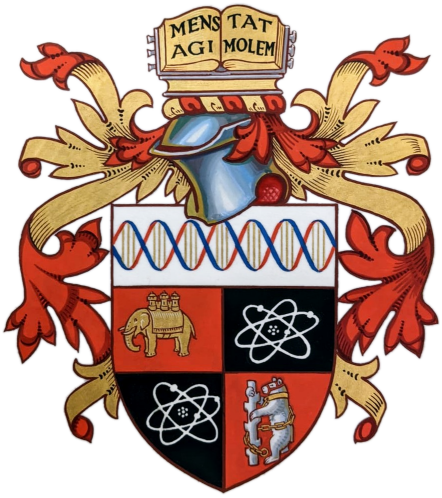
WARWICK
THE UNIVERSITY OF WARWICK



SECURITY ANALYSIS OF AN IP CAMERA

by

JAMES NOTTINGHAM
U2102122

CS310 - THIRD YEAR PROJECT

supervised by

PROF. FENG HAO

UNIVERSITY OF WARWICK
DEPARTMENT OF COMPUTER SCIENCE

April 27, 2024

Abstract

The Internet of Things has rapidly proliferated over recent years, with devices becoming increasingly more commonplace amongst homes and businesses. Despite this, security measures employed by these devices remain substandard, leading to exploitable vulnerabilities. Additionally, there remains a current lack of security testing of available devices. Through security research vulnerabilities can be exposed and mitigated, leading to enhanced security of the Internet of Things sector as a whole. In this investigation, a common internet connected hidden camera is analysed, deriving its communication structure and enumerating vulnerabilities. We perform this through a process primarily consisting of packet capture and analysis. Through multiple flaws in communications security, several vulnerabilities were identified, allowing for complete compromise of the command and control system of the camera. Furthermore, the communications structure is compared to that of a different camera, demonstrating the similarities and differences between the two implementations, highlighting areas of common weakness. From these findings, a better understanding of the state of security in commercially available Internet of Things devices is achieved, paving the way for future security research.

List Of Keywords

Security, Internet of Things, IP Camera, Android, Communications, WiFi, Vulnerability

Acknowledgements

I would like to thank my supervisor Prof. Feng Hao for his advice and feedback through discussions to aid this investigation. I would also like to thank my friends and family for their continued support throughout this project.

Contents

List of figures

List of tables

List of abbreviations

AP access point . . . . . 11 ff., 21, 23 ff., 27 f., 36, 38, 40–43, 59 f.

APK android package . . . . . 8, 13, 29 f.

BSSID basic service set identifier 40

DHCP dynamic host configuration protocol 36

GPU graphics processing unit 11

IoT internet of things . . . . . 1–7, 12, 22, 34, 60, 62, 66 f., 69

IP internet protocol 1 f., 5–8, 20, 33 f., 36, 42 ff., 48, 50, 52 f., 57 f., 61 ff., 65, 67

IR infra-red 23, 60

JSON JavaScript object notation . . . . . 46, 49, 51, 53 f., 64 f.

LAN local area network . . . . . 21, 24, 26 f., 33, 40, 42, 51, 61 f.

MAC media access control 34, 62

MITM man-in-the-middle 51

NIC network interface card 28

OS operating system . . . . . 11, 13, 26, 33 f., 61 f.

RTSP real time streaming protocol 8, 35

TCP transmission control protocol . 7 f., 34, 37, 42, 45 f., 48, 50, 62 ff.

UDP user datagram protocol . . . . . 7 f., 63 f.

VM virtual machine 11

VoIP voice over IP 34

1 Introduction

The use of internet of things (IoT) devices has proliferated throughout recent years, encompassing many facets of life for individuals and businesses across the globe. IoT refers to traditionally non-internet connected devices such as cameras or fridges that have been given networking capabilities, allowing them to interact both their environment, and other internet connected endpoints. From personal use in smart homes to business applications that increase efficiency and automate tasks, broad application of such devices has led to them becoming commonplace in both public and private locations. Although such endpoints have existed since the 1980s [1], their general uptake and normalisation of use throughout society has only seen explosive growth over the past decade [2]. Due to their rapid recent uptake, IoT devices have permeated many sectors aided in part by their ability to assume the role of almost any commonplace item with few functional drawbacks in comparison to their non IoT counterparts. Arising from their rapid proliferation, costs of IoT devices have become more affordable [3], making them more applicable for everyday applications.

One such type of common IoT device is the internet protocol (IP) camera, a form of internet connected smart camera capable of networking and being accessed remotely. Specifically in the case of spy cameras, camera lenses are often concealed within other objects, or constructed in a small form factor as to avoid detection. Such devices are uniquely positioned amongst internet connected endpoints to provide video and oftentimes audio in a surveillance scenario. Both legitimate and illegitimate use cases for these exist; in example, monitoring of private property and undisclosed spy cameras situated in Airbnbs respectively. The illegitimate use of spy cameras is evidenced by several cases over recent years where undisclosed spy cameras have been discovered by guests at Airbnbs. Moreover, over half of respondents in a 2019 study were worried about the prospects of hidden cameras in Airbnbs [4].

1.1 Motivation

Unusually in comparison to other forms of endpoint, IP cameras are exposed to sensitive video and audio data in both commercial and personal settings. This is only exaggerated in the case of spy cameras as, due to their hidden design, users may be more likely to present sensitive personal information in view of such devices if they are not aware of the presence of the device. For example, if a user was placed directly in front of a visible camera, they would likely take extra precautions protecting visual information such as inputting a debit card PIN compared to if they were not aware the camera was present. As such, ensuring effective security measures are implemented in such devices is of paramount importance; the data collected by these cameras makes them more probable to be targeted in cyber attacks than other forms of IoT endpoints. Only through thorough security being adopted throughout the entire camera system can the CIA triad be maintained and end users kept safe.

Despite the importance of security in the applications of such endpoints, measures taken to ensure security of devices and their software interfaces often remain far substandard leaving vulnerabilities that can be taken advantage of. Malicious actors whom discover said vulnerabilities are subsequently able to launch attacks, gaining control over devices and information, potentially without the user ever being aware that their camera has been compromised. Disclosure of information is already prevalent within the IoT space as highlighted by *Shodan*, an IoT search engine [5] which allows for easy searching and viewing of insecure endpoints. Platforms of this manner which allow such easy discovery of millions insecure devices highlights the risks posed by inadequate security measures and the urgent need to address the issue of IoT vulnerabilities.

By addressing these vulnerabilities, we can better mitigate both the risks posed to individual users, and to IoT ecosystems as a whole as a single compromised device can pose a threat to other components in its network. One such method we can use to aid in promoting enhanced security in the IoT space is through conducting security analysis on existing products. Doing so establishes readily available information about the systems through which devices operate, and exposes any vulnerabilities to public attention, pressuring manufacturers to take measures to secure their products and notifying the public where they may wish to take additional precautions before using their devices. Additionally, publishing details surrounding IoT communication structure provides foundational knowledge from

which future research on similar systems can build upon.

1.2 Objectives

The overarching goal of this project is to attempt to construct a detailed representation of the communication structure of the *UYIKOO Spy Clock Camera 140°HD 1080P* [6], following by enumerating and verifying potential vulnerabilities in a systematic manner, finally reporting on any findings. The project was further split into a multitude of smaller, attainable targets using the MoSCoW method of objective prioritisation. MoSCoW allows an importance to be assigned to each tangible objective, aiding in focusing the project on those objectives which are most vital to ensuring project success [7]. Prioritising of targets allows for improved scheduling of the project and increased flexibility in the case that some of the tasks overrun as some of the lower priority objectives can be pushed out of scope without having a negative impact on the core goals of the project.

As the project evolved, objectives were reprioritised and altered to adapt to information discovered and issues faced as progress was made. Reprioritisation was possible without negatively impacting the quality of the project due to the agile methodology as detailed in chapter 4 which allowed for beneficial alterations to be implemented without significant time loss resulting from extensive re-planning. Specifically, this involved the removal of a focus on recreating attacks generally common in IoT devices as, whilst discovered vulnerabilities may fall under common categorisations, recreation of common attacks would require rewriting to match the specific communication structure determined. Hence, separation between common attacks and those derived from testing this device was not logical.

Must:

- Reverse engineer the app to gain an understanding of how it works and communicates with the camera.
- Perform packet capturing to derive the communication structure and technical methods used by the camera system for communication.
- Report on any found vulnerabilities including the methodology used to exploit that particular vulnerability and the affect it had on the camera.

Should:

- Perform testing to obtain a list of potential vulnerabilities in this particular device.
- Attempt attacks on any potential vulnerabilities to verify security flaws using both manual attacks and automated attacks such as fuzzing if appropriate.
- Explain attack methodology clearly enough in the report such that another researcher could replicate the attack to test their own device.

Could:

- Perform security analysis on an additional IP camera to identify if there are vulnerabilities common to multiple IP cameras currently on sale.
- Outline a testing methodology that can be used to perform security analysis in a systematic and repeatable manner on other IoT devices.
- Test vulnerabilities from a previous report on an IP camera and report whether the camera being tested in this project is vulnerable to these previously known attacks.

Won't:

- Perform attacks with the potential to disclose or otherwise affect data from other users.
- Attack external servers in a manner other than for communication with the purchased camera.

2 Background

Before conducting the investigation into the camera being tested, it is first important to evaluate the background literature currently available. By doing so, we gain a foundational understanding of the current state of the sector, and the technologies underpinning the device being tested. As highlighted in the introduction, IoT devices have experienced explosive uptake in the past decade with 14.3 billion active IoT endpoints present in 2022 [8], greater than the number of non IoT devices recorded [9]. Hence, security of such endpoints has significant implications; by first gaining an understanding of the existing research, we are able to better position this investigation to provide information in areas which lack current research.

2.1 IOT security overview

Despite the prevalence of IoT devices, there remains a defect of systematic and thorough security research, particularly when narrowing the field to that of IP cameras in particular. However, where security analysis or testing has been performed on IoT devices, numerous oversights and vulnerabilities have been discovered, demonstrating the need for both further security analysis of more devices and more effective mitigations implemented by manufacturers. Furthermore, there is no evidence of security analysis performed on the *UYIKOO* camera being investigated in this project. The device in question is positioned at a low price point (~£50) and is easily available for purchase online, making the consequences of vulnerabilities applicable to a large consumer market.

In one previous investigation into the security of specific IoT devices, Wurm et al. [10] perform a security analysis of both a commercial and an industrial IoT endpoint. First, the commercially available device was analysed, starting by conducting a hardware analysis. From this they are able to obtain root access to the Linux system running on the device. Subsequently to root access being gained, network analysis is performed, revealing plaintext communications and details of the firmware update

process. Uniquely compared to other explorations of device security, the study then performs a further analysis of an industrial IoT endpoint, following the same basic format of using a hardware analysis as a basis for compromise. It is concluded that both devices are vulnerable, with some suggestions for security enhancement made. Notably, the industrial endpoint, which may often be assumed to be more secure than commercial products, still does not offer sufficient security to thwart significant compromises to its system. Although successfully providing vulnerability verification and comparison, this study focuses primarily on attacks which require hardware access as a prerequisite for exploitation. Due to the networking focus of IoT, these attacks are less likely to be performed in practise than purely network based attacks, particularly in the case of commercial devices which are more likely to have publicly exposed networking capabilities. Additionally, where networking was explored, little detail as to the communication structure with external interfaces such as compatible apps was attempted to be derived.

Similarly, when performing an investigation on an IP camera, Biondi, Boganni, and Bella [11] discover a commercially available IP camera to have multiple significant security flaws. Unlike the study performed by Wurm et al., no hardware exploitation is performed, instead focusing fully on vulnerabilities within the communication and networking capabilities of the camera. Here they find three primary vulnerabilities consisting of a denial of service attack, video feed reconstruction, and an oracle exploit in which motion information can be derived from packet transmission rates without requiring access to the video feed itself. In doing so, a detailed view of the communication system is explained. Making use of extra hardware, resolutions to the vulnerabilities are both proposed and verified in practise. Overall, it is found that although the tested camera has security measures in place, they do not cover all scenarios, a finding intended to be validated over other IoT devices. The work performed in our study over the *UYIKOO* camera provides a similar point for which comparison can be drawn.

Beyond the scope of individual device security analysis, smart cameras also prove to be subject to basic security failings at a massive scale. Bugeja, Jonsson, and Jacobsson [12] reveal how *Shodan* can be used to rapidly discover a large number of vulnerable devices with little technical skill required. Whether from open ports, insecure default credentials, or other weaknesses, many devices broadcast information which can be publicly accessed through *Shodan*. From here keywords were used to identify targets and identifiable information such as firmware was obtained. By scanning against the CVE database, pre-existing vulnerabilities affecting the target devices

were able to be obtained. The results shown only further demonstrate the need for enhanced IoT security.

Although IoT security has been demonstrated to be an important factor as the proliferation of IoT devices progresses, there is a lack of regulatory enforcement of the application of good security practises in such devices and their associated software. The UK has a code of practise that outlines methods in which IoT security can be increased such as not implementing default passwords and secure storage of credentials [13]. However, this serves merely as a guideline and does not contain any legally enforceable security requirements. Frameworks allowing for more comprehensive security of IoT systems have additionally been proposed, such as the security framework devised by El-Gendy and Azer [14]. In this framework, security is divided into sections concerning the physical, network, and application components of an IoT system. Using such a framework would provide more robust security across networked devices provided the proposed architecture was adhered to. However, this relies on manufacturer investment into greater security, a shift unlikely to occur without external pressure such as legally binding regulation.

2.2 Networking

To perform a security analysis of networked devices, a fundamental understanding of network communications is essential for conducting effective research. Internet functionality is divided into a layered architecture, providing a common structured way in which all communications can be represented. Commonly, this is represented using the five layer protocol stack [15]. Of this stack, we focus primarily on the transport layer which provides communication services to hosts. The protocols used in this layer will likely bear significant influence to the communication structure of the camera being tested. The two most significant protocols applied at this layer are transmission control protocol (TCP) and user datagram protocol (UDP).

UDP is a connectionless protocol for data transmission that does not guarantee delivery or ordering of packets to the recipient, providing only the basic services required by IP. As a result, UDP has minimal overheads and allows for low-latency communication. Conversely, TCP is a connection oriented protocol, requiring a three way handshake to be performed between the sender and recipient before data transmission is allowed to begin. Additionally, TCP ensures delivery and correct ordering of packets, using methods such as sequence numbers and timestamping to

achieve these requirements. Beyond this, additional protocols are built as extensions of the features provided by TCP or UDP. In the context of video streaming as is required by IP cameras, one such protocol used is the real time streaming protocol (RTSP) [16]. This is a protocol specifically developed for video streaming whilst providing control over the video stream. It can be delivered over either TCP or UDP.

Additionally, as the primary goal of IoT devices is to have some physical interaction with the world, different structures to that of normal internet devices are used to define the function of these devices. Several different layered approaches have been proposed that are designed to enhance privacy and security such as a six-layered structure based on the network hierarchy [17]. IoT devices are able to communicate with each other through a variety of different methods such as Device-To-Device or Device-To-Cloud [18] amongst others, but can be seen as split between the Device-To-Device model which only involves the communication of devices directly connected over a network, and the other models all of which require an application service provider to connect the device. The camera being analysed in this project is capable of Device-To-Device connection as well as a form of connection via an application service provider.

2.3 Android application testing

Finally, as reverse engineering is to be performed upon an *Android* application, foundational knowledge needs to be established as to the format of such testing. The *Android* system is built in a layered architecture with a foundation of a Linux kernel with C or C++ libraries on top of it to add functionality. Apps are then loaded onto the system in the form of android package (APK) files [19]. The underlying code of these packages is often written in Java, but can be written in other languages such as C. Decompiling an APK file to obtain the source code can allow for a better understanding of the function of an app, and how it interacts with libraries to achieve its goals.

Testing of an android app can be done using an emulated android device to run the app, connected to a host machine via Android Debug Bridge [20] which allows Linux commands to be sent to and executed on the android device. Penetration testing of an application should be done in a structured manner and may follow frameworks such as Open Web Application Security Project [21] to focus on specific

vulnerabilities or follow a certain testing methodology.

3 Technical Methodology

Prior to any experimentation or testing being performed, a methodology was constructed to outline the steps required to conduct the investigation. This is of high importance as the methodology shapes the direction of the investigation and provides detail as to the resources required to conduct the project. Additionally, the methodology provides a rough outline of the structure of the investigation which can be followed to ensure that the project remains on track.

3.1 Initial methodology

The first version of the methodology consisted of the following primary steps:

1. Reverse engineer app to retrieve source code.
2. Analyse source code to determine communication structure between the app and the device.
3. Using source code and knowledge of communication structure, determine potential vulnerabilities with the system.
4. Perform attacks to verify the existence of the potential vulnerabilities.

