloadable log files.

5.3.2 Hardware attacks

The hardware attacks will be presented in two stages. Firstly, the debugging ports' results, followed by the findings we extracted from the eMMC, and their explorations.

Debugging ports

As mentioned earlier, we soldered cables on the potential debugging ports (see Figure 3.1) and connected them to a logical analyzer to capture signals if any. We attempted to obtain stimuli by pushing, and sometimes keeping pressed, the three buttons present on the base unit's board (yellow and blue in Figure 3.1) during and after the booting process. Unfortunately, our efforts have been unsuccessful. According to the base unit's official documentation, the JTAG is permanently disabled on the production devices by blowing up the *eFuse*[5], and the serial port is disabled in the boot loader on the production devices, which explains our failed attempts.

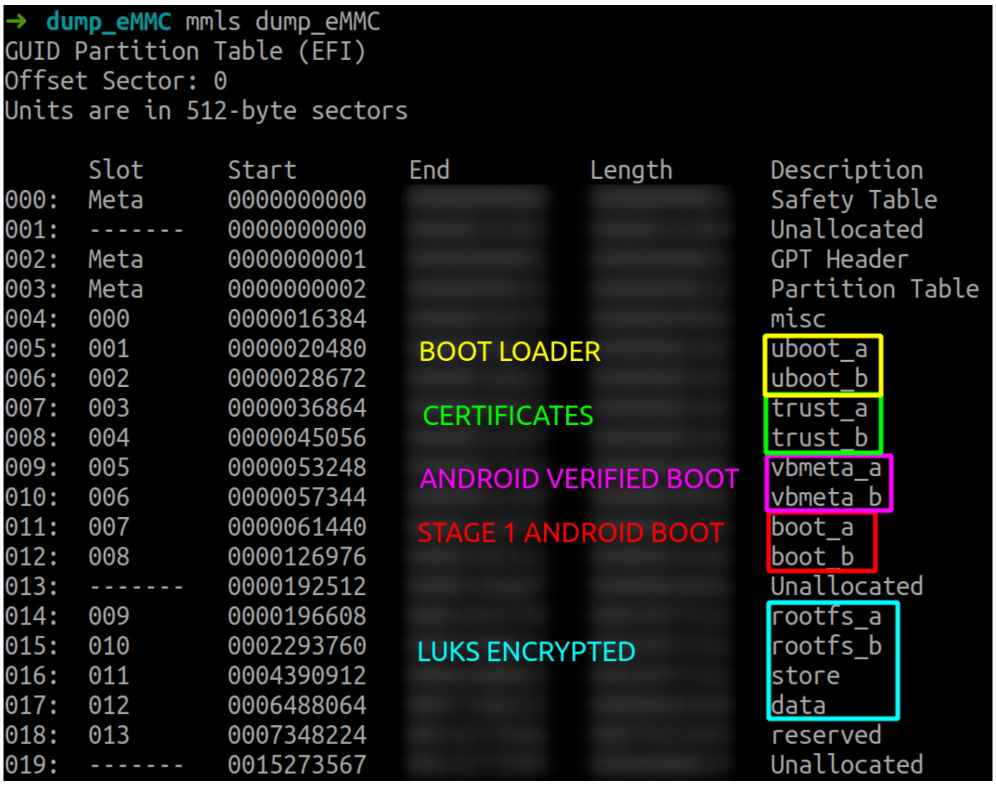
We presented the SoC as a RockChip RK3399. This chip should possess a flashing mode. We will see in section 5.3.2 that the eMMC contains some Android boot image, and we know that Android may offers an Android recovery mode [11]. With this knowledge, we tried to find both special modes through button push combinations at various boot stages. We consider the button which is inaccessible when the plastic case is closed (blue in Figure 3.1), pushed it, and held it at different booting phases. Disappointingly, the only behavior we observed, was that it stopped the booting process at the step it was when we pushed the button. No signal was captured on the potential debugging ports or on the USB ports (both A and C types). We also attempted to

[5]A microscopic fuse located directly in the SoC, intended for being blown up

connect to Android using *Android Debug Bridge* (ADB) [10], but the results were negative. In consequence, we did not discover the flashing or recovery modes.