To reverse engineer the app, it was proposed to use an emulated *Android* device to run the app which interfaces with the camera. The emulated device would be running on a virtual machine. This allows for the camera app to be run from a desktop computer, subsequently making it easier to analyse the app and manipulate communications. Determining the communication structure was intended to be primarily derived from analysis of the decompiled source code. This stage was planned as a prerequisite for any assessment of vulnerabilities. Separation of the stages for determining communication structure and finding vulnerabilities was intended to reduce confusion during the project by compartmentalising the tasks into smaller, more manageable chunks. Similarly, the stages for enumerating potential vulnerabilities and testing by performing attacks were also separated. In

this proposed methodology, the communication structure and potential vulnerabilities would be fully determined before any connection is attempted between the attacking system and the camera. All other **should** and **could** objectives were to be performed after the four primary steps had been completed.

3.2 Methodology changes

After initial technical work on the project began, discoveries relating to the structure of the app and communication system between the app and the camera highlighted flaws with the initial proposed methodology. Hence, it was decided to make alterations to the methodology before proceeding further to better align the methodology with the system being tested.

During the initial setup of the app on the emulated device, a *VirtualBox* [22] virtual machine (VM) was used to run the testing operating system (OS). *Android studio* [23] was used to run the emulated device. However, it was quickly discovered that this was not a feasible method for carrying out the project as *VirtualBox* does not support graphics processing unit (GPU) pass-through [24]. Due to the lack of GPU acceleration, the VM lacked the processing power to run the emulated device; hence a switch was made to using a native OS. Subsequently, it was discovered that the app requires access to external access points (APs) which was not possible using the emulated *Android* device as the emulator only supported internet connections through a simulated Ethernet connection. Hence it was not possible to connect to the camera using the emulated device, so a methodology change was made to use a mobile phone to run the app rather than the emulator.

Additionally, the last three of the four sections outlined in section 3.1 were merged to be less separate as it was found from initial testing that determining communication structure, enumerating potential vulnerabilities, and testing vulnerabilities were all interlinked and would prove difficult to perform completely independently of one another. Thus, despite the benefits of compartmentalisation, it was decided to merge the stages whilst keeping them as independent goals for the project. Moreover, the scope of the methodology used for determining communication structure and finding potential vulnerabilities was expanded to include packet capturing and analysis in addition to source code analysis to allow for more information to be collected, and a more accurate representation of the communication system to be derived.

3.3 Resources and risks

As the project involves testing of a physical device, a variety of both hardware and software devices are required to be able to perform the testing. Additionally, a risk assessment was conducted at the beginning of the project to highlight any potential threats that may pose a risk to project success.

3.3.1 Hardware

- **Laptop computer** - Used to capture and analyse intercepted packets and run attacks on the camera system. Also used to record findings and results of testing.
- **Mobile phone** - Used for installing and running the app that communicates with the camera.
- **UYIKOO spy clock camera [6]** - The IoT camera being analysed in this project.
- **Qualihome mini spy camera [25]** - The secondary IoT camera being analysed.
- **Alfa AWUS036NHA network adapter [26]** - An external network adapter connected to the laptop computer used for creating an AP and sniffing packets.

Figure 3.1: Alfa AWUS036NHA network adapter being used for testing

The AFLA network adapter shown in figure 3.1 is used to interface with the phone and the camera rather than the inbuilt network adapter in the laptop. We do this

as the ALFA network adapter has an Atheros chipset [27] which gives the adapter support for monitor mode which is not supported using the inbuilt network adapter in the laptop. This allows us to sniff packets.

3.3.2 Software

- **HDlivecam [28]** - The app that is used to interface with the camera. Capable of sending commands and receiving video feed.
- **Kali Linux [29]** - A Linux distribution specialised for information security applications. Used as the OS to run all the programs on the laptop. This distribution was chosen for its specialisation in information security, many of the required programs to perform testing come preinstalled in this OS.
- **Wireshark [30]** - A packet capturing utility used to intercept the messages between the camera and the phone.
- **Python [31]** - High level programming language used to construct attacking scripts to compromise the camera.
- **VScode [32]** - Text editor used to write the attacking python scripts and allows for a simple interface for GitHub.
- **GitHub [33]** - Source control software used to provide version control and an external backup of collected data.
- **Overleaf [34]** - LaTeX program used for report writing.
- **JADX [35]** - APK reverse engineering software used to decompile the APK of the *Android* app to retrieve the source code for analysis.
- **Create_ap [36]** - Access point creation software allowing the network adapter to host a WiFi AP that other devices can connect to.

Besides the primary software components outlined, several additional software resources were applied in certain aspects of the project. However, these components were employed in a substantially more minor capacity than the listed resources.

3.3.3 Risk assessment

A risk assessment was conducted so that potential threats to project success could be highlighted before beginning the experimentation. The issues and risks highlighted were categorised by area of impact and severity of the risk. Mitigation strategies

were proposed to limit the potential damage of each risk.

Table 3.1: Risk assessment

Category	Issue	Risk Severity	Mitigation Strategy	Residual Risk
Time	Laptop computer breaks, preventing reverse engineering and testing.	Low	Access to a desktop computer allows for continued testing after installing required software.	Low
Time	Camera becomes broken or damaged.	Medium	The project has a £200 budget allowing for a replacement to be purchased, although no testing of the camera would be possible until the replacement arrives.	Low
Scope	Additional IoT camera is required for testing.	Medium	Additional cameras are readily available for purchase online within the project budget.	Low
Time	Software for reverse engineering and vulnerability testing does not function as intended.	High	Alternative Linux distributions are available in the case of the chosen distribution being unsuitable. Additionally, all of the other software components have alternatives that could be used in the case of the software being unsuitable for this project.	Medium
Time	Illness	Low	Contingency has been built into the timeline for project completion, allowing for some time lost due to illness without adversely affecting project schedule.	Low
Quality	No security issues are found to be present in the camera.	Medium	The primary goal of the project is to determine the communication structure between the camera and the app which would still be possible in this case. Additionally, in this case it would be suitable to conclude that the camera is invulnerable to a range of attacks on the communication system demonstrating the system employs effective security measures; still providing a valuable contribution to the field of security knowledge in IoT devices.	Low

3.4 Legal, social, and ethical issues

As with any ethical hacking or penetration testing project, precautions were taken to ensure all hacking performed on the devices was done so legally in accordance with the computer misuse act [37]. To facilitate this, both devices being tested were privately purchased, ensuring legal permission to perform testing on the devices. Additionally, testing for general security of the remote servers used by the vendor was declared out of scope as no permission for testing was gained from the server operators and product vendor. In all cases where attacks required active malicious communication with external servers; attacking scripts were constructed to only manipulate information related to the owned device and user account with care taken to ensure that requests do not return data linked to other users of the service.

No ethical consent was required to be obtained from third parties as no other persons were involved with the project.

4 Project Management

An agile based approach was used for this project due to the short time frame for for completion. No specific agile methodology was strictly adhered to but the project implemented features of the scrum methodology such as short sprints of work with progress reviewed and future plans adjusted at the end of each one or two week sprint [38]. Doing this was key to the success of the project as security analysis is inherently unpredictable. Hence, it is difficult to strictly timeline and comprehensively define all requirements from the beginning of the project. This is due to the amount of time and resources required for testing being dependent on the exact system being tested, where these requirements cannot be accurately known before technical knowledge of the system is acquired. A more rigid project management approach such as waterfall would not be suited to this as waterfall is best applied to projects where the project scope is unlikely to change; waterfall is a linear method where project changes would require re-planning from the beginning [39].

The agile methodology used allowed the project to easily adapt to change in scope and timing without wasting time for extensive re-planning of the project. To this end, supervisor meetings were scheduled based on progress as opposed to at regular intervals to match the nature of progress being more episodic than continuous as obstacles were overcome. Flexibility to change in the project was an essential component as factors such as the number and type of vulnerabilities discovered affected the time required and ordering of the stages of the project.

4.1 Timeline

Despite the agile nature of the project, a timetable was constructed to provide a rough outline for the primary goals of the project. This was produced primarily to visualise the structure of the project and track the progress of what has been completed, rather than providing a concrete schedule for all activity to follow as this

would not be appropriate for an agile approach. The timetable was produced in the form of a Gantt chart to clearly show all the tasks with their respective timings and progress.

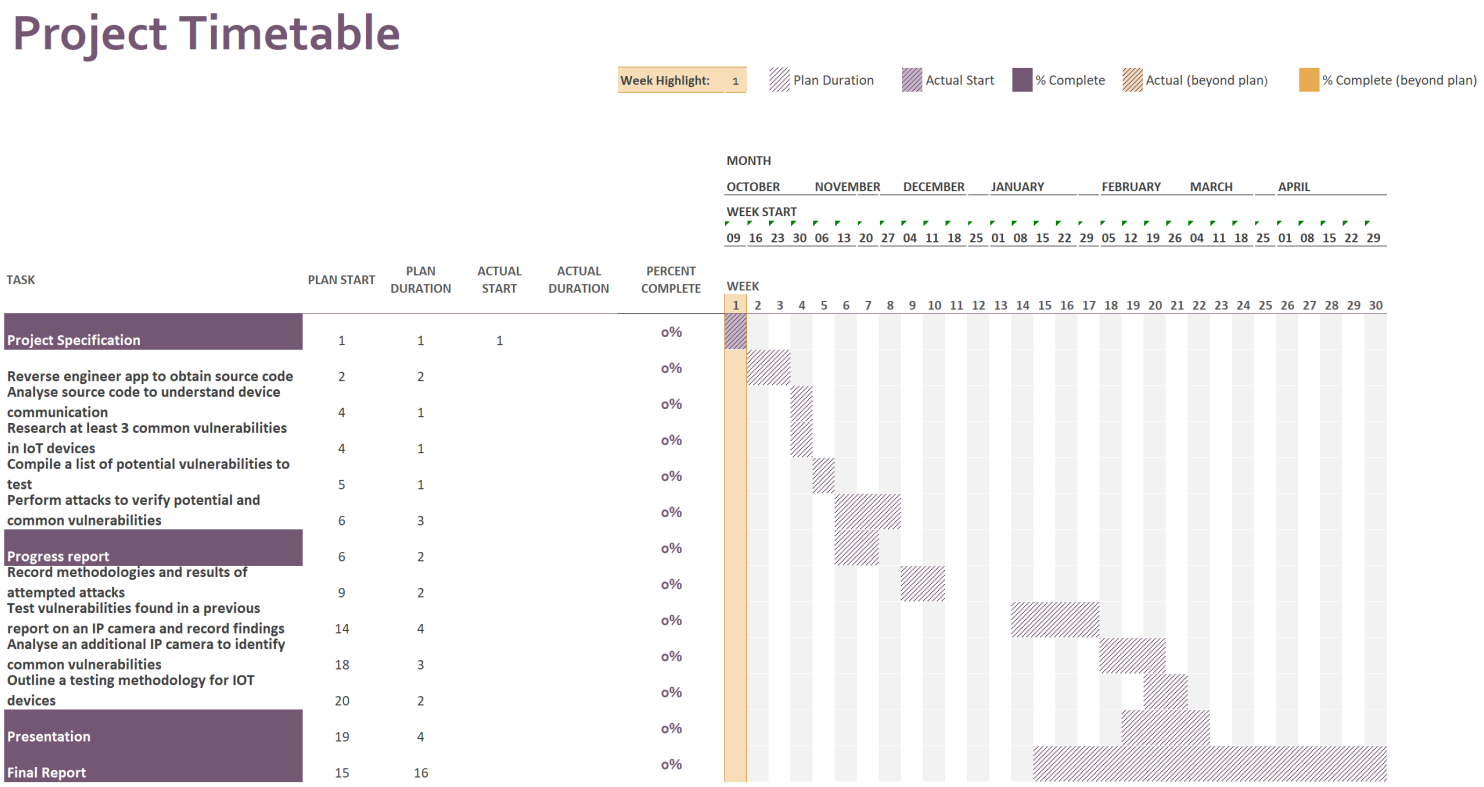


FIGURE 4.1: Original project Gantt chart prior to project start

The original project timetable included all of the original objectives before any changes to project scope had been made. A break was incorporated over weeks 11 to 13 as shown in figure 4.1 to provide a break over the Christmas period and also allow for contingency should previous tasks overrun. Contingency time allows for some overrun of the early tasks without requiring alterations to the schedule of tasks subsequent to the contingency period. Additionally, an extra week of contingency was planned around the Easter weekend. However, due to limitations of the spreadsheet software used to construct the Gantt chart, this is not represented in the chart.

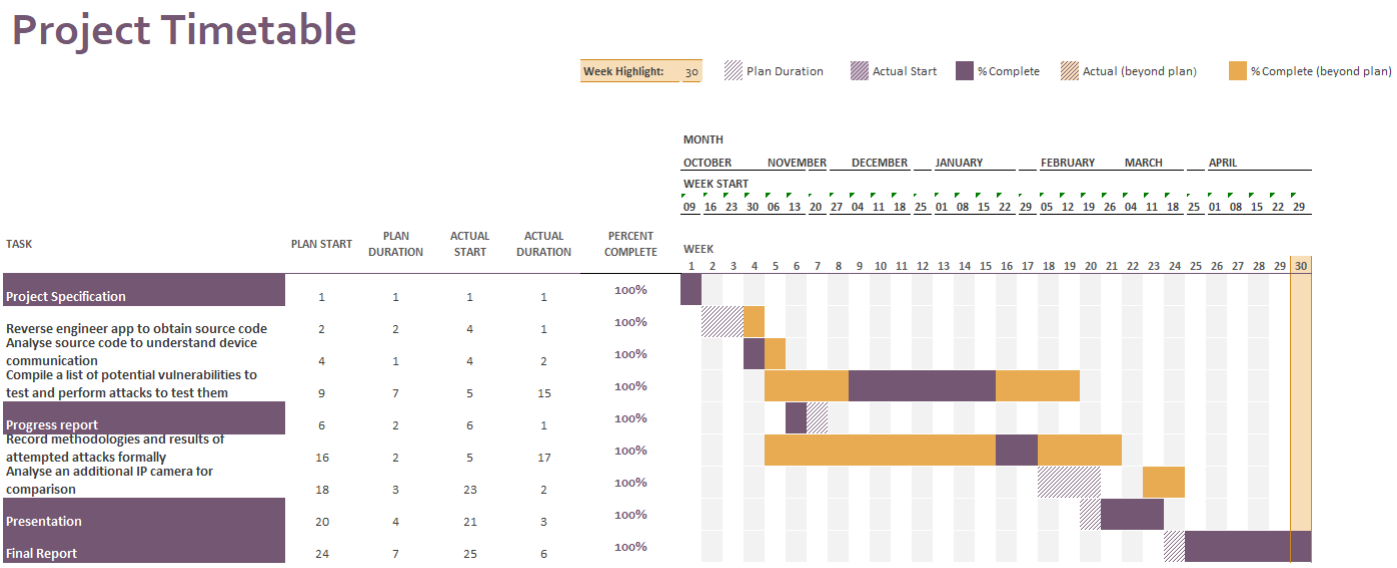


FIGURE 4.2: Gantt chart updated with actual project progression

As the project progressed, changes to the scope and timeline were made which the Gantt chart was updated to reflect. Primarily this involved combining the sections for finding and testing vulnerabilities, matching the revised version of the methodology. Additionally, several of the low priority objectives were removed from the Gantt chart to allocate additional time to the higher priority tasks. The contingency time originally incorporated proved useful in minimising the impact overruns had on high priority tasks. Updating the chart with these changes as they occurred aided in reducing any confusion caused as changes were implemented, and helped ensure the project remained on track.

5 System Overview

Before performing any technical analysis of the system to determine communication structure, an overview of the camera system and its functionality was first obtained. Doing so allows us to build up a better understanding of what capabilities the camera has and provides insight into the operation of the device without requiring technical investigation. With improved understanding of the system obtained from the system overview, we are able to better plan the approach used to determine communication structure and enumerate vulnerabilities. Due to the lack of extensive documentation provided by the vendor as to the functions of the camera or the app, collecting general information about the system was done experimentally, primarily by running the camera under normal usage conditions and exploring the features provided within the app.

5.1 Camera hardware

The IP camera being tested is a hidden camera in the form of a digital clock. In normal operation, front and back cover panels are placed to cover up the controls and camera portion so that the device looks like a standard digital clock.