eMMC extraction and analysis

On Figure 3.2 we can observe the eMMC (flash memory) the *Operation System* (OS). The eMMC is pretty small (11.5 x 13 mm) with 153 soldering contact points, then we required assistance to get the eMMC's content. With the expertise and material of Mr. Nicolas Oberli, we desoldered the eMMC and extracted its data using a compatible eMMC reader[6]. Figure 5.5 illustrates the 20 partitions it includes and the interesting partitions are highlighted. Notice that some partitions end with `_a` or `_b`. This technique is called the "*seamless updates*". It ensures that a viable booting system remains on the disk during the updates and guarantees a fail-safe design [9]. To keep it simple, at each update, it copies the data on the inactive partitions, i.e., `a` or `b` group, and tries to boot multiple times to verify that everything is working correctly before flipping a bit telling which partition is active. If something went wrong during the update, then the bit is not flipped and the update is not applied.



```
→ dump_eMMC mmls dump_eMMC
GUID Partition Table (EFI)
Offset Sector: 0
Units are in 512-byte sectors

      Slot      Start        End          Length       Description
000:  Meta      0000000000                             Safety Table
001:  -------   0000000000                             Unallocated
002:  Meta      0000000001                             GPT Header
003:  Meta      0000000002                             Partition Table
004:  000       0000016384                             misc
005:  001       0000020480                             uboot_a
006:  002       0000028672                             uboot_b
007:  003       0000036864                             trust_a
008:  004       0000045056                             trust_b
009:  005       0000053248                             vbmeta_a
010:  006       0000057344                             vbmeta_b
011:  007       0000061440                             boot_a
012:  008       0000126976                             boot_b
013:  -------   0000192512                             Unallocated
014:  009       0000196608                             rootfs_a
015:  010       0002293760                             rootfs_b
016:  011       0004390912                             store
017:  012       0006488064                             data
018:  013       0007348224                             reserved
019:  -------   0015273567                             Unallocated
```

Figure 5.5: List of eMMC partitions. Our targets were the light-blue partitions, and they are decrypted by the red ones. (Blurred only for readability.)

[6]See https://fr.aliexpress.com/item/1005003391265061.html

The `uboot_[ab]` partitions include the boot loader, `trust_[ab]` contain the certificates used by Android verified boot (`vbmeta_[ab]`) to confirm the system integrity and authentication. Recall that the serial port was disabled in the boot loader. Since Android verified boot checks the boot loader integrity, it is unfortunately impossible to alter it to reactivate the serial port before flashing the eMMC and soldering it back to get a serial access to the base unit.

Then, we will continue with the red and light-blue partitions in Figure 5.5, because we cannot modify the others without being noticed by the Android verified boot. We named partitions `boot_[ab]` as "*Stage 1 Android boot*", given that they are composed of Android boot images and that, according to their content that we will detail, they are responsible for the decryption and mounting of the LUKS encrypted partitions (blue in Figure 5.5). The Android boot image of `boot_[ab]` partitions consist in the kernel, a RAM disk, and a second stage. The interesting part here is the RAM disk since it contains an easily extractable UNIX file system.

The file system has a common structure, but around ten scripts, listed in Table 5.1, have been added to it by the ClickShare's teams (according to the comments in them). As we discussed before, they are responsible to check the updates availability and installation, and to decrypt and mount the LUKS encrypted partitions including the base unit's logical core. We followed the path from the file `/init` to the LUKS decryption to understand the whole process. We illustrated it in Figure 5.6 as it can be separated in three parts.

Firstly, we need to retrieve the key from the eMMC flash memory. Its size depends on whether the hardware encryption is activated, which is enabled by default unless the file `/AW0076-366` exists, but it is not the case in our situation. Hence the 32-byte key is stored encrypted on the eMMC and we get it with a simple `dd` call. Secondly, if the hardware encryption is on, then we call a *Trusted Execution Environment* (TEE) to decrypt it. This is confirmed by the base unit official user guide [20]. For this reason, we did not manage to decrypt the key since it is hardware based. Finally, `cryptsetup` employs the decrypted key to decrypt the LUKS partitions. However, the base unit uses a special option of `cryptsetup` that is not documented (even in the documentation of the used version of `cryptsetup`), i.e., `-iteration-cipher` [36]. We assumed that it was a customized variant of `cryptsetup`. We retrieved the option's description by reversing the uncommon `cryptsetup` and it is used to pass a hardware driver that will be employed during the decryption process. Nevertheless, the `/sys/` folder is empty in our file system and we did not recover the file mentioned earlier on the eMMC. These design choices and the driver issue prevented us from decrypting the LUKS key and decrypt the LUKS partitions content.

Thanks to the seamless updates feature, we had access to two versions of the stage one file system. Nevertheless, the differences were only linting and little details in code without changing the output and without introducing or fixing any vulnerabilities. We think that most of the alterations occurred in the LUKS encrypted partitions.