(a) Camera front (b) Camera back

Figure 5.1: Front and back of the camera with protective covers removed

The camera consists of the following primary components as highlighted in figure 5.1:

Item number	Description
1	Camera module
2	Digital clock
3	Infra-red (IR) blaster
4	Battery
5	USB charging port
6	Reset button
7	On/off switch
8	Indicator LED
9	SD card slot

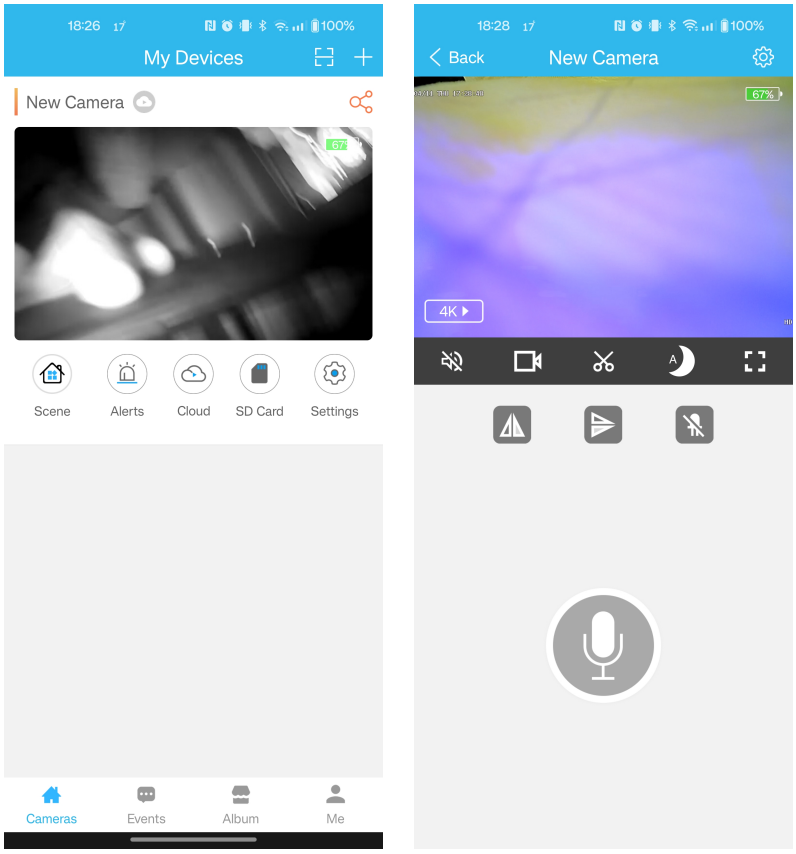
Table 5.1: Table of camera components

The battery integrated into the camera allows for operation of the camera regardless of whether it is connected to an external power supply via the USB port. However, the battery depletes quickly whilst the camera is streaming video. Although not shown in figure 5.1 the camera internally contains both a WiFi module and WiFi receiver as it has functionality to both connect to external networks, and create its own APs. APs created by the camera do not provide connection to the internet, only to the cameras local area network (LAN).

For the testing performed in this project, the indicator LED on the back of the camera is of particular interest. Unlike the other components on the device, the indicator LED can be toggled on or off via the software that interfaces with the software. Thus, command execution attacks performed on the camera focus on toggling the state of the indicator LED to provide immediate visual confirmation as to whether an attack has successfully been executed.

5.2 Application

To interface with the camera, the *HDlivecam* app is provided as the method for users to access their cameras. The app supports both *iOS* and *Android*.



(a) Home screen (b) Camera feed

Figure 5.2: Home screen and camera page for the *HDlivecam* app

The app is published under the vendor name of *Care Home* [40], although the app manufacturer website links back to a company called *smartcloudconn* [41]. Little information is available online about the company, although services provided by this company have been previously linked to other insecure IoT devices [42].

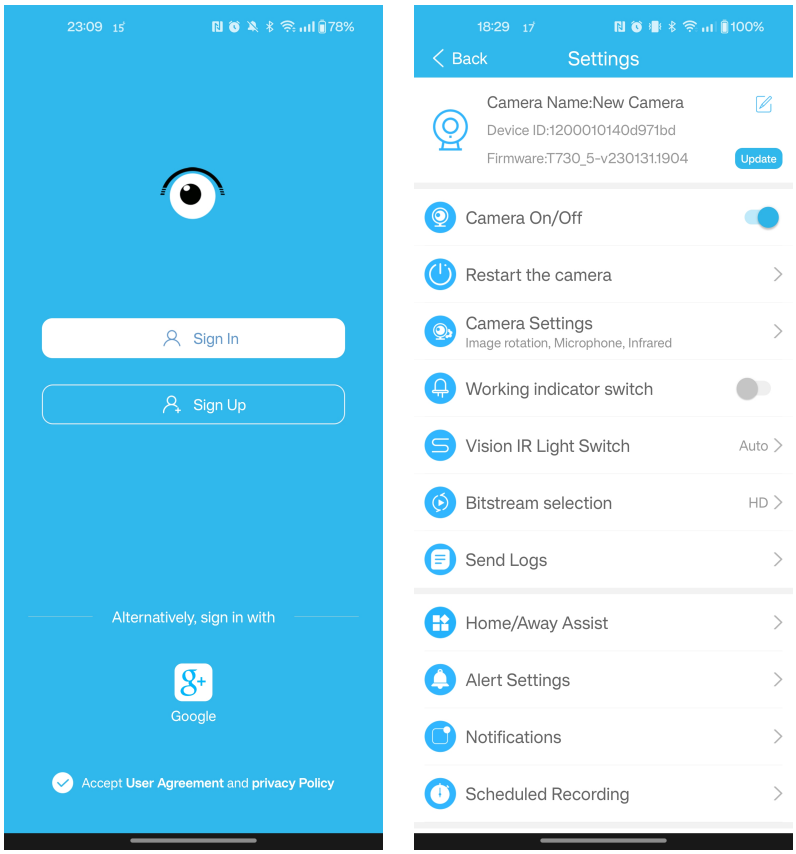
The app opens to a homepage which serves as a page containing all of the devices registered to the user. Although only one device is registered for this project, the app supports additional devices which may be applicable if a user was to use multiple devices to obtain greater camera coverage. The other tabs at the bottom of the home screen are for *events*, *album*, and *me*. The events section contains records related to the motion detection feature whereby if the camera detects motion in the video feed a notification is sent to the device with the app installed, and a record of the event is created. The album tab contains images and videos taken by the devices registered to the user. Media can also be downloaded from the cloud or the camera although cloud storage requires a subscription payment. The me tab contains basic profile information and app options. Due to cloud services requiring additional payment, none of the cloud features are tested in this project.

Each camera card contains some additional options for the camera. Alerts simply links to the events tab, scene allows for configuring different camera options for home and away, Cloud and SD card are not available with the setup being used. When the camera card is clicked, the live camera feed is opened which provides more controls over the camera. At the top of the page are two options for adding a new camera, via a QR code or manually. The manual method supports adding cameras either by the camera generated AP, or by searching for cameras already on the local network. Below these buttons is the option to share a camera. This allows the user to give other users permission to view and interact with cameras owned by that account.

Clicking on the camera card opens the camera feed display which provides a real time stream of the video feed. Here several alterations can be made to the video feed such as mirroring the video, or taking pictures and video. Additionally, there are several options for controlling the camera hardware including toggling the microphone, indicator LED, and night vision mode using the infra-red (IR) lights on the front of the camera. Replicating the indicator LED operation offered from this page is the primary focus of the attacks carried out on the camera.

5.2.1 Use case scenario

Outlined here is a regular scenario detailing the procedure required to setup the app, add a camera, and open the video feed and send commands to the camera.



(a) Initial display (b) Camera settings

Figure 5.3: Initial logged out display and extra camera settings

When the app is initially launched, the user is presented with an initial display prompting them to either login or sign up. On this screen, if the app logo is tapped repeatedly 10 times, a zipped file of password protected logs is created. From this display, the user can create their account which requires an email address and a password that must be at least six characters long and have at least one letter and one number. The email address is not validated and hence a fake email address would be accepted by the app.

Once the user has created an account, they are taken to the homepage whereupon they are prompted to add a camera to the account. The user then has the option to either add a camera using the AP created by the camera, or by searching for cameras connected to the same LAN as the device. In the case of a camera that has never been set up before, the initial setup must be done using the cameras AP. After the device has been associated with the user account using this method, the user can set

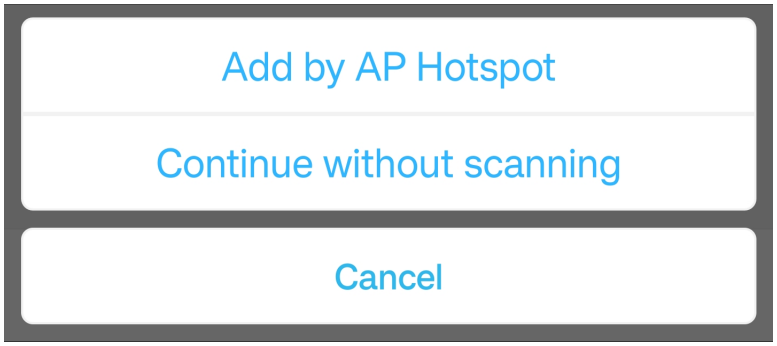


Figure 5.4: Options for adding a camera

an external network for the camera to use by selecting the appropriate WiFi network and providing the network password. This allows the camera to be accessed from anywhere rather than just within the range of the AP generated by the camera.

Once this is done, the device is fully set up and ready for use as described in section 5.2. In addition to those features, the user can also press the settings button to be taken to a more detailed settings page for the camera that provides more control over the camera than is available from the camera feed screen. Here there is also the option to produce a log file, but despite being signed into the app, log files produced from this display are still password protected with a different password to that of the user account and hence were unable to be opened. From this display, pressing the indicator LED toggle causes the LED to turn off, a function that is replicated in the attacks performed on the camera.

6 Testing setup

The testing of the camera system is performed using a laptop running the Kali Linux OS. The laptop can be connected to the camera and the phone using the ALFA network adapter as shown in figure 6.1.

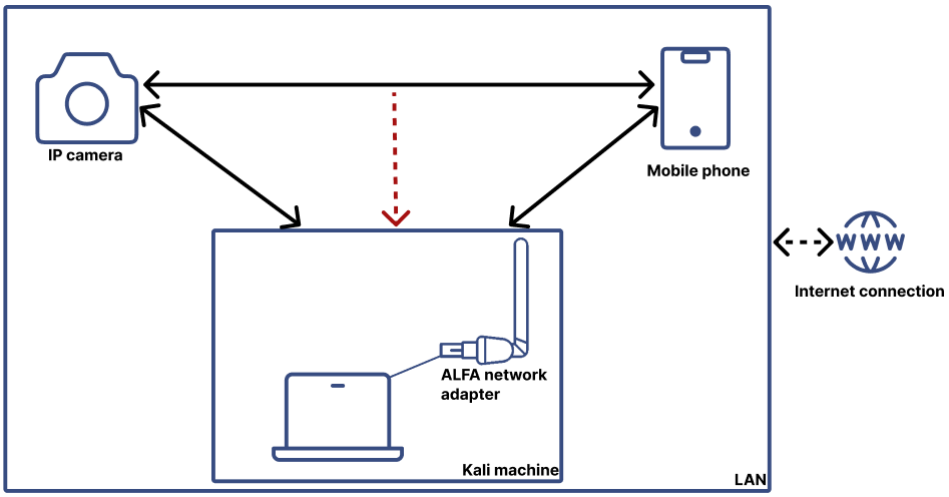


Figure 6.1: Generic version of the testing setup

The setup shown here provides a representation of the general format of the test bed used for the project, but the exact format differs depending on the communication structure currently being tested. In the general case, the Kali machine, camera, and phone are all connected to the same LAN. The solid black arrows show the direct communications between devices - as all devices are connected to the same LAN, all three of the devices are able to communicate with each other directly. The red dotted arrow indicates that in the case of communications directly between the phone and the camera, the laptop will attempt to intercept these communications. An external internet connection is provided in some cases, but is not required for the camera system to function.

This setup represents the format used for determining communication structures, but different structures were used during the attacking stage of the project as the

attacking stage requires communications between the laptop, camera, and any relevant servers, but does not require connection to the phone. Additionally, during the app decompilation stage, only the Kali machine is required as all of the software components needed for this stage are able to run locally on the laptop. Hence, the phone is not required to run the software. The camera is also not required as this stage does not require interaction with the hardware, and hence the ALFA network adapter is not necessary as no communications are being intercepted.

In the generic testing setup, the LAN is produced by either the camera itself, or by the Kali machine connected to the ALFA network adapter. The LAN produced by the camera can only be used if the camera has not been connected to an external network that is within range of the camera. If the camera is able to connect to an external network, it will automatically do so and will deactivate its own AP as part of the process. If no external network is available to the camera, the camera will automatically activate its own network. This can also be triggered manually by holding down the reset button on the camera which will force the camera to forget any external networks it was previously connected to.

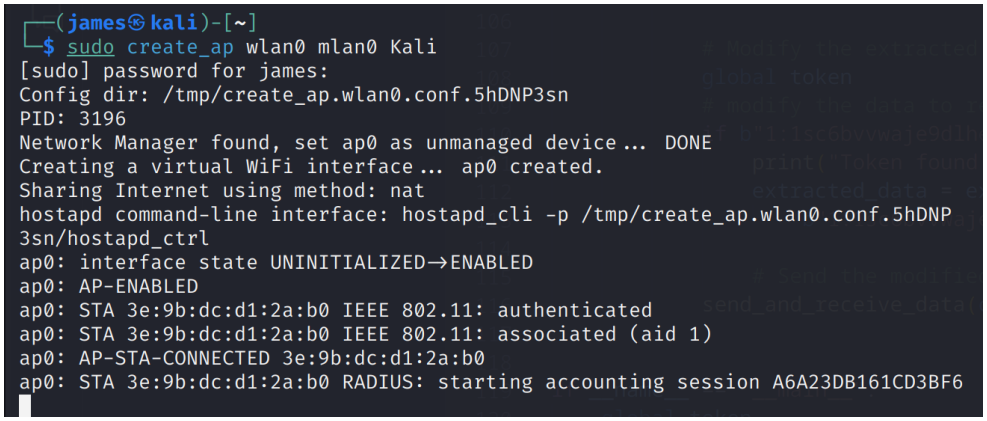


Figure 6.2: Creating an AP using the *create_ap* utility

In the second scenario, the AP is created using the *create_ap* tool. This is a command line utility that allows us to construct an AP using a regular network adapter, in this case the ALFA network adapter. The utility is built to use the hostapd tool which is a software tool for hosting APs on normal network cards [43] but hostapd has not been used directly as the *create_ap* tool simplifies this process.

Using the command command shown in figure 6.2, we create a network named Kali on the new software device ap0. An internet connection is able to be provided as this command creates a bridge between wlan0 (the ALFA network adapter running

the AP) and mlan0 (the network interface card (NIC) inbuilt into the laptop). The command outputs `ap0: AP-ENABLED` once the network is running. The subsequent information shown displays the mobile phone connecting to the Kali network. No password is set on this network, but the command could be altered to password protect the network if required.

```
┌──(james㉿kali)-[~]
└─$ iwconfig
lo        no wireless extensions.

mlan0     IEEE 802.11  ESSID:"SHELL-45BF08"
          Mode:Managed  Frequency:5.745 GHz  Access Point: 8C:FD:DE:45:BF:06
          Bit Rate=351 Mb/s   Tx-Power=5 dBm
          Retry short limit:9   RTS thr=2347 B   Fragment thr=2346 B
          Power Management:on
          Link Quality=46/70  Signal level=-64 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0

wlan0     IEEE 802.11  ESSID:off/any
          Mode:Managed  Access Point: Not-Associated   Tx-Power=20 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Power Management:off

ap0       IEEE 802.11  Mode:Master  Tx-Power=20 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Power Management:off
```

FIGURE 6.3: Checking the AP is running

We can verify that the network is running and functioning by using the command `iwconfig` or by using another device to search and attempt to connect to the network. Once the network is no longer needed, it can be shut down using `Ctrl + C` as shown in figure 6.4.

```
^Cap0: interface state ENABLED→DISABLED
ap0: AP-STA-DISCONNECTED 3e:9b:dc:d1:2a:b0

Doing cleanup.. ap0: AP-DISABLED
ap0: CTRL-EVENT-TERMINATING
nl80211: deinit ifname=ap0 disabled_11b_rates=0
done
```

FIGURE 6.4: Removing the AP once testing is finished

7 Application Decompilation

The process of reverse engineering and determining the communication structure began with decompiling the *HDlivecam* app. Decompiling is done to get a better understanding of how the app functions [44] and interfaces with the camera as well as any external servers it may require for authentication or command processing.

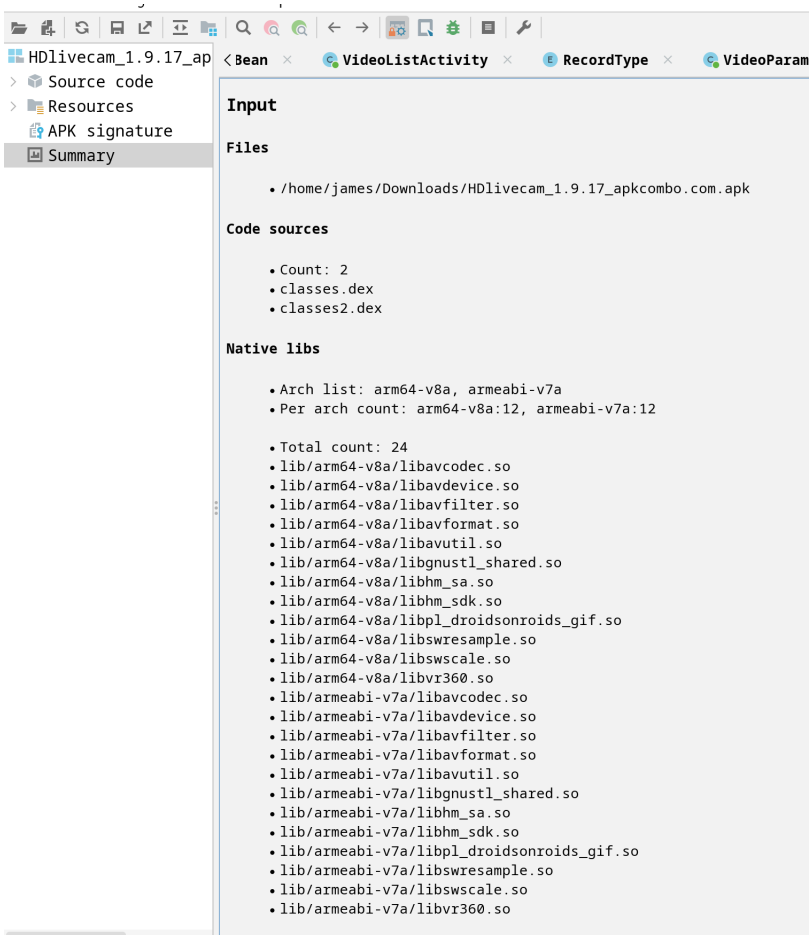


Figure 7.1: Decompilation summary of the app using *JADX*

Decompiling the app was done using the *JADX* tool. To do this, we require an APK of the app that we can obtain online from APKcombo [45]. After providing the APK, the software decompiles the application to return the Java source code.

Also provided is the directory structure of the app, along with the libraries the app uses.

Once the app has been decompiled a summary of the APK is generated including information such as the number of native libraries in the APK, and any errors and warnings that occurred during the decompilation. The code returned from this process was found to be heavily obfuscated. Obfuscation is a common security technique used to protect source code by giving methods and variables hard to read names such as random strings of characters. This results in the code being far harder for a potential attacker to decipher the function of as the names provided bear no relation to the purpose of the code. Other, more complex versions of obfuscation such as class encryption can be used to further obscure code [46], although not applied to the app being tested. As a result of this, an initial visual analysis of the code and file structure proved to be ineffective in determining the communication structure between the app and camera. However, this could be partially combated using the deobfuscation tool inbuilt into *JADX*. This improved the readability of some aspects of the code, although large portions remained unreadable. As a result, some files and folders were able to be identified for further analysis such as `okhttp3`.

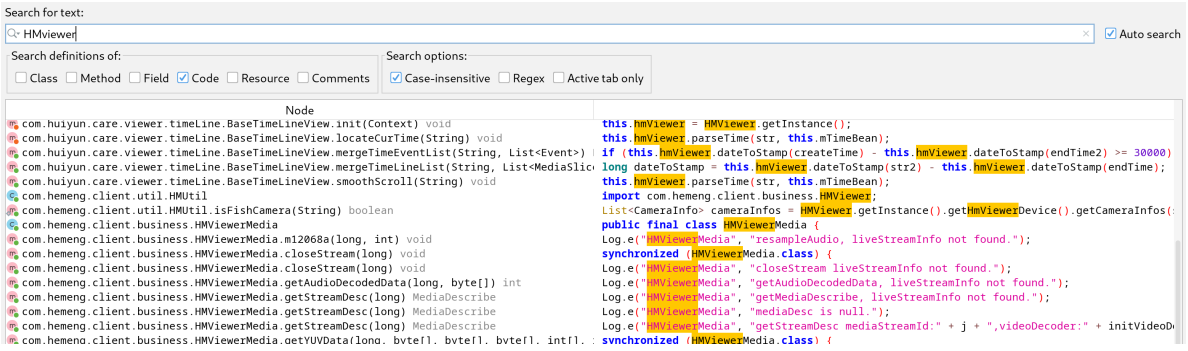


Figure 7.2: Performing a search in *JADX*

We can also perform a search to find references to specific keywords. From browsing the code, a reference to `HMviewer` was found which was suspected to be linked to the camera feed viewing capability of the app. Searching for references to this as in figure 7.2 allows us to view all of the other mentions of `HMviewer` in the code base. Looking at the context of the other references made, it confirms that this object is likely linked to the streaming of video from the camera to the app. In particular, this reveals a method labelled `HMViewerMedia.getYUVData`. The presence of this suggests that the video feed is encoded in the YUV colour space. YUV is a colour space that consists of a single luminance channel and two chrominance channels

and is an alternative way to represent image and video data to RGB [47].

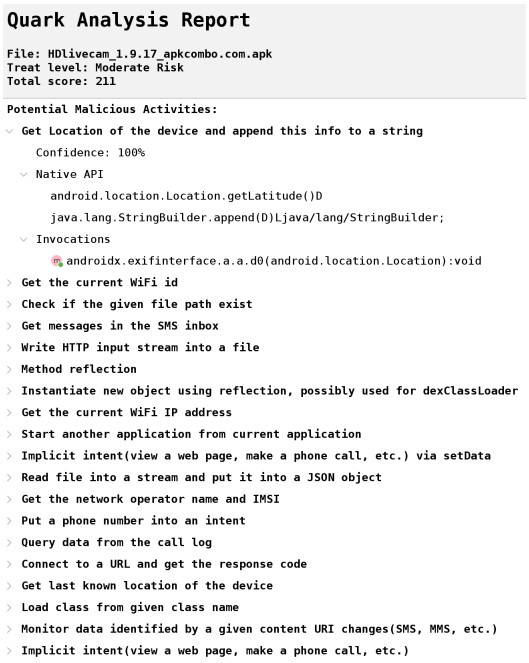


Figure 7.3: Quark analysis results

In addition to the visual inspection of the code, *JADX* provides an inbuilt tool called Quark-Engine [48] that was used to process the code to find additional information. This tool returns information about the predicted function of different portions of the code with a confidence level that the code performs the function predicted by Quark. Also returned is the location of that code which can be followed to perform additional manual analysis. Although this tool is primarily developed to detect malicious program behaviour, the list of suspected program behaviour aids in determining how the app functions.

From scanning the libraries used, several libraries titled `libac` were discovered which were suspected of handling the video stream encoding and decoding. We are not able to use *JADX* here as it is unable to decompile `.so` files. Therefore, in order to investigate further, we use *Ghidra*. *Ghidra* is a set of reverse engineering tools which are able to decompile the libraries used [49] and return their source code.

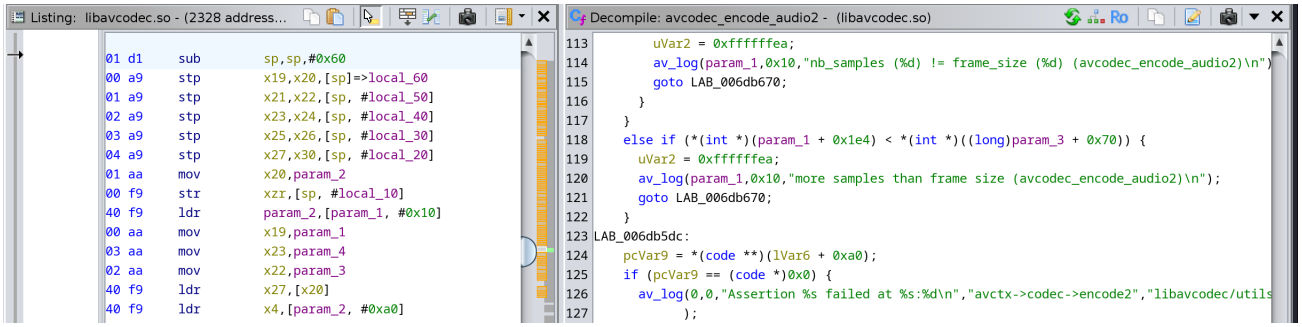


Figure 7.4: *Ghidra* code browser decompiled code

Opening the library in *Ghidra* code browser decompiles the library code into assembly code with an estimated conversion to C code as shown in figure 7.4. Examining the code produced reveals multiple references to encoding and frames, confirming that these libraries perform the encoding of the video feed in the app.

8 Communication Structure

Building upon the information obtained from the app decompilation stage, we can derive more information about the communication structure of the system by performing packet capturing and analysis. This is done using the setup outlined in chapter 6 with alterations made to match the different communication modes supported by the camera. In total three communication structures were discovered to be used by the system corresponding to the following:

- Direct communication over camera produced LAN.
- Same network communication over Kali produced LAN.
- Remote network communication.

To determine how the communication structures operate in more detail, we begin by running a scan to gain information about the camera and its network configuration. Ideally, technical documents or a white paper detailing the system information and communication methodology would be studied initially before performing any scans. However, no technical documentation about the camera or software could be found publicly available beyond the quick start guide for setting up the camera system and connecting it to the app. Hence, scanning formed the first component of communication analysis in this case. *NMAP* was used to perform this analysis of the camera; this is a 'free and open source utility for network discovery and security auditing' [50]. To perform the scan on the camera, we first need to recover the local IP address that the camera is using. Opening the app and navigating to the settings page for the camera allows us to open the network configuration page for the camera, revealing the local IP of the device. Using this we can then use the command shown in figure 8.1 to perform a scan on the device, the `-A` flag is used to prompt the software to search for additional information such as OS and script information; `--osscan-guess` allows for more aggressive guessing when trying to determine the OS of the device being scanned.

```
┌──(james㉿kali)-[~]
└─$ sudo nmap -A --osscan-guess 192.168.12.234
Starting Nmap 7.94 ( https://nmap.org ) at 2024-04-14 16:15 BST
Nmap scan report for 192.168.12.234
Host is up (0.0038s latency).
Not shown: 998 closed tcp ports (reset)
PORT     STATE SERVICE         VERSION
1234/tcp open  hotline?
1300/tcp open  h323hostcallsc?
MAC Address: DC:97:58:4A:A6:54 (Sichuan AI-Link Technology)
Device type: specialized|general purpose|webcam|VoIP phone
Running (JUST GUESSING): Espressif embedded (89%), lwIP 1.4.X (89%), Philips embedded (89%), NodeMCU
es embedded (88%), Ocean Signal embedded (88%)
OS CPE: cpe:/a:lwip_project:lwip cpe:/h:philips:hue_bridge cpe:/a:lwip_project:lwip:1.4 cpe:/o:nodemc
gies:dsg3060
Aggressive OS guesses: Espressif esp8266 firmware (lwIP stack) (89%), Philips Hue Bridge (lwIP stack
 (88%), Grandstream GXP1105 VoIP phone (88%), lwIP 1.4.0 lightweight TCP/IP stack (88%), Rigol DSG306
, ESPEasy OS (lwIP stack) (87%), Revo Blik Wi-Fi Internet radio (87%)
No exact OS matches for host (If you know what OS is running on it, see https://nmap.org/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=7.94%E=4%D=4/14%OT=1234%CT=1%CU=32533%PV=Y%DS=1%DC=D%G=Y%M=DC9758
OS:%TM=661BF3BB%P=x86_64-pc-linux-gnu)SEQ(SP=3C%GCD=1%ISR=8B%TI=I%CI=I%II=R
OS:I%SS=O%TS=U)SEQ(SP=3E%GCD=1%ISR=8B%TI=I%CI=I%II=RI%SS=O%TS=U)SEQ(SP=40%G
OS:CD=1%ISR=8B%TI=I%CI=I%II=RI%SS=O%TS=U)SEQ(SP=41%GCD=1%ISR=8B%TI=I%CI=I%I
OS:I=RI%SS=O%TS=U)OPS(O1=M5B4%O2=M5B4%O3=M5B4%O4=M5B4%O5=M5B4%O6=M5B4)WIN(W
OS:1=FFFF%W2=FFFF%W3=FFFF%W4=FFFF%W5=FFFF%W6=FFFF)ECN(R=Y%DF=N%T=C8%W=FFFF%
OS:O=M5B4%CC=N%Q=)T1(R=Y%DF=N%T=C8%S=O%A=S+%F=AS%RD=0%Q=)T2(R=N)T3(R=Y%DF=N
OS:%T=C8%W=FFFF%S=O%A=S+%F=AS%O=M5B4%RD=0%Q=)T4(R=Y%DF=N%T=C8%W=FFFF%S=A%A=
OS:S%F=AR%O=%RD=0%Q=)T5(R=Y%DF=N%T=C8%W=FFFF%S=A%A=S+%F=AR%O=%RD=0%Q=)T6(R=
OS:Y%DF=N%T=C8%W=FFFF%S=A%A=S%F=AR%O=%RD=0%Q=)T7(R=Y%DF=N%T=C8%W=FFFF%S=A%A
OS:=S+%F=AR%O=%RD=0%Q=)U1(R=Y%DF=N%T=C8%IPL=38%UN=0%RIPL=G%RID=G%RIPCK=G%RU
OS:CK=G%RUD=G)IE(R=Y%DFI=S%T=C8%CD=S)

Network Distance: 1 hop

TRACEROUTE
HOP RTT     ADDRESS
1   3.82 ms 192.168.12.234

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 188.21 seconds
```

FIGURE 8.1: Running a detailed *NMAP* scan on the IP address of the camera

The result of the scan reveals several open ports, the device media access control (MAC) information, OS and device information, the TCP/IP fingerprint, and traceroute information. From the output of the command we can determine that the camera has two ports open using the TCP protocol including `h323hostcallsc?` running on port 1300 likely corresponding to the `H.323 Host Call Secure service`. This is a set of protocols that is used for control over multimedia communications such as voice over IP (VoIP) [51]; hence this may be used in the camera to control the communication of video and audio. Additionally, the MAC address shows that the network hardware is provided by `Sichuan AI-Link Technology`, a Chinese company that specialises in internet modules for IoT devices [52].

Multiple guesses at the OS running on the device were also made, although *NMAP* was not able to find any exact match. Despite no exact match being found, there are multiple guesses returned that provide a close match of over 85%. Hence, it is likely that the camera uses either a niche OS or a custom version of software similar to that of the guesses provided. Looking for further information in the TCP/IP

fingerprint shows the software to be a version of `x86_64 Linux`.

```
┌──(james㉿kali)-[~]
└─$ sudo nmap -sS -Pn -p1-65535 192.168.12.234
Starting Nmap 7.94 ( https://nmap.org ) at 2024-04-14 17:04 BST
Nmap scan report for 192.168.12.234
Host is up (0.057s latency).
Not shown: 65529 closed tcp ports (reset)
PORT     STATE SERVICE
1234/tcp open  hotline
1300/tcp open  h323hostcallsc
6980/tcp open  qolyester
7847/tcp open  csoauth
8554/tcp open  rtsp-alt
8699/tcp open  vnyx
MAC Address: DC:97:58:4A:A6:54 (Sichuan AI-Link Technology)

Nmap done: 1 IP address (1 host up) scanned in 73.35 seconds
```

FIGURE 8.2: *NMAP* scan of additional ports

The initial scan does not cover the full range of ports but this can be extended manually using the `-p` flag to the full port range to discover any additional services running on less common port number. We do this in a separate command as the flags used in the previous command cause the scan to require substantially more time even over a small port range. Extending the range to cover the full port range reveals several other services with open ports on the camera. Notably, a service named `rtsp-alt` is shown as running on port 8554 indicating that the camera may be transferring video using the RTSP protocol. This is a protocol commonly used for streaming multimedia. Attempts were made to connect to connect to the RTSP service both using *VLC media player* [53] and a web browser. However, no connection was able to be established.

Using the understanding of the networking services gained from the scan, we can use packet capture and analysis to derive how the camera and app interface and transfer data. We do this by intercepting the communications that are sent to and from the devices as well as to any remote servers that either device communicates with. By collecting this data over multiple communications in different scenarios, we can examine the packet information to extract details about the data flows involved in operating the camera system. With this information, we can begin to build a graphical representation of how the communication system operates.

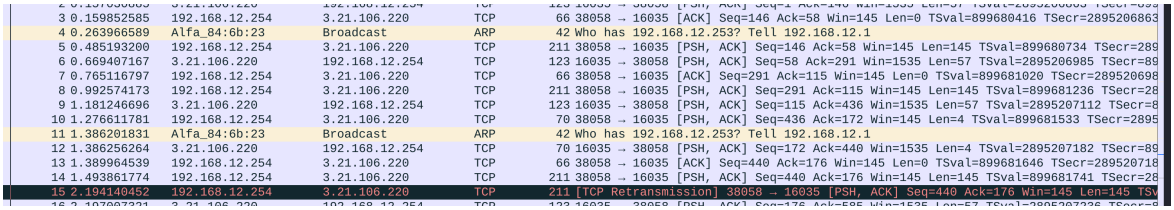


FIGURE 8.3: Example *Wireshark* capture of packets sent between the camera and the phone

Wireshark is the utility primarily used to perform this capture as it provides detailed information about the captured packets. Once a capture has been performed, a chronological list of packets will be shown as in figure 8.3. This contains information such as the packet number, source and destination IP addresses, and a summary of the function of the packet.