We observed the `root` password hash of the stage one partitions. For completeness, we display them in Listing 5.3. We cracked them fast since the combinations were twice `root:letmein`, but with distinct salts. WithSecure already reported this issue as CVE-2019-18828 [3], but for the

`/newroot/*`	(empty)
`/init`	
`/builddate.txt`	
`/etc/default/*`	(contains some default base unit settings)
`/etc/init.d/*`	
`/etc/public_key.gpg`	(updates signature verification key)
`/sbin/cross_platform_functions.sh`	
`/sbin/mkhomedir_helper`	
`/sbin/secure-system-update`	
`/sbin/w3duff_platform_functions.sh`	
`/usr/bin/hardware_hash_app`	
`/usr/lib/os-release`	(basic information about current version)
`/usr/sbin/cryptsetup`	(custom version)
`/var/www/*`	(empty)

Table 5.1: The files added by ClickShare's team in "stage 1" (`boot_[ab]`) partitions responsible for decrypting and mounting the LUKS encrypted partitions of the base unit.

Listing 5.3: `root` passwords extracted from the eMMC's stage 1 partitions (`/etc/shadow`)

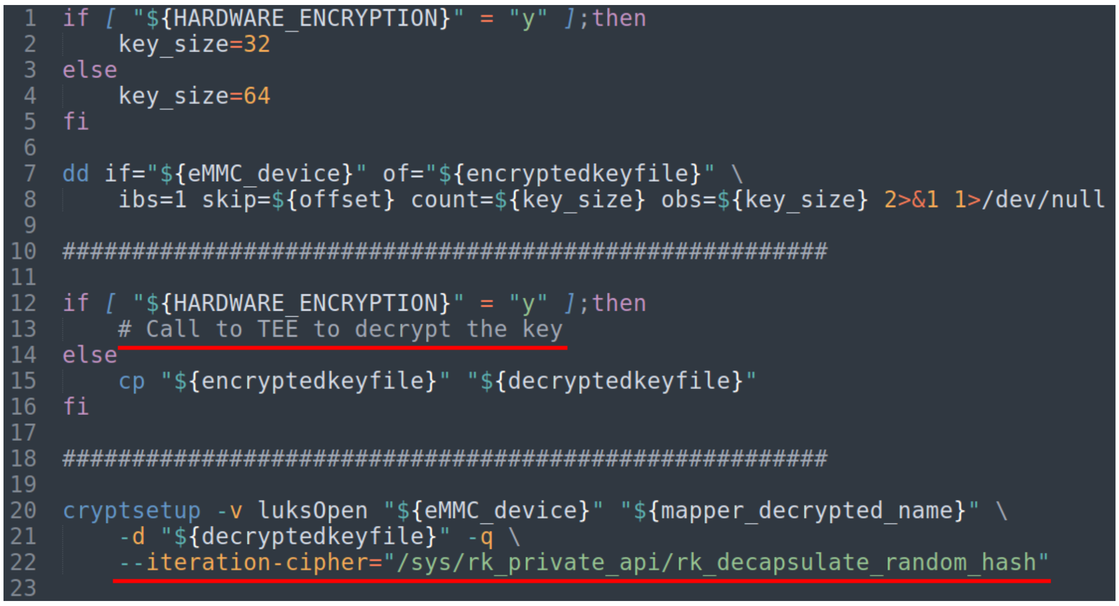
```
# /etc/shadow boot_a --> root:letmein
root:$6$ANR1AiGU$
NMRDm/YFpgrZPoyMEsVweHt59vbzGoPqdz7rquN60kx
XsnJnbzX0fekjhiiohbr2TEYmgQ41/LyMXgeuofkSF0:::::::

# /etc/shadow boot_b --> root:letmein
root:$6$mD3rAwj0K$
5FwH8MPH3L1uXE/12yENvW7Y6n0scGzaKru5m3RH82U
.zjNDJCEHZeAtRD7UgDTGWBDiJYb0WIeCjrRnUn8xZ1:::::::
```

buttons [51]. Apparently, ClickShare modified the buttons' root password [21], yet not in the base unit while reusing the same easy one, at least for the stage 1.

Furthermore, the firmware updates are signed with a *GNU Privacy Guard* (GPG) key which avoids the possibility of crafting malicious firmware signature. The key is listed in the files added by ClickShare in the stage one partitions in Table 5.1.