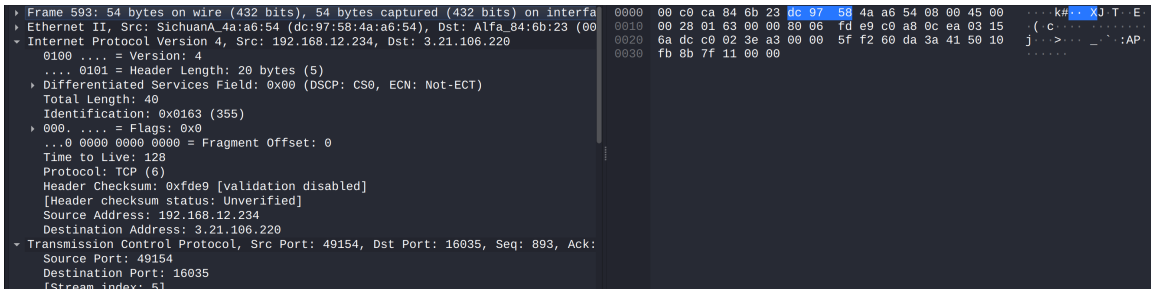


FIGURE 8.4: Packet details in *Wireshark*

Selecting an individual packet provides more detailed information about that particular packet such as the source and destination ports. Hexadecimal and ASCII representations of the raw packet data are also shown. This information can aid in identifying the purpose of the packets. *Wireshark* also allows for additional filtering and generation of graphics and metrics about the captured packets. Using these features, the packets specific to the camera system can be filtered for by specifying that the IP source or destination must include either the address of the phone or the camera. These addresses are known to be `192.168.12.253` and `192.168.12.234` respectively when the devices are both connected to the AP generated by the Kali machine. However, this would change if testing on a different network as networks commonly assign addresses using dynamic host configuration protocol (DHCP) [54]. As this filter will include all packets to and from the phone, it is still important to manually examine the resulting packets to extract only those which are related to the camera system communications.

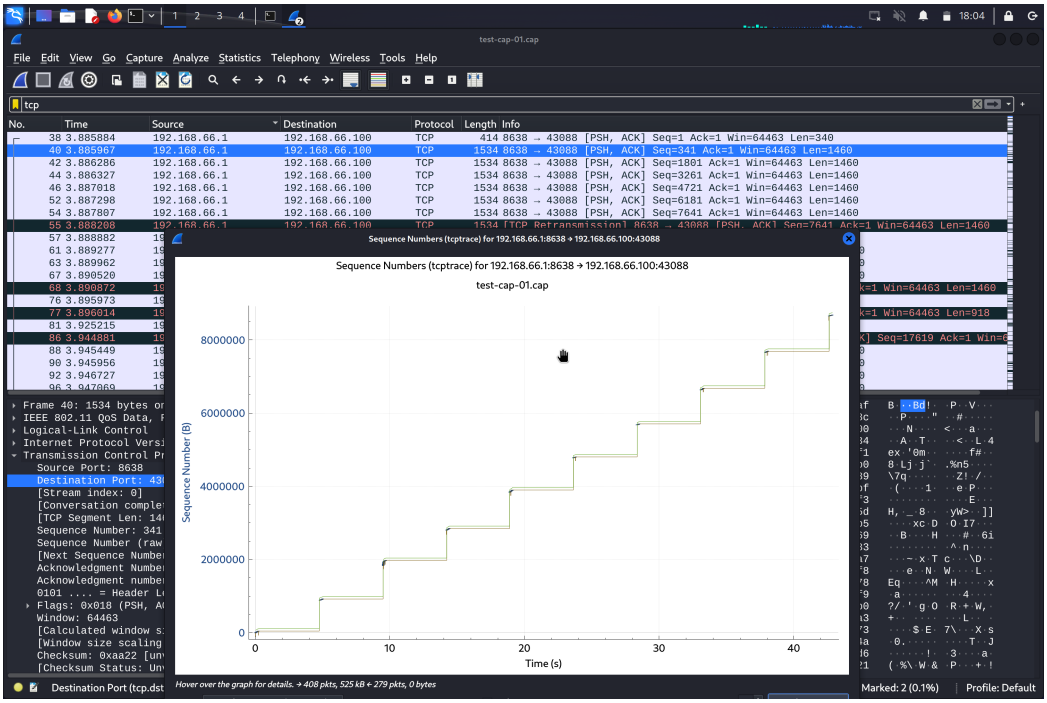


FIGURE 8.5: Graphical representation of the TCP stream between the phone and the camera

Using the filters it is revealed that a large proportion of the communications involving the camera or the phone are over TCP. These packets are of consistent size and being sent in large quantities, indicating they may be related to the video streaming feature of the camera. Visualising the TCP stream graphically supports this theory as data is being sent at a consistent rate. All communication occurring over TCP is further supported by the open ports discovered using *NMAP*. When analysing the contents of the packets, we can extract each stream individually using the follow TCP stream feature which isolates a single stream and provides an ASCII representation of all the data sent and received over that stream.

8.1 Direct communication structure

The first communication structure discovered was the direct structure. This is the format used in the setup of the camera whereby the camera produces an AP and the phone connects directly to it.

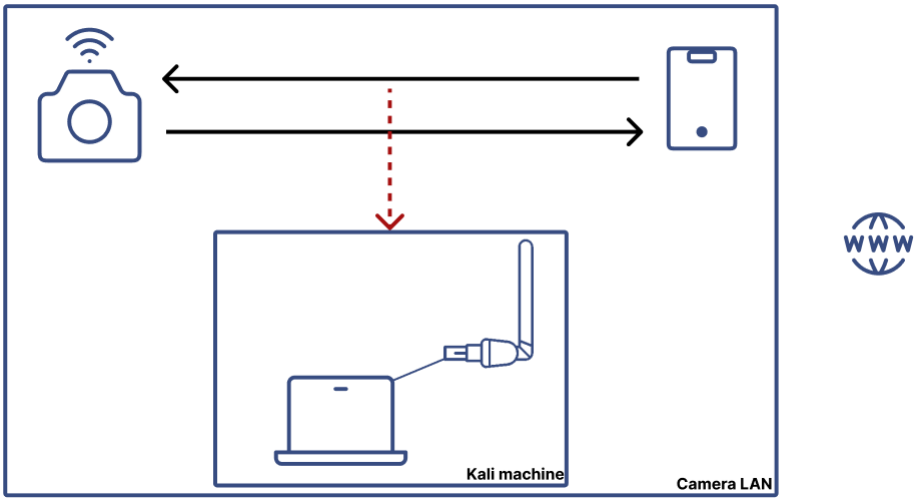
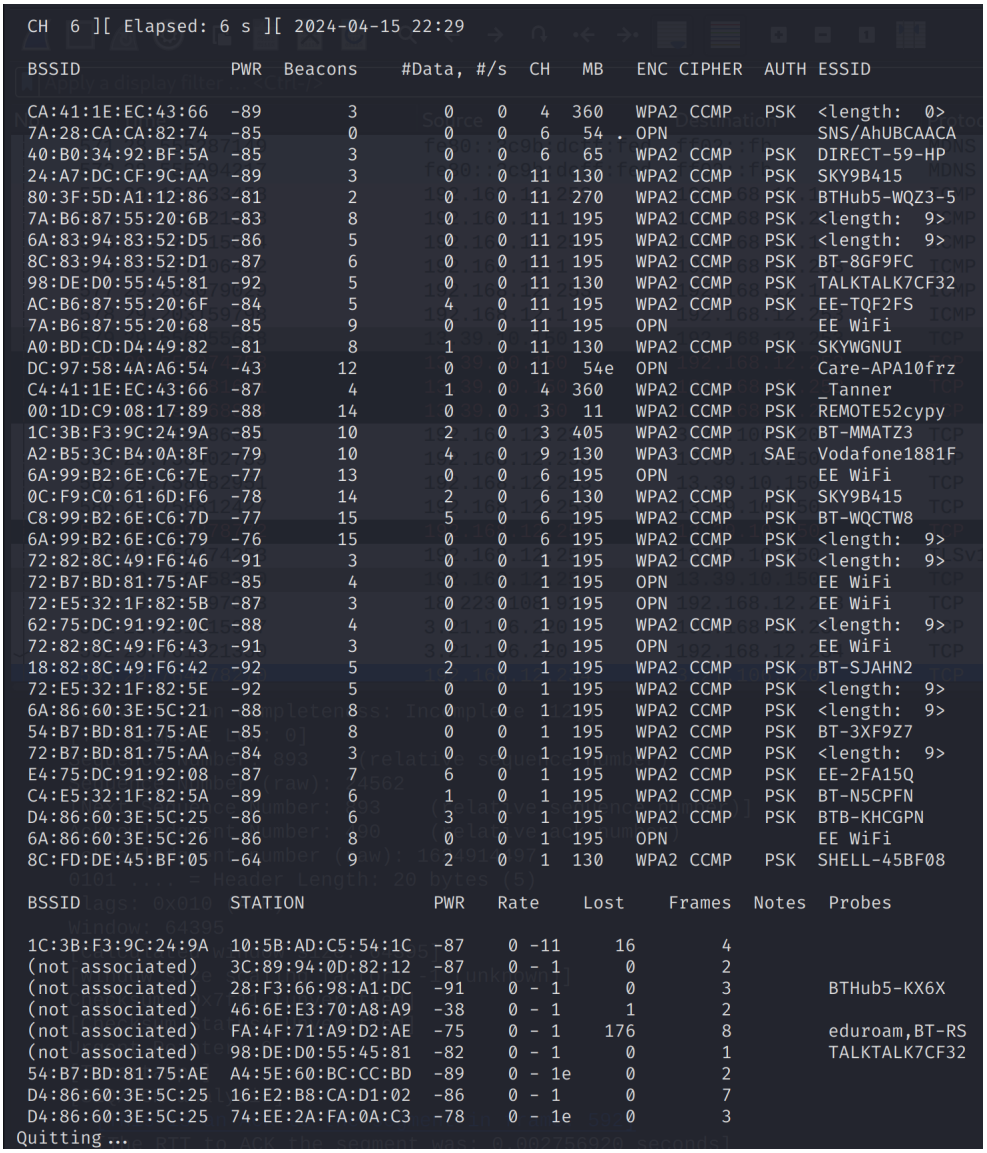


Figure 8.6: Direct communication structure

Over this method of communication, to attempt to capture packets, the Kali machine was first connected to the network produced by the camera. As the camera network is open, no authentication is required to do this. Subsequently *Wireshark* was run on the Kali machine whilst the video feed was being streamed to the phone in an attempt to capture packets. However, no packets were able to be captured using this method even once an additional attempt was made after setting the wireless adapter to promiscuous mode manually. This could suggest that the network created by the camera is switched and thus is only sending the packets directly to the recipient device [55], preventing the Kali machine from intercepting the packets.



```
 CH  6 ][ Elapsed: 6 s ][ 2024-04-15 22:29

 BSSID              PWR  Beacons    #Data, #/s  CH   MB   ENC CIPHER  AUTH ESSID

 CA:41:1E:EC:43:66  -89        3        0    0   4  360   WPA2 CCMP   PSK  <length:  0>
 7A:28:CA:CA:82:74  -85        0        0    0   6   54 . OPN              SNS/AhUBCAACA
 40:B0:34:92:BF:5A  -85        3        0    0   6   65   WPA2 CCMP   PSK  DIRECT-59-HP
 24:A7:DC:CF:9C:AA  -89        3        0    0  11  130   WPA2 CCMP   PSK  SKY9B415
 80:3F:5D:A1:12:86  -81        2        0    0  11  270   WPA2 CCMP   PSK  BTHub5-WQZ3-5
 7A:B6:87:55:20:6B  -83        8        0    0  11  195   WPA2 CCMP   PSK  <length:  9>
 6A:83:94:83:52:D5  -86        5        0    0  11  195   WPA2 CCMP   PSK  <length:  9>
 8C:83:94:83:52:D1  -87        6        0    0  11  195   WPA2 CCMP   PSK  BT-8GF9FC
 98:DE:D0:55:45:81  -92        5        0    0  11  130   WPA2 CCMP   PSK  TALKTALK7CF32
 AC:B6:87:55:20:6F  -84        5        4    0  11  195   WPA2 CCMP   PSK  EE-TQF2FS
 7A:B6:87:55:20:68  -85        9        0    0  11  195   OPN              EE WiFi
 A0:BD:CD:D4:49:82  -81        8        1    0  11  130   WPA2 CCMP   PSK  SKYWGNUI
 DC:97:58:4A:A6:54  -43       12        0    0  11   54e  OPN              Care-APA10frz
 C4:41:1E:EC:43:66  -87        4        1    0   4  360   WPA2 CCMP   PSK  _Tanner
 00:1D:C9:08:17:89  -88       14        0    0   3   11   WPA2 CCMP   PSK  REMOTE52cypy
 1C:3B:F3:9C:24:9A  -85       10        2    0   3  405   WPA2 CCMP   PSK  BT-MMATZ3
 A2:B5:3C:B4:0A:8F  -79       10        4    0   9  130   WPA3 CCMP   SAE  Vodafone1881F
 6A:99:B2:6E:C6:7E  -77       13        0    0   6  195   OPN              EE WiFi
 0C:F9:C0:61:6D:F6  -78       14        2    0   6  130   WPA2 CCMP   PSK  SKY9B415
 C8:99:B2:6E:C6:7D  -77       15        5    0   6  195   WPA2 CCMP   PSK  BT-WQCTW8
 6A:99:B2:6E:C6:79  -76       15        0    0   6  195   WPA2 CCMP   PSK  <length:  9>
 72:82:8C:49:F6:46  -91        3        0    0   1  195   WPA2 CCMP   PSK  <length:  9>
 72:B7:BD:81:75:AF  -85        4        0    0   1  195   OPN              EE WiFi
 72:E5:32:1F:82:5B  -87        3        0    0   1  195   OPN              EE WiFi
 62:75:DC:91:92:0C  -88        4        0    0   1  195   WPA2 CCMP   PSK  <length:  9>
 72:82:8C:49:F6:43  -91        3        0    0   1  195   OPN              EE WiFi
 18:82:8C:49:F6:42  -92        5        2    0   1  195   WPA2 CCMP   PSK  BT-SJAHN2
 72:E5:32:1F:82:5E  -92        5        0    0   1  195   WPA2 CCMP   PSK  <length:  9>
 6A:86:60:3E:5C:21  -88        8        0    0   1  195   WPA2 CCMP   PSK  <length:  9>
 54:B7:BD:81:75:AE  -85        8        0    0   1  195   WPA2 CCMP   PSK  BT-3XF9Z7
 72:B7:BD:81:75:AA  -84        3        0    0   1  195   WPA2 CCMP   PSK  <length:  9>
 E4:75:DC:91:92:08  -87        7        6    0   1  195   WPA2 CCMP   PSK  EE-2FA15Q
 C4:E5:32:1F:82:5A  -89        6        1    0   1  195   WPA2 CCMP   PSK  BT-N5CPFN
 D4:86:60:3E:5C:25  -86        6        3    0   1  195   WPA2 CCMP   PSK  BTB-KHCGPN
 6A:86:60:3E:5C:26  -86        8        0    0   1  195   OPN              EE WiFi
 8C:FD:DE:45:BF:05  -64        9        2    0   1  130   WPA2 CCMP   PSK  SHELL-45BF08

 BSSID              STATION            PWR   Rate    Lost    Frames  Notes  Probes

 1C:3B:F3:9C:24:9A  10:5B:AD:C5:54:1C  -87    0 -11     16        4
 (not associated)   3C:89:94:0D:82:12  -87    0 - 1      0        2
 (not associated)   28:F3:66:98:A1:DC  -91    0 - 1      0        3         BTHub5-KX6X
 (not associated)   46:6E:E3:70:A8:A9  -38    0 - 1      1        2
 (not associated)   FA:4F:71:A9:D2:AE  -75    0 - 1    176        8         eduroam,BT-RS
 (not associated)   98:DE:D0:55:45:81  -82    0 - 1      0        1         TALKTALK7CF32
 54:B7:BD:81:75:AE  A4:5E:60:BC:CC:BD  -89    0 - 1e     0        2
 D4:86:60:3E:5C:25  16:E2:B8:CA:D1:02  -86    0 - 1      0        7
 D4:86:60:3E:5C:25  74:EE:2A:FA:0A:C3  -78    0 - 1e     0        3
Quitting ...
```

Figure 8.7: *Airodump* capture output

To resolve this, a different packet capturing process was used. First the Alfa network adapter was set to monitor mode, allowing it to capture all packets that are being transmitted [56]. The adapter is set to monitor mode using the sequence of commands:

```
sudo ifconfig wlan0 down
sudo iwconfig wlan0 mode monitor
sudo ifconfig wlan0 up
iwconfig
```

The final `iwconfig` command simply provides a check that the previous commands correctly set the mode of the adapter. With the adapter in this mode, the *airodump-*

ng tool was used to sniff packets and enumerate all of the networks that the adapter could detect [57]. Using the name of the network as reference, the basic service set identifier (BSSID) was found in the list of networks detected along with the channel being communicated over. Using this information as a filter, a new capturing session was started and set to write to a file; this captures only packets from the network hosted by the camera as the BSSID is a unique identifier for the network [58]. The saved file was then opened in *Wireshark*. This allows us to collect and read information about the packets sent. However, a limitation of this method is the `.cap` format of the output files. Thus, less information about the packets is available compared to the `.pcapng` files normally captured by *Wireshark*. Additionally, some of the filtering and visualisation tools are not supported in this format.