Other attempts

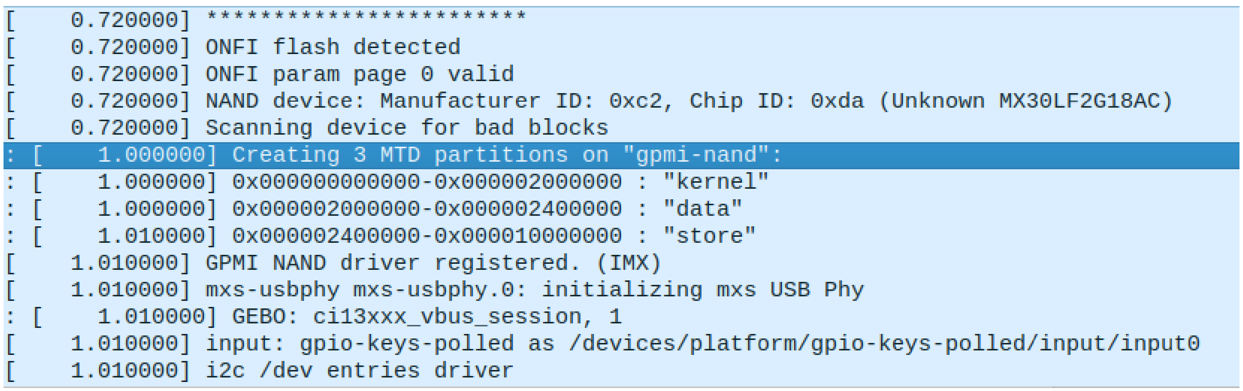
In addition to these attacks, we thought about *Direct Memory Access* (DMA) [52] through the Wi-Fi card slot. Unfortunately, according to the RockChip RK3399 data sheet, in chapter 16.3.3, the SoC uses ARM Trust Zone which protects the *DMA Controller* (DMAC) [45]. Therefore, it should not be possible to perform DMA attacks.



```
if [ "${HARDWARE_ENCRYPTION}" = "y" ];then
    key_size=32
else
    key_size=64
fi

dd if="${eMMC_device}" of="${encryptedkeyfile}" \
    ibs=1 skip=${offset} count=${key_size} obs=${key_size} 2>&1 1>/dev/null

#######################################################

if [ "${HARDWARE_ENCRYPTION}" = "y" ];then
    # Call to TEE to decrypt the key
else
    cp "${encryptedkeyfile}" "${decryptedkeyfile}"
fi

#######################################################

cryptsetup -v luksOpen "${eMMC_device}" "${mapper_decrypted_name}" \
    -d "${decryptedkeyfile}" -q \
    --iteration-cipher="/sys/rk_private_api/rk_decapsulate_random_hash"

```

Figure 5.6: Some simplified lines of code to understand the LUKS partitions decryption process. Notice TEE usage and custom `cryptsetup` with hardware driver option.

During this project, we only had access to an old button (R9861500D01), so we decided that it was not relevant to do the same job as WithSecure did earlier, but we verified that the system logs were sent in clear text over Wi-Fi to the base unit as reported by WithSecure (BCSD-FSC-RND-F011) [51]. Since our button was not compatible with the recent base unit we had, then we created a fake Wi-Fi AP with equivalent settings as the one the button tries to connect to. The button established a connection with our fake Wi-Fi AP and we observed its system logs (see Figure 5.7). We did not continue in this path since WithSecure already did the work two and a half years ago on the same kind of old buttons and we did not have access to a modern one for this project.



```
[    0.720000] ************************
[    0.720000] ONFI flash detected
[    0.720000] ONFI param page 0 valid
[    0.720000] NAND device: Manufacturer ID: 0xc2, Chip ID: 0xda (Unknown MX30LF2G18AC)
[    0.720000] Scanning device for bad blocks
: [    1.000000] Creating 3 MTD partitions on "gpmi-nand":
: [    1.000000] 0x000000000000-0x000002000000 : "kernel"
: [    1.000000] 0x000002000000-0x000002400000 : "data"
: [    1.010000] 0x000002400000-0x000010000000 : "store"
[    1.010000] GPMI NAND driver registered. (IMX)
[    1.010000] mxs-usbphy mxs-usbphy.0: initializing mxs USB Phy
: [    1.010000] GEBO: ci13xxx_vbus_session, 1
[    1.010000] input: gpio-keys-polled as /devices/platform/gpio-keys-polled/input/input0
[    1.010000] i2c /dev entries driver
```

Figure 5.7: Capture of an old button syslogs in clear text through a fake Wi-Fi AP. Reported by WithSecure under BCSD-FSC-RND-F011 [51]

Chapter 6

Related Work

In this chapter, we will expose the effort that has already been completed up to the present day. We will firstly cover the labor of WithSecure [51], which was previously called F-Secure (they changed their name recently). Then, we will continue with the ClickShare's published security papers addressing several aspects of their devices' security. However, since ClickShare's security has been assessed by the famous WithSecure team, there is few research about these devices, because they already pointed out a lot of weaknesses. Thus, the ClickShare's security advisories will be our last documents in this chapter.