Manually reviewing the packets captured allows us to derive the communication structure shown in figure 8.6. As was discovered during the setup process, the camera produces the AP in this structure, with the phone and the Kali machine connecting to its LAN. All of the communications are sent directly between the phone and the camera where the Kali machine uses monitor mode to intercept packets. In this setup, no external internet access is available. Hence, for the phone to communicate with the camera, it must be within range of the camera AP. Camera functionality in this mode is also limited to basic features. This mode appears intended primarily just for the setup process of the camera, and to provide a redundancy in the event of the camera being placed in a location without external WiFi networks. It is unlikely that this mode would be used in the general operation of the camera and thus the subsequent vulnerabilities tested do not focus on this method of communication. Such a method of operation is additionally less than optimal for the specific application of a hidden camera due to the open network being publicly broadcast which may reveal the presence of the device to a user searching WiFi networks even if they were not specifically looking for the camera network.

8.2 Same network communication structure

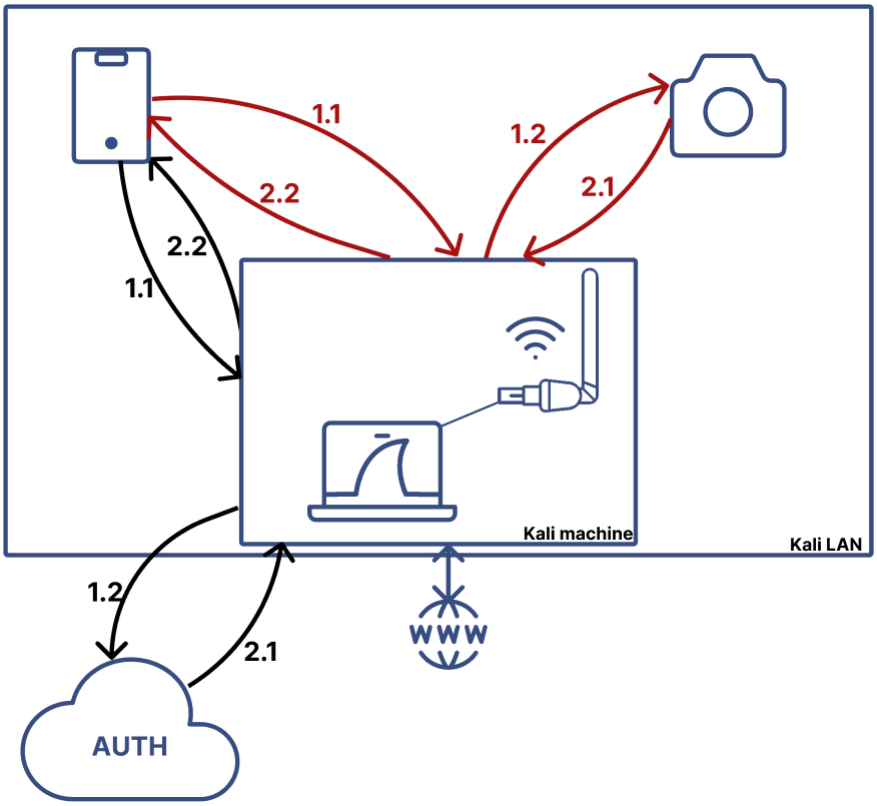


Figure 8.8: Same network communication structure

In the same network communication structure, the Kali machine produces the network which both the phone and the camera are connected to. The camera is connected to this network using the network options in the *HDlivecam* app. As the Kali machine is creating the AP, *Wireshark* can be used to capture communications directly as opposed to having to use the alternative capturing method used in section 8.1. We can use this method as the Kali machine acts as a router in this structure. Hence, all communications between the devices on the network are passed through the Kali machine on the path to their destination. As a result, the Kali machine is able to capture all communication information in the more detailed `.pcapng` format without the need for monitor mode.

With this structure, an internet connection was provided by the Kali network, allowing the full set of features provided by the app to be used. From examining the packets, we can determine that there are two main communication streams involving the camera system. These are those shown in red and black in figure 8.8. The red stream details the communication process between the camera and the phone.

This includes the streaming of the video feed from the camera to the phone, as well as sending commands from the phone to the camera which returns a response when the command has been executed.

The stream in black shows the communication process that occurs when the app is started. When the app is started, the phone establishes a connection to the authentication server at `18.218.23.115:16035`. If the user already has an account logged into the app, this is automatically authenticated with the server and the user is taken to the app homepage as shown in figure 5.2. If the user does not have an account created, the requests required to make a new account are sent to the authentication server to perform this process. Once the account has been created, the authentication server returns success and automatically logs the user in. If packet capturing was not required, the Kali machine could be replaced by a regular AP.

FIGURE 8.9: Additional remote connections established by the phone

Aside from the primary two streams discussed, the app additionally establishes several other connections to external servers when it is launched. We can filter to find these additional connections by following individual TCP streams. These external connections do not transfer large amounts of data and appear to primarily be to certificate authorities and server or service providers.

Unlike the direct communication structure, this structure is more likely to be used in a normal usage scenario for an IP camera. One such scenario where this structure may be used would be in a home surveillance application. Here the camera would be connected to the home WiFi network with the phone connected to the same network. However, the communication between the camera and the phone is not 100% reliable in this structure. Therefore, sometimes the system will switch to the different networks communication structure as shown in section 8.3 even when the camera and the phone are connected to the same LAN.

8.3 Different networks communication structure

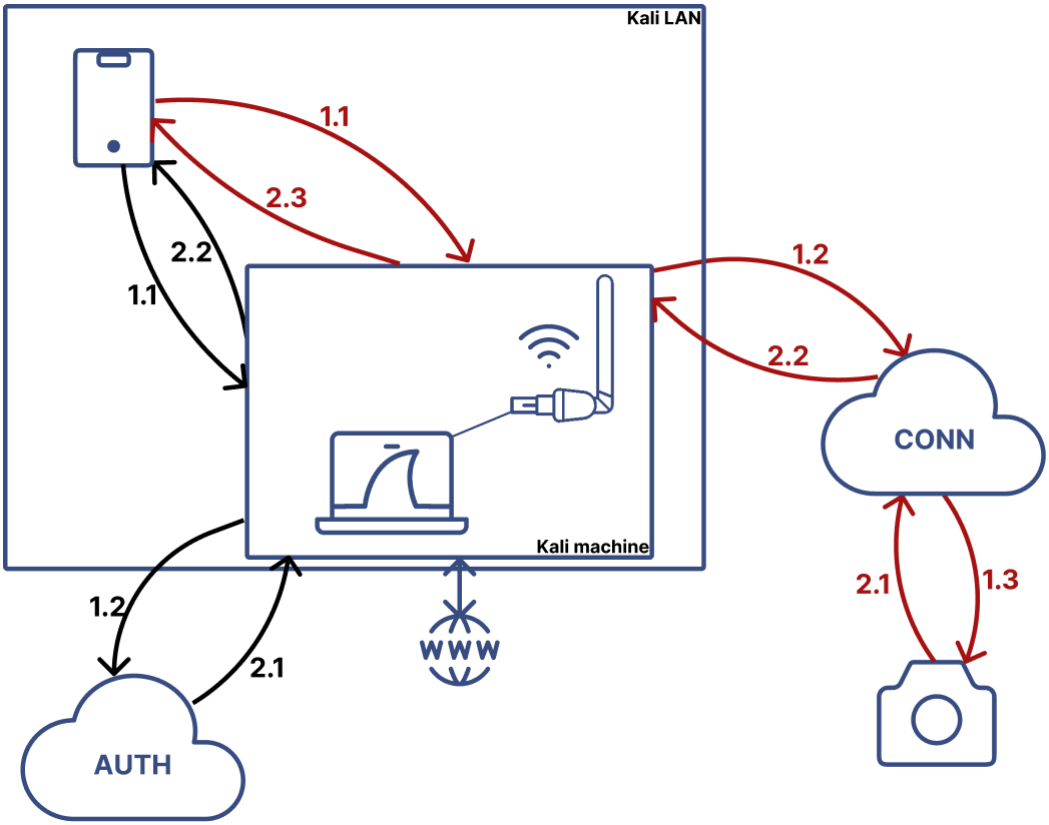


Figure 8.10: Different networks communication structure

Similarly to the same network communication structure, the different networks communication structure provides full functionality over the camera controls. This communication structure also operates with the Kali machine creating the AP and the phone connecting to this network. However, in this structure, the camera is not connected to the same network.

Again there are two primary streams present for the communication between the authentication server and phone, and communication between the camera and the phone. In this structure, as the phone and camera are on different networks, communications between these devices are handled via a connection server which acts as a relay between the two devices. When communicating with the camera the phone never addresses messages with the IP of the network that the camera is connected to. Instead, messages are addressed to a specific IP and port number combination belonging to the connection server. A security token is also required

in this communication or the connection server will reject the connection to the camera. If all information is correct, communication will be established to the camera and messages will be forwarded. For the app to gather the required information it first communicates with the authentication server. This server still handles login requests as in the previous structure. However, in the different networks structure, when the phone wishes to connect to the camera it first sends a request to the authentication server which returns the correct IP address, port number, and security token which the app then forwards to the connection server to establish communication.

Of all the three communication structures, the different networks structure provides the most versatility in a general usage scenario involving the camera. This structure would be employed in any scenario where the two devices are connected to different networks, likely for remote access of the camera feed. One such example would be if a user had set up a camera in their house; they would be able to access the camera using this communication structure even if they were not at home.

9 Vulnerabilities

With an understanding of the communication methods obtained, we can examine the captured communications to enumerate potential vulnerabilities within the communication system. As we enumerate the potential vulnerabilities, we can perform attacks to verify the presence of weaknesses within the security of the system. The most severe of the vulnerabilities discovered and tested allow for complete compromise of the command and control system of the camera, verified by successfully switching off the indicator LED to demonstrate that command execution has been performed.

9.1 Unencrypted communications

Using the captured communications, we can filter and follow specific TCP streams in *Wireshark* to isolate those that contain the camera communications. Examining these shows that the communications occur in plaintext rather than being encrypted. This allows us to view all the packet data in ASCII format; hence we are able to read the contents of the packets sent.

Figure 9.1: Transmission header for the video feed communication stream

Unencrypted transmissions allows us to determine which feed contains the video stream. Shown in figure 9.1 we can observe that the beginning of the communication stream contains variables relating to the state of the camera. The communications highlighted in blue are those sent from the camera, with those displayed in red sent from the phone. Additionally included from the phone is

`HM_DESCRIBE_LIVE_VERSION=1.0`; this likely refers to the software used to decode and render the video feed in the app. The parameters `streamid` and `camid` confirm that this is the TCP stream over which the video feed is transmitted. Subsequent to the header shown in figure 9.1, the remainder of the communication contains the ASCII decoding of the video feed. Using *Wireshark*, the raw data from the packets in this stream are able to be exported as a binary file. If the encoding and decoding scheme for the video was reverse engineered or discovered in the source code of the app, it would likely be possible to reconstruct the captured video feed using the binary file containing the data from the capture. Although this was not tested in this project, such an attack has a high probability of being successful due to the unencrypted nature of the capture and detailed parameters regarding the stream structure such as resolution being included in the header.

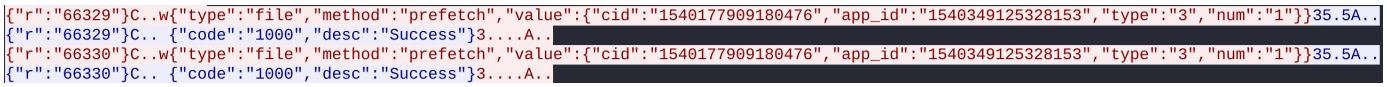


FIGURE 9.2: JavaScript object notation (JSON) requests and responses

The same process of stream analysis was extended to other communications relating to the camera, revealing that the devices communicate both with each other and with the remote servers using JavaScript object notation (JSON). As these JSON communications are also sent in plaintext, it is easy to determine what information is transferred during requests and responses between the devices. Specifically, as shown in figure 9.2, we can determine that requests are numbered sequentially, and a return code of 1000 indicates success. If an attacker was able to intercept these communications for example by packet sniffing on the same network as the user, they would be able to recover this information easily. If the user was using the app on an open network such as public WiFi this would be especially dangerous as the attacker would not need authentication to access the network before performing the attack.

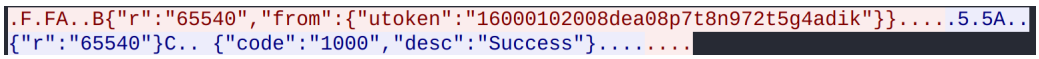


FIGURE 9.3: Plaintext capturing of the user login token

Plaintext communications extend throughout all of the communications, including those to the external servers. Subsequently more sensitive user information is also vulnerable to being captured in plaintext format such as the user token as shown in figure 9.3. This token is unique and associated to an individual user account.

Upon startup of the app, the user token is transmitted to the authentication server and used to validate the app login. As the user token is static and does not change per session, a single capture of the token can permanently compromise the security of the account associated with the token.


....[1 bytes missing in capture file].....QA..
{"r":"65604"}C...{"app_id":"1540349125328153","product_id":"1540178842268459","company_id":"1540177909180476","account":"testproj2@gmail.com","type":"1","password":"testpass1","name":"testproj2@gmail.com"}.5.5A..
{"r":"65604"}C.. {"code":"1000","desc":"Success"}....A..
{"r":"65605"}C..u{"app_id":"1540349125328153","product_id":"1540178842268459","company_id":"1540177909180476","push_token":"ejQrZkTWQzG8VymJ7A1-10:APA91bFn6H_HMSIyjNCNpm0ksDVY39wyigcIMwf-857Xf9lc3XOM6CyHJrkbISGYygfm67g1sHvywYtf53emIzNwFwn1bnmxsru0fyFFnX6zbDNhhOycQRN1CslU3dKcnHguT8uVKblz","push_platform":"7","account":"testproj2@gmail.com","password":"testpass1","type":"1","language":"2"}....[A..
{"r":"65605"}C...{"code":"1000","data":{"email":"testproj2@gmail.com","name":"testproj2@gmail.com","srcid":"1600010280923b8el8p","token":"1600010280923b8epe4fbq56w9o8ihhg","uid":"1600010280923b8e"},"desc":"Success"}.F.FA..B{"r":"65606","from":{"utoken":"1600010280923b8epe4fbq56w9o8ihhg"}}.5.5A..
{"r":"65606"}C.. {"code":"1000","desc":"Success"}.?.?A..
{"r":"65608"}C..*{"uid":"1600010280923b8e","type":"smsnum"}....A..

FIGURE 9.4: Capturing user email and password information

If a capture occurs during the sign up or sign in process from the logged out state as shown in figure 5.3, the email address and password information that the user submits to the app will also be captured. This capture is less likely to be obtained by a malicious actor as, if the user already has an account, the account is automatically logged in through solely the user token, thus this information is never transmitted. However, if this capture is obtained, the consequences of this information disclosure are greater than the other examples as the disclosed information entails practically significant user details, extending beyond the context of the camera system. Particularly if the user utilises the same email and password combination for other software would this compromise prove especially dangerous.

9.2 SERVER AUTHENTICATION

As login information is able to be compromised and retrieved in plaintext from the previously captured communications, we can attempt to use the login information obtained to impersonate the user and login with the authentication server. If the full login information from figure 9.4 is able to be captured, we can simply install a legitimate copy of the *HDlivecam* app on another device and input the login information to gain complete control over the camera and user account. We are able to do this as the login system does not implement any form of multi-factor authentication.

If only the user token is captured, we are not able to gain direct access to the mobile app. Instead we can perform manual authentication with the server by using the captured token to emulate a legitimate login request.

```
┌──(james㉿kali)-[~/3rdyrproj]
└─$ sudo tcpreplay -i ap0  captures/commands/justlogin.pcap
Actual: 1 packets (144 bytes) sent in 0.000009 seconds
Rated: 16000000.0 Bps, 128.00 Mbps, 111111.11 pps
Flows: 1 flows, 111111.11 fps, 1 unique flow packets, 0 unique non-flow packets
Statistics for network device: ap0
        Successful packets:        1
        Failed packets:            0
        Truncated packets:         0
        Retried packets (ENOBUFS): 0
        Retried packets (EAGAIN):  0
```

FIGURE 9.5: Attempting to replay the login packets

Initial attempts to perform this authentication were performed using *Tcpreplay*, a program that allows captured `.pcap` files to be replayed to their original destination [59]. This was attempted with both the individual packet containing token, and by replaying the entire communication stream from the phone IP up until the point of the user token being transmitted. In both scenarios the packets are sent and displayed as being successfully delivered. However, running a *Wireshark* capture as the packets are sent reveals that the attempts to authenticate with the server are unsuccessful. In both cases, the server either does not provide a response or requests a reset by returning a RST flag. This occurs due to the connection oriented nature of TCP. Using this communication protocol, packets are numbered and timestamped. As a result the server is able to detect that the attempted login is not authentic and hence rejects the authentication request.