6.1 WithSecure

WithSecure is an international cyber security company. In 2019, under their previous name, F-Secure, they evaluated ClickShare's device security which resulted in fourteen reported vulnerabilities. They assessed the security in the same manner as we did. However, they discovered a base unit internal access allowing them to get a lot more information about the inner workings. This knowledge allowed them to attempt a large panel of attacks while monitoring the produced errors and use them to debug their tries.

For instance, they discovered two exploitation paths to obtain root shell accesses to a button and to a base unit. Regarding the base unit, they noticed that the SoC was sensitive to two old common vulnerabilities, i.e., CVE-2017-7936 [32] and CVE-2017-7932 [31][51]. These vulnerabilities address the High Assurance Boot functionality[1]. They allow a malicious actor to bypass this security, compromise the device by modifying the eMMC's content, and activate a root access through the serial port. This attack was possible in 2019, due to the hardware used to build the base unit.

[1]A secure boot feature

Concerning the buttons, the exploitation path was slightly longer. With the base unit access, they gathered more information about the interactions between buttons and a base unit, such as the code responsible for the button update while the button is connected to the base unit through USB. In other words, they retrieved the *USB dongle bridge*'s code. Then, WithSecure teams noticed that some commands transmitted over this canal were vulnerable to command injections (CVE-2019-18830 [5]) [51]. A group of sensitive commands sent via this channel were signed using a certificate. However, the mutual certificate chain verification was incorrect (CVE-2019-18826 [1]) [51] which gave to WithSecure teams the possibility to obtain a shell through the serial port. The last step was to elevate the shell privileges. We think that they did it by brute forcing the `root` password, since the file `/etc/shadow`[2] is not readable by users (except `root`) by default. The buttons `root` password was `letmein` (CVE-2019-18828) [51].

WithSecure discovered another interesting exploitation path that can compromise a button user. They combined a button file system mounting options poor configuration exposing, in read-write access, a partition of the button's file system through USB, with a DLL-sideloading attack taking place in the desktop application offered by the button via USB. With this attack, they obtained an entry on the computer of users who run the application present on the button containing the malicious DLL.

Concerning the vulnerabilities that we have also identified, we already discussed the `root:letmein` weak password. However, we noticed as well the lack of key negotiation algorithms used to encrypt the video stream while being delivered via the control channel between the base unit and the desktop application (see Figure 3.3). We will reference the ClickShare's solution in the next section, but for impatient readers, ClickShare only corrected the entry point of the attack suggested by WithSecure. We also discussed about the buttons' system logs sent in clear through Wi-Fi connection (BCSD-FSC-RND-F011) [51].

6.2 ClickShare's work

Besides the safety patches they provided to correct the weaknesses reported by WithSecure, ClickShare also seems to have increased their security awareness and published articles on their website assessing common vulnerabilities such as Dirty Pipe [14], Meltdown and Spectre [16], Log4Shell [15], and RIPPLE20 [17].

Moreover, WithSecure greatly increased the security of the ClickShare's devices since the WithSecure's report. We will go through the main updates they provided using their security advisory[21].

Regarding the buttons, they addressed the hardware vulnerability by developing fresh buttons with a new SoC, disabled the JTAG permanently, deleted the command injection via the

[2]Contains the passwords hashes of the operation system users

USB dongle bridge, removed the test certificate from the production devices, corrected the mutual certificate verification process during the button-base unit authentication through USB, adapted the `root` password of the buttons for a 26-symbol length one, and finally, they prevented DLL-sideloading. However, these patches did not fix the missing key negotiation algorithm to exchange the video stream encryption key as we discussed in section 5.2.

Concerning the base unit, they only needed to modify SoC model, which has been done when the WithSecure's report became public. They are now using RockChip RK3399 SoC, as presented in subsection 3.2.1.

Chapter 7

Future Work

Among the attacks we thought about, there are some in which we failed, some that we did not attempt because of lack of expertise or material, and some others that we could not try due to absence of entry points in the ClickShare devices. Figure 7.1 illustrates some locations we did not try to attack and that can be considered as future work.

First, the buttons have not been attacked due to lack of material, i.e., we only had access to an old button (R9861500D01) and not a new one. The security of the cloud infrastructure connections from the base unit has not been assessed, because of the limited time we had for this project. Concerning the updates servers, we assumed that even if a malicious actor could impersonate them (which should be hard, since the base unit should verify the server identity with the TLS certificate), then the attacker would still have to forge a correct GPG signature and a valid encryption for the firmware update that he will propose. Thus, we focused on the remainder of the attack surface as described in this document. Finally, the connection to the authentication beta server has not been investigated since it seemed less engaging than the rest, because there was no restriction to create an account having an access to the beta version of the desktop application.

Besides, we did not try the CVE-2020-14382 [43] impacting `cryptsetup` to decrypt the LUKS partitions, because it would have required a lot of soldering and desoldering of the eMMC. Since we performed this step with the help (in expertise and material) of Mr. Nicolas Oberli and Mr. Karim Sudeki, it would not have been possible to accomplish this process multiple times within the project deadline.

We considered the option to fuzz the desktop application in several different ways (described in chapter 4), but some were displeasing, and others did not work. Someone with a better background in fuzzing than us could obtain worthier results. The main issue we faced was the instrumentation of stripped binaries.

Since the buttons must be paired to the base unit through USB before being employed, it

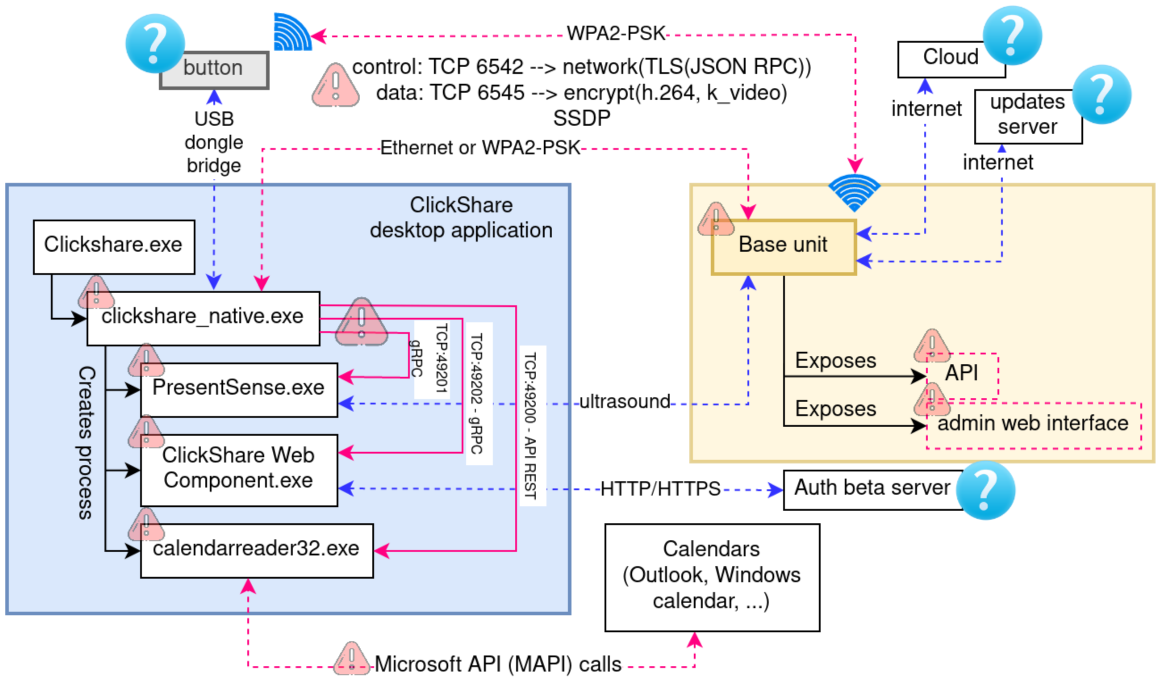


Figure 7.1: The blue question marks depict the locations we did not test and believed that attempted attacks could be interesting. The transparent warning signs and pink links represent the ones we attacked in this document.

should be possible to connect a FaceDancer21 [48] to the base unit to emulate a button and leak information by interacting with the base unit. However, we need the code running either on the base unit or the buttons to observe the endpoints or callable functions.

We could also fuzz the base unit's video library to find exploitable issues in it. That way, a malicious actor could send a malicious video stream to the base unit to execute commands, for example, and install a backdoor allowing him to get access to the shared content. Nevertheless, it requires the base unit's video library to test this attack with fuzzers. Unfortunately, we did not have access to it during this project, because we did not manage to obtain an access to base unit's inner workings.

Finally, side-channel attacks should also be considered. We had neither the equipment nor the experience, to carry out this type of attack. Nonetheless, this may lead a malicious actor to discover hidden secret keys, for example.