To resolve the issues encountered, we first need to establish an independent connection to the server before sending the data required to initiate a login. By doing so, we avoid any issues as a result any issues dealing with packet numbering and timestamping. Using the information from the captured communications, we know the IP address and port number of the authentication server and hence can manually establish a connection to that address

We do this using a *Python* script to initially perform a TCP handshake with the server using sockets. Once a connection to the server has been established we then extract the data to be sent from the packets in the `.pcap` file. Using the *scapy* module, we can extract the data from the login sequence including the user token at a per packet scale. Each packet of data is then sent to the server, with the *Python* script waiting for a response from the server after each packet before sending the subsequent packet.

```
┌──(james㉿kali)-[~/3rdyrproj]
└─$ sudo python3 connection.py
Connected to 18.218.23.115:16035
Sent data: b'\x02F\x00FA\x00\x00B{"r":"65540","from":{"utoken":"1600010280968aaamw09g3i8bzqgd9d6"}}\x1d\x00\x00\x00'
Received data: b'\x025\x815A\x00\x00\r{"r":"65540"}C\x00\x00 {"code":"1000","desc":"Success"}'
```

FIGURE 9.6: Authenticating with the server using a *Python* script and the user token

Running the script results in the response {"code":"1000","desc":"Success"}. This indicates that the attacking program has successfully authenticated with the server as the returned response is identical to the response returned when the app submits a genuine login request. As authentication is only required for the initial login, a malicious user would now be able to communicate freely with the server by sending the appropriate JSON messages.

9.3 SAME NETWORK COMMAND EXECUTION

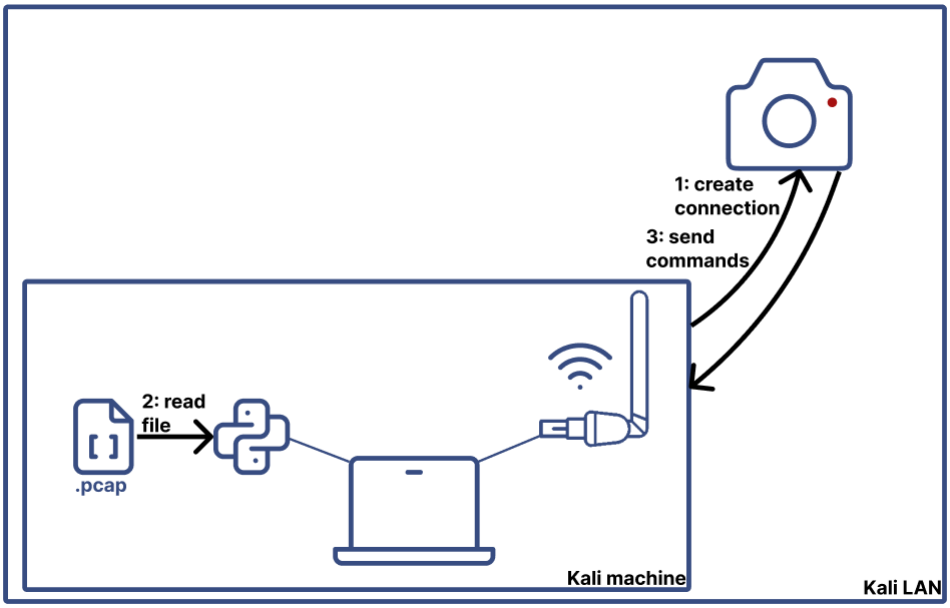


FIGURE 9.7: Command execution attack process with both devices on the same network

Building upon the method used to perform the server authentication attack, we can perform a command execution attack to turn off the indicator LED when the camera and phone are connected to the same network as described in section 8.2. The same network communication structure must be used as this allows the phone and camera to communicate directly. This attack can be achieved with only minor modification to the `connection.py` *Python* program that was used to perform

server authentication.

For the command execution attack to be performed, we first require a capture of the TCP stream between the phone and the camera whilst the system is using the same network communication structure including a command being sent. We then save this portion of the communication sent from the phone up until the point of the command being sent; this is used as the input `.pcap` file for the *Python* program. Once the correct port number and IP address for the camera are set, the attack can be executed.

```
Sent data: b'#\x00\x06\x00\x00\'\x00\x00\x00\x00\x00\x03\x00\x01\x00\x15\x00\x00\x00\x18{"ledopenflag":"2"}'
Received data: b'#\x00\x06\x00\x00\x18\x00\x00\x00\x00\x00\x02\x00\x01\x00\x10\x00\x00\x02\x00\x00\x00\x00\x01#\x00\x06\x00\x00\x18\x00\x00\x00\
x00\x00\x04\x00\x01\x00\x14\x00\x00\x00\x18\x00\x00\x00\x00#\x00\x06\x00\x00\x18\x00\x00\x00\x00\x00\x04\x00\x01\x00\x15\x00\x00\x00\x18\x00\x00
\x00\x00'
Connection closed
```

FIGURE 9.8: Same network command execution attack results

Running the program with these updated parameters results in the LED off command being sent to the camera. This causes the LED to turn off on the camera, verifying that the command has been executed successfully. The camera also returns a packet to the *Python* program. However, unlike the server authentication attack, we cannot simply verify the success of the attack with this response as the returned data is not utf-8 encoded. Thus, the program is unable to decode this data into a readable format.

Despite the initial success of the attack, it was subsequently discovered that running the attack program later resulted in the program being unable to establish a communication to the camera. When the program attempted to set up the connection, the camera would actively refuse to establish a connection to the laptop. Creating a new set of captures of commands sent from the app revealed that the port being used by the camera to communicate had changed, hence the attempts to connect to the old port number were being refused. Switching the port number in the program restored the intended function of the attack again.

As the camera does not use the same port number permanently, this attack requires knowledge of a relatively recent capture to be executed successfully as the port information can be easily extracted from any capture over the correct communication structure. However, this requires a user to be accessing the device before the attack is able to be performed. One method to resolve this issue is to modify the script to sequentially attempt to establish a connection to port numbers, only sending the command packets once a connection has been established. However,

this would adversely impact the time required to perform an attack as potentially 1000s of port numbers may have to be tried before finding a suitable port to connect to. Another potential solution to speed up this process would be to integrate a *NMAP* scan into the program such that a list of open ports could be identified before the program attempts to establish a connection. The program would then attempt to connect to the ports returned as open by the scan rather than sequentially enumerating all possible port numbers.

Although the LED off command has been performed here, an attacker could change this to execute arbritrary commands supported by the camera by substituting `{"ledopenflag":2"}` with the appropriate JSON for the command they wish to execute. It is not necessary for the attacker to wait for a user to execute a command to capture the correct JSON for this as the camera automatically reveals a list of parameters at the top of the video feed header in figure 9.1.

Due to this, the same network command execution attack has higher severity than the previous attacks as complete compromise of the command and control system of the camera can be achieved through this attack. Additionally, as the attacking device sets up an independent connection with the camera as opposed to intercepting and altering an authentic users communication, the attack would function any time the devices are connected to the same network without a man-in-the-middle (MITM) structure being required. This attack would also be especially dangerous if the camera was connected to an open network as the attacker would not require any authentication to connect to the network being used by the camera. However, this attack is restricted to the same network communication structure so the attacker must be on the same LAN and hence geographic location to perform the attack.

9.4 Different networks command execution

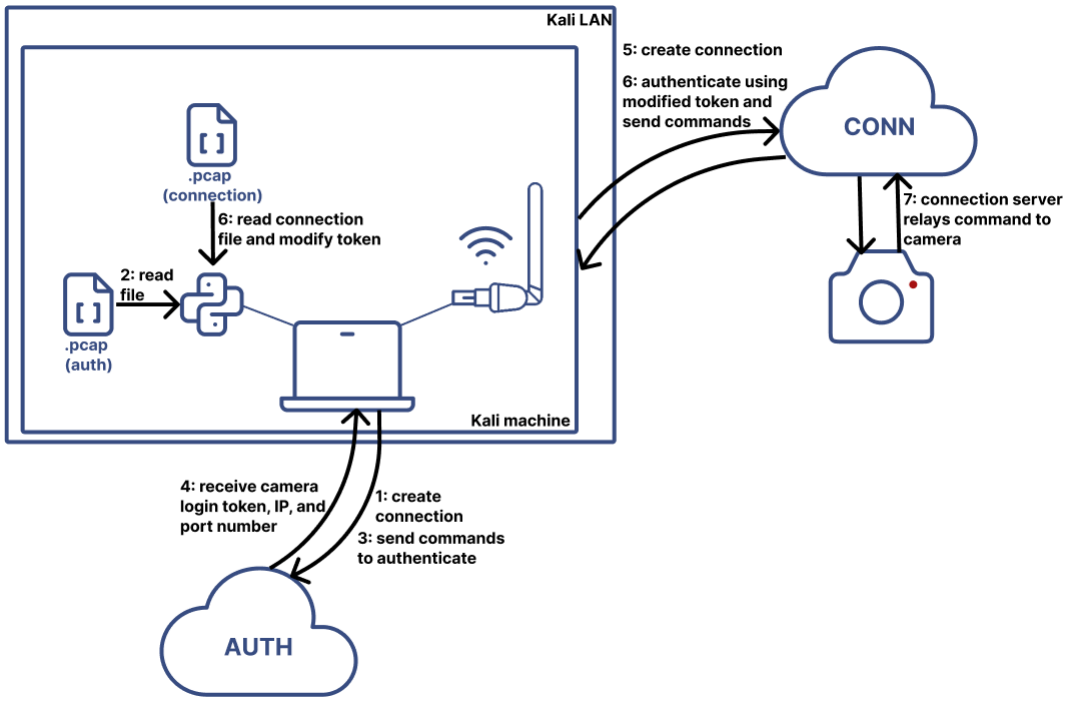


Figure 9.9: Command execution attack process with both devices on different networks

The versatility of the command execution attack can be enhanced by extending the attack to function when the devices are connected to different networks. For the attack to function with the devices on different networks, the extra communication steps need to be accounted for as in section 8.3. In particular, establishing a connection to and authenticating with the connection server must be performed before the attacking device is able to execute any commands. As such, the code used to perform the same network attack is unsuitable for performing this attack.

As this attack requires authenticating with two servers, two capture files are required for the attack to function. The first of these is a capture of the phone authenticating with the authentication server up until the point at which it sends a command to retrieve the IP, port number and security token required to establish a connection to the connection server, similar to the capture that was required to carry out the authentication attack in section 9.2. Additionally, a capture file containing the authentication to the connection server and communication up until the point of a command being sent is required. We are able to obtain both capture files during the same capturing session by recording the app being opened whilst the devices are on different networks, and disabling the indicator LED through the camera settings.

The captures required for the attack can then be processed by a *Python* program to perform the attack.

Initially, the program creates a connection to the authentication server and authenticates with the user token. Subsequently, the remaining sequence of commands is sent, prompting the server to return the IP, port number, and security token required for authentication with the connection server. As the user token is permanent, the capture file obtained need not necessarily be recent.

```
Sent data: b'\x1f\x99\x00\x81\x18A\x00\x00-{"r":"65559","to":{"did":"1200010140d971bd"}}C\x00\x00c{"type":"event","method":"queryCalendar","valu
e":{"src_id":"1200010140d971bd","from":"2024-04-17"}}'
{"r":"65547"}C~{"code":"1000","data":{"conn_token":"1:d87vxcugofydd1nzdj","media_ip":"104.233.210.75","media_port":"22045"},"desc":"Success"}

Token: 1:d87vxcugofydd1nzdj
Media IP: 104.233.210.75
Media Port: 22045
```

FIGURE 9.10: Extraction of the security token, IP, and port number from communication with the authentication server

Once the response is received from the authentication server, we extract and store only the required information into variables. Regular expressions are employed to perform this operation by pattern matching in the returned string for the JSON strings `conn_token`, `media_ip`, and `media_port`. Using an expression in the format `'"conn_token"\s*:\s*"([^"]*)"'` allows us to extract the corresponding values. Searching for matches is performed on all server responses, although all responses are stripped of non utf-8 characters before searching commences.

Prior to authenticating with the connection server, the attacking script must first modify the contents of the second capture file to replace the security key originally in the capture with the new key retrieved from communication with the authentication server. Following this operation being performed, a connection is established to the connection server at the IP and port retrieved, and authentication is performed by sending the altered packets from the second capture to the server. We must be diligent to avoid sending any extra SYN, ACK, or repeated data packets from the original capture as upon receiving packets of this nature, the server will not provide a response. In contrast to the authentication server which automatically filters out duplicate packets and provides error messages on receipt of invalid queries, the connection server halts communications by not responding. Hence, the data transmitted must be entirely accurate for the authentication to succeed. Subsequent to authentication, the remainder of the commands to turn off the LED are transmitted to the connection server which, in turn, relays these commands to the camera.

Figure 9.11: Different networks command execution attack results

Running the attack results in the LED successfully being turned off. Additionally, the connection server returns a response to the laptop. As this response, unlike in the same network attack, is utf-8 encoded we can view the plaintext to confirm the response matches that of when a valid operation is performed by the app. Similarly to the same network attack, the different networks attack can be extended to perform arbitrary operations by substituting in the correct JSON string.

Compared to the same network attack, the versatility of this attack is vastly improved as the attacker can be situated remotely. This drastically increases the threat posed by this attack through reduction of the barrier to entry. For example, provided that the required captures were previously obtained, an attacker could carry out commands on a victim camera without needing to leave their home. Remote operation also allows malicious persons to compromise cameras connected to secure networks as network access is not a prerequisite. Although the current format of the attack necessitates the capture being obtained from the device we wish to attack, it may be possible to extend the attack to function using only the user login token of a valid user account. As the information communicated to the connection server to perform the attack follows a set pattern consisting primarily of information returned from the authentication server, with further reverse engineering of the communication stream, it may be feasible to fully reconstruct the required command using only information returned by supplying the authentication server with the user token.

9.5 SUMMARY OF VULNERABILITIES

We can summarise the attacks enumerated, categorising their risk relative to each other by applying the DREAD threat modelling system to determine the overall danger posed by each attack. In this model, the formula $Risk = (Damage + Affected\ Users) * (Reproducibility + Exploitability + Discoverability)$ is applied to evaluate the risk posed by the attack in a structured manner [60]. Using such an approach, we are able to more effectively quantify risk although it important to acknowledge that the DREAD scores given are not a definitive quantification of risk comparable to DREAD analysed attacks on other systems [61]. For all categories, the attacks presented are scored from one to five with one given as the affected users score in all cases as all attacks focus on compromise of a single user account or camera.

Attack	DREAD risk score	Summary
Plaintext command capture	$(1+1)*(3+5+5) = 26$	Plaintext communications allows commands and general system communications to be easily intercepted.
Attacker app login	$(5+1)*(1+5+4) = 60$	Plaintext login and sign-up services allow user email and password to be divulged and used by an attacker. If the user uses the login information in other places, this could also compromise accounts for other products.
Remote server authentication	$(2+1)*(4+4+4) = 36$	Capturing of the user login token allows illegitimate authentication with the authentication server.
Same network command execution	$(4+1)*(2+3+3) = 40$	Lack of authentication required allows for arbitrary command execution using a python script whilst both devices are connected to the same network.
Different networks command execution	$(4+1)*(4+3+2) = 54$	Non changing user token allows access information of the connection server to be revealed, allowing for arbitrary command execution remotely.

TABLE 9.1: Summary of attacks tested

In all of the attacks presented, the overall risk of the attack could be reduced by encrypting communications. This would entirely mitigate the first two attacks as they rely on unencrypted communications and significantly reduce the impact of

the replay based attacks provided the encryption method was completely secure. Complete mitigation of the remaining attacks could be achieved by combining encryption with a timestamp or single use authentication token with each message to prevent replayed encrypted messages from being misinterpreted as authentic communications.

10 Secondary Camera Testing

Having gained an understanding of the system functionality of the camera, we can further perform analysis of a secondary camera system to discover if other IP cameras operate using a similar system. Understanding where similarities and dissimilarities exist between different camera systems allows for improved understanding as to whether weaknesses may be common across multiple systems, or if particular systems observe higher security standards. Due to secondary camera testing being a **could** objective and having far greater time constraints, the testing is performed as an overview of the system function and communication structure rather than as a detailed analysis to verify the maximal number of vulnerabilities.