Chapter 8

Conclusion

During this project, we learned several interesting attack methods and discovered many tools for assessing the security of physical devices. Besides, we became aware of a good deal of actual attacking techniques.

Even if no amazing new attack has been discovered during this project, we still managed to recover the `root` password of the first stage initiating the base unit. This issue has already been reported by WithSecure [51], yet for buttons. In addition, we understood how to implement a DoS attack on a partition of the desktop application as soon as we can send traffic to the targeted user's localhost interface. We also discovered potential buffer overflows in the desktop application, but did not succeed in their exploitation. Moreover, we exposed a new phishing vector operable through a simple meeting invitation email. Regarding the communications between the desktop application and the base unit, WithSecure already reported the lack of use of key negotiation algorithms to exchange the video stream encryption key (sent over the control flow, see Figure 3.3, to encrypt the data flow) and this weakness is still present despite that the TLS protocol (employed to encrypt the control flow) should be sufficient to avoid Man-in-the-Middle attacks. Finally, even if we managed to extract and analyze a portion of the eMMC's content, the most interesting partitions were encrypted with LUKS. Brute forcing the encryption could not be considered due to the immensely vast number of password possibilities, i.e., 64 bytes, implying $2^{64*8} \cong 1.34 \cdot 10^{134}$ possibilities.

In conclusion, even though Barco greatly reinforced the security of their devices and strengthened them well against black box attacks, there is always room for improvement. Unfortunately, we did not discover new sensational attacks on ClickShare devices.

Bibliography

[1] WithSecure. *CVE-2019-18826*. https://nvd.nist.gov/vuln/detail/CVE-2019-18826. 2019.

[2] WithSecure. *CVE-2019-18827*. https://nvd.nist.gov/vuln/detail/CVE-2019-18827. 2019.

[3] WithSecure. *CVE-2019-18828*. https://nvd.nist.gov/vuln/detail/CVE-2019-18828. 2019.

[4] WithSecure. *CVE-2019-18829*. https://nvd.nist.gov/vuln/detail/CVE-2019-18829. 2019.

[5] WithSecure. *CVE-2019-18830*. https://nvd.nist.gov/vuln/detail/CVE-2019-18830. 2019.

[6] WithSecure. *CVE-2019-18831*. https://nvd.nist.gov/vuln/detail/CVE-2019-18831. 2019.

[7] WithSecure. *CVE-2019-18833*. https://nvd.nist.gov/vuln/detail/CVE-2019-18833. 2019.

[8] US National Security Agency. *Ghidra*. https://ghidra-sre.org/. Mar. 2019.

[9] Android. *A/B (Seamless) System Updates*. https://source.android.com/devices/tech/ota/ab. 2022.

[10] Android. *ADB*. https://developer.android.com/studio/command-line/adb. 2022.

[11] Android. *Recovery Images*. https://source.android.com/docs/core/bootloader/recovery-images. Aug. 3, 2022.

[12] APIMonitor. *API Monitor*. https://www.apimonitor.com/.

[13] Barco. *Barco*. https://www.barco.com/en/. 2022.

[14] Barco. *Dirty Pipe fix*. https://www.barco.com/en/clickshare/support/clickshare-cse-200/knowledge-base/5640-dirty-pipe-vulnerability-impacts-on-barco-products. June 14, 2022.

[15] Barco. *Log4Shell fix*. https://www.barco.com/en/clickshare/support/clickshare-cse-200-plus/knowledge-base/5656-is-there-any-impact-for-barco-products-on-the-log4shell-vulnerability-cve202144228. July 7, 2022.

[16] Barco. *Meltdown and Spectre fix*. https://www.barco.com/en/clickshare/support/clickshare-cse-200/knowledge-base/4051-is-clickshare-affected-by-the-meltdown-and-spectre-vulnerabilities. June 14, 2022.

[17] Barco. *RIPPLE20 vulnerabities evaluation*. https://www.barco.com/en/clickshare/support/clickshare-cse-200-plus/knowledge-base/3480-ripple-20-vulnerabilities-on-barco-products. June 14, 2022.

[18] Bettercap. *Bettercap*. https://www.bettercap.org/. 2022.

[19] bloomrpc. *BloomRPC*. https://github.com/bloomrpc/bloomrpc. Mar. 2022.

[20] ClickShare. *C-5 User guide*. https://www.barco.com/en/clickshare/support/clickshare-c-5/docs. Mar. 2022.

[21] ClickShare. *ClickShare Security Advisory*. https://www.bis.be/media/downloads/whitepaper/clickshare_security_advisory.pdf. Dec. 17, 2019.