The secondary camera being testing is the *Qualihome Mini Spy Camera*. This device was chosen as, similarly to the primary camera tested, the secondary camera is an IP spy camera that is readily available on Amazon in a similar price bracket. This makes the *Qualihome* camera suitable for comparison to the primary device as both are applicable to the same use case. Hence, both devices would likely be considered for use by the same set of consumers in the same use case scenarios. Moreover, the secondary camera also uses a mobile application for commands and video transmission. The software used is the *365Cam* app which is available on Android and iOS. Crucially, this software is different to that used by the primary camera and is created by a different manufacturer. Devices from differing manufacturers are required to make a useful comparison as two devices using the same software would likely use the same communication structure and hence no comparison would be able to be drawn between the two.

10.1 Camera overview

As with the primary camera, an overview of the system that the secondary camera uses is initially obtained. From this we are able to gain an understanding of the general functionality of the second camera and how this compares to that of the

primary camera. Constructing such an overview also gives a qualitative impression of the similarity between the two devices from the perspective of a general user of the devices. Comparably to the primary camera, no comprehensive documentation pertaining to the functions of the camera or app is readily available. Subsequently, information about the system and its function is collected experimentally by testing the app under normal usage conditions.

10.1.1 Camera hardware

In comparison to the primary camera, the secondary camera is substantially more physically compact. Both devices act as IP spy cameras, however the secondary camera acts only as a camera without being disguised as another device.

(a) Camera top (b) Camera bottom

Figure 10.1: Top and bottom of the secondary camera

The primary components of the secondary camera are listed in table 10.1:

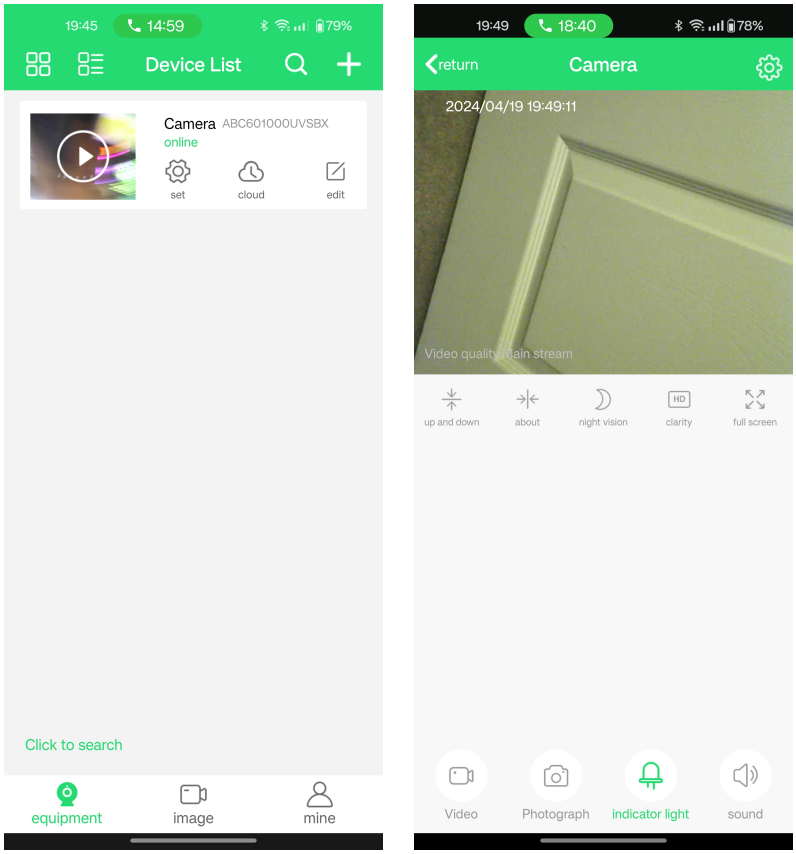
Item number	Description
1	Reset button
2	Indicator LED
3	Power button
4	Camera module
5	SD card slot
6	USB charging port

Table 10.1: Table of camera components

Consistent with the primary camera, this camera also contains an inbuilt battery along with an internal WiFi module and receiver. This provides the camera with functionality to both produce an AP, and also connect to external WiFi networks. APs created by the inbuilt WiFi module do not provide any external internet connectivity, but are used as the default connection method to the mobile app until a connection to an external network is configured.

10.1.2 APPLICATION

Interaction with the secondary camera is performed through the *365Cam* mobile application. The *365Cam* app is structured in a manner reminiscent of the *HDlivecam* app and contains a similar feature set.



(a) Home screen (b) Camera feed

FIGURE 10.2: Home screen and camera page for the *365Cam* app

The app is listed on the Google Play store under the company name *SHIX ZHAO* [62] although no link to a company website is provided, with very limited other information regarding the company available. This is consistent with the app and device listings of the primary camera as, in both cases, the app is published under a

different manufacturer name to that of the camera.

Upon initially launching the app, an advert is displayed and several permissions are requested such as location, microphone, and camera access of the mobile phone running the app. When prompting for approval to allow said permissions, the phone indicates that location data is shared with third parties. Only notification and microphone permissions were requested by the *HDlivecam* app.

Subsequent to opening the app, the user is immediately taken to the home screen without any form of account creation. At no point in the function of the app, does the user create an account to which cameras are associated. On the home screen, a list of connected devices is displayed, as well as several tabs linking to additional app functionality. At the top of the homepage, the user has options to change the view type and add new devices. The options presented when attempting to add a new device indicate that the app supports types of IoT devices other than just cameras, with the menu specifically providing a section for tools such as smart dental monitors. Devices can also be added using either the camera created AP or by searching for cameras currently connected to the same WiFi network as the phone. When the camera is initially being set up, the camera AP must be used as there is no way to connect the camera to an external network without first connecting via the inbuilt AP. Cameras access is also able to be shared through a QR code.

On each device card, options for additional controls are presented including the camera settings, an edit menu for device name and password configuration, and cloud options. In parallel with the *HDlivecam* app, cloud services require a subscription payment to be used. At the bottom of the homepage, tabs for *image* and *mine* are present. The first of these provides a local storage for any images or videos taken by the camera using the controls in the app. The latter contains a page of general app settings such as language configuration.

The camera feed page is formatted very similarly to that of the *HDlivecam* camera feed page shown in figure 5.2. The camera feed is displayed at the top of the page along with a link to additional camera settings. Below this, the camera options are presented. These options match those presented in the primary camera such as night vision, microphone capture, and selection between two camera quality settings. Unlike the primary camera, the IR functionality of the second camera appears to be only passive as no IR blasters are visible on the device.

10.2 Communication structure

Subsequent to obtaining the system overview, we are able to conclude that the user experience between both devices remains similar. Following on from this, we analyse the communication structure to determine whether the similarities extend to the technical implementation of the camera, and hence if common points of security weakness may exist.

Again, the process of deriving communication structure is performed through systematic packet capturing and analysis. Overall, three types of communication between the app and camera can be identified, matching those found to exist in the primary camera system. However, the operating principles of the secondary camera were found to differ substantially from those employed in the primary camera communication system. We can, however, generalise the communication structures available to the following:

- Direct communication over camera produced LAN.
- Same network communication over Kali produced LAN.
- Remote network communication.

Firstly, a *NMAP* scan is performed on the camera to gain information about the networking services operating on the device. The `-A` and `--osscan-guess` flags were applied to return additional information about the OS running on the camera. As the camera IP address is not shown in the app of this system, we first identify the correct address by connecting the camera to the Kali network and observing the IP shown in a test *Wireshark* capture.

```
┌──(james㉿kali)-[~]
└─$ sudo nmap -A --osscan-guess 192.168.12.246
Starting Nmap 7.94 ( https://nmap.org ) at 2024-04-19 21:57 BST
Nmap scan report for 192.168.12.246
Host is up (0.0096s latency).
Not shown: 999 closed tcp ports (reset)
PORT      STATE SERVICE     VERSION
10002/tcp open  documentum?
MAC Address: 00:02:05:0E:CE:3E (Hitachi Denshi)
Aggressive OS guesses: iRobot Roomba 980 vacuum cleaner (93%), IHome SmartPlug iSP5WWC (92%), Anova
), Milight WiFi Receiver bridge (89%), Denver Electronics AC-5000W MK2 camera (89%), High-Flying HF-
eight TCP/IP stack (88%), Rigol DSG3060 signal generator (88%)
No exact OS matches for host (If you know what OS is running on it, see https://nmap.org/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=7.94%E=4%D=4/19%OT=10002%CT=1%CU=31344%PV=Y%DS=1%DC=D%G=Y%M=00020
OS:5%TM=6622DB69%P=x86_64-pc-linux-gnu)SEQ(CI=I%TS=U)SEQ(SP=0%GCD=1%ISR=5E%
OS:TI=I%CI=I%II=RI%SS=S%TS=U)SEQ(SP=0%GCD=14F%ISR=5E%TI=I%CI=I%II=RI%SS=O%T
OS:S=U)SEQ(SP=0%GCD=152%ISR=5E%TI=I%CI=I%II=RI%SS=O%TS=U)SEQ(SP=18%GCD=1%IS
OS:R=5E%TI=I%CI=I%II=RI%SS=O%TS=U)OPS(O1=M578%O2=M578%O3=M578%O4=M578%O5=M5
OS:78%O6=M578)WIN(W1=1C84%W2=1C84%W3=1C84%W4=1C84%W5=1C84%W6=1C84)ECN(R=Y%D
OS:F=N%T=FF%W=1C84%O=M578%CC=N%Q=)T1(R=Y%DF=N%T=FF%S=O%A=S+%F=AS%RD=0%Q=)T2
OS:(R=N)T3(R=Y%DF=N%T=FF%W=1C84%S=O%A=S+%F=AS%O=M578%RD=0%Q=)T4(R=Y%DF=N%T=
OS:FF%W=1C84%S=A%A=S%F=AR%O=%RD=0%Q=)T5(R=Y%DF=N%T=FF%W=1C84%S=A%A=S+%F=AR%
OS:O=%RD=0%Q=)T6(R=Y%DF=N%T=FF%W=1C84%S=A%A=S%F=AR%O=%RD=0%Q=)T7(R=Y%DF=N%T
OS:=FF%W=1C84%S=A%A=S+%F=AR%O=%RD=0%Q=)U1(R=Y%DF=N%T=FF%IPL=38%UN=0%RIPL=G%
OS:RID=G%RIPCK=G%RUCK=G%RUD=G)IE(R=Y%DFI=S%T=FF%CD=S)

Network Distance: 1 hop

TRACEROUTE
HOP RTT     ADDRESS
1   9.57 ms 192.168.12.246

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/
Nmap done: 1 IP address (1 host up) scanned in 199.06 seconds
```

FIGURE 10.3: Detailed *NMAP* scan on the IP of the second camera

Resulting from the scan, it is shown that the device has a single open port at 10002 running the service `documentum?`. Further expanding the search to include more port numbers was found to not reveal any more open ports. This contrasts the primary camera which had multiple ports found to be open, although in both devices the open ports are shown to be running the TCP protocol. Furthermore, the scan reveals the MAC, showing the network hardware to be provided by `Hitachi Denshi`.

Analogously to the results obtained from the primary camera, the scan for the device OS is unable to uncover an exact match. Despite this, several high percentage matches are returned, corresponding to several other types of IoT device. In addition to this, the TCP/IP fingerprint again references `x86_64 Linux`.

Building on the foundational information gained from the results of the *NMAP* scan, packet capturing is performed to uncover further detail as to how communications are performed between the two devices. As no communication capturing was performed over the direct communication structure with a camera produced LAN, all captures are able to be performed using *Wireshark* with the ALFA adapter in promiscuous mode as opposed to having to apply the alternative capturing method from section 8.1. To create the initial capture for analysis, we open the app and

connect to the camera. The indicator LED command is also issued before stopping the capturing process.

2332	19.267192950	120.76.133.14	192.168.12.253	UDP	1074	10773 → 15978 Len=1032
2333	19.297959680	120.76.133.14	192.168.12.253	UDP	1074	10773 → 15978 Len=1032
2334	19.311933059	192.168.12.253	120.76.133.14	UDP	52	15978 → 10773 Len=10
2335	19.312173010	192.168.12.253	120.76.133.14	UDP	52	15978 → 10773 Len=10
2336	19.312192358	120.76.133.14	192.168.12.253	UDP	87	10773 → 15978 Len=45
2337	19.312309097	120.76.133.14	192.168.12.253	UDP	62	10773 → 15978 Len=20
2338	19.315049845	192.168.12.253	120.76.133.14	UDP	52	15978 → 10773 Len=10
2339	19.316974095	120.76.133.14	192.168.12.253	UDP	62	10773 → 15978 Len=20
2340	19.317904400	120.76.133.14	192.168.12.253	UDP	1074	10773 → 15978 Len=1032
2341	19.317980356	120.76.133.14	192.168.12.253	UDP	1074	10773 → 15978 Len=1032

FIGURE 10.4: UDP communications between the camera and the phone

Reviewing the capture reveals that communications between the camera and the phone occur primarily over UDP rather than TCP. Unlike the TCP system used by the primary camera system, UDP is a connectionless protocol which does not guarantee delivery or ordering of packets. The UDP communication stream discovered appears to cover both video feed and commands sent between the two devices as it contains consistently sent packets with large amounts of data from the camera to the phone, with occasional data transmissions in the opposite direction. However, it is not possible to confirm exactly what is sent through viewing the ASCII representation of the stream as the received data is not in plaintext. Lack of readability indicates that either the data transmission is encrypted, or the format of commands is not in a readable format initially. In the same network communication structure, we are able to verify that the camera and phone communicate directly, similarly to in the primary camera system. This is validated by observing the two IP addresses associated with the UDP stream are `192.168.12.246` and `192.168.12.253` respectively. When performing the capture with the camera connected to a different network, an external server is used to connect to the camera as shown by the addresses in figure 10.4. As this does not match the public IP of the network the camera was connected to during testing, the address confirms the presence of an external server being used to facilitate access to the camera.

FIGURE 10.5: Additional connections to external services

Much akin to the primary camera system, the app established multiple additional connections to external services upon startup such as in figure 10.5. The majority of these are to certificate authorities and service providers. Additionally, when the app is started one specific TCP connection is established to an external server using the POST method regarding communications directly related to the camera communication system. Unlike the other camera communications over UDP, the data transferred to this server is done so in plaintext. Hence we can determine that the data sent is a combination of message content information and JSON commands.

(a) POST message (b) cont.

FIGURE 10.6: Plaintext POST communications with remote server

Examining the communication reveals that the app is authenticating with a remote connection at the address `147.139.39.175:9093`. In this authentication, the weak username and password pair `"userName"="admin","passwd"="admin123"` is being used to login with the server. Such a combination is highly vulnerable to brute force attacks as the password is both common and short. Once the login is processed, the server returns a result that is then used as the authorisation for the subsequent POST request. This request POSTs information about the app and camera including a token.

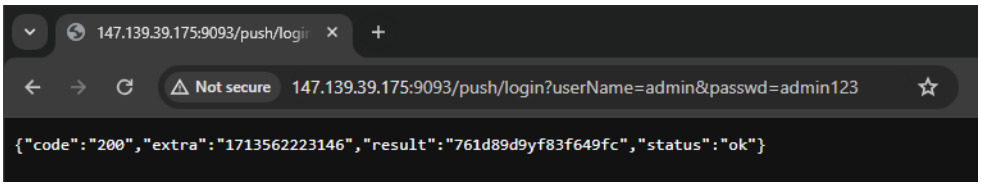


FIGURE 10.7: Attacker authentication with the remote server

As the plaintext communications divulged the weak username and password, we are able to attempt to connect to the remote server using a web browser as communication is occurring over a POST request. Simply connecting to the required address

and port combination at the path specified in the intercepted communications displays only an error message regarding invalid username or password; no web interface containing a POST form is present at the address. However, despite the app communications being performed entirely using POST, constructing a GET request in the URL in the format `?userName=admin&passwd=admin123` acts as an equivalent command. This is accepted by the server which successfully returns the authorisation key as shown in figure 10.7.

10.3 Summary

Overall, the methods employed by the secondary camera bear both similarities and differences to that of the primary camera, although the overall function and structure of both systems remains highly similar. This indicates that the security findings discovered in regards to the primary camera may be relevant in aiding to detect vulnerabilities and mitigate issues in the second device. Despite the small dataset, this is also a positive indication that other comparable IP cameras may benefit from the findings.

Both devices operate three types of communication structure, despite using different protocols for transmission of data. With both, many other data streams are established by the app for external services and contacting certificate authorities; authentication servers and JSON commands are also both employed by both systems. Moreover, plaintext commands and weak security measures regarding the authentication servers prove to be a common weakness with both systems, one which could be mitigated with strong encryption and protection against replay attacks. Additionally, the user experience and app format between the devices remains very similar. Especially in the case of the visual layout of the app, very little difference is present. Furthermore, the feature set of both devices is nearly identical, including software provided features such as vertical and horizontal flipping of the camera feed