[22] ClickShare. *Security Whitepaper, ClickShare Conference and ClickShare Present*. https://www.barco.com/en/clickshare/support/docs/TDE10355. May 28, 2021.

[23] ClickShare. *Which ports does the ClickShare button and base unit use?* https://www.barco.com/de/clickshare/support/clickshare-cse-200/knowledge-base/4130-which-ports-does-the-clickshare-button-and-base-unit-use. 2022.

[24] CloudFlare. *inside-mirai-the-infamous-iot-botnet-a-retrospective-analysis*. https://blog.cloudflare.com/inside-mirai-the-infamous-iot-botnet-a-retrospective-analysis/. Dec. 14, 2017.

[25] Gerald Combs and the community. *Wireshark*. https://www.wireshark.org/. 2022.

[26] Jyoti Deogirikar and Amarsinh Vidhate. "Security attacks in IoT: A survey". In: *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*. 2017, pp. 32–37.

[27] Ettercap-Project. *Ettercap*. https://www.ettercap-project.org/. 2022.

[28] David Fabian Niko. *Joern*. https://joern.io/. 2022.

[29] Google. *gRPC*. https://grpc.io. 2022.

[30] HackTricks. *Dll Hijacking*. https://book.hacktricks.xyz/windows-hardening/windows-local-privilege-escalation/dll-hijacking. 2022.

[31] ICS-CERT. *CVE-2017-7932*. https://nvd.nist.gov/vuln/detail/CVE-2017-7932. 2017.

[32] ICS-CERT. *CVE-2017-7936*. https://nvd.nist.gov/vuln/detail/CVE-2017-7936. 2017.

[33] jeremiahsnapp. *OpenSSL, method for proxies*. https://gist.github.com/jeremiahsnapp/6426298#openssl. 2022.

[34] Keen Security Lab. *Binary Abstract Inspector*. https://github.com/KeenSecurityLab/BinAbsInspector. 2022.

[35] Clément Labro. *Windows DLL Hijacking (Hopefully) Clarified*. https://itm4n.github.io/windows-dll-hijacking-clarified/. Apr. 2020.

[36] Linux. *cryptsetup(8) — Linux manual page*. https://man7.org/linux/man-pages/man8/cryptsetup.8.html. 2022.

[37] longld. *gdb-peda*. https://github.com/longld/peda. 2021.

[38] Gordon Lyon and the Community. *Nmap*. https://nmap.org/. 2022.

[39] marin-m. *PBTK*. https://github.com/marin-m/pbtk. Mar. 2021.

[40] OsandaMalith. *PESecInfo - GitHub's repo*. https://github.com/OsandaMalith/PESecInfo. Oct. 2018.

[41] PortSwigger. *Burp Professional.* https://portswigger.net/burp/pro. 2022.

[42] Rapid7. *MSFVenom.* https://github.com/rapid7/metasploit-framework/wiki/How-to-use-msfvenom. 2022.

[43] RedHat. *CVE-2020-14382.* https://access.redhat.com/security/cve/CVE-2020-14382. 2020.

[44] ReFirmLabs. *Binwalk.* https://github.com/ReFirmLabs/binwalk. 2022.

[45] RockChip. *RockChip RK3399 TRM.* https://www.t-firefly.com/download/Firefly-RK3399/docs/TRM/Rockchip%20RK3399TRM%20V1.3%20Part1.pdf. Dec. 2016.

[46] Objectif Sécurité. *TProxy.* https://objectifsecurite.gitlab.io/tproxy/. 2022.

[47] tacnetsol. *Custom Ghidra scripts.* https://github.com/tacnetsol/ghidra_scripts. May 2021.

[48] GoodFET's team. *FaceDancer21.* https://int3.cc/products/facedancer21. 2020.

[49] ImageTragick's team. *ImageMagick Is On Fire — CVE-2016–3714.* https://imagetragick.com/. 2016.

[50] MITMProxy core team. *MITMProxy.* https://mitmproxy.org/. 2022.

[51] F-Secure teams. *Multiple Vulnerabilities in Barco ClickShare.* https://labs.withsecure.com/advisories/multiple-vulnerabilities-in-barco-clickshare/. Dec. 2019.

[52] Wikipedia. *DMA Attack.* https://en.wikipedia.org/wiki/DMA_attack. 2022.

[53] Windows. *Process Monitor.* https://docs.microsoft.com/en-us/sysinternals/downloads/procmon. 2022.

[54] Windows. *WinDBG.* https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/debugger-download-tools. 2020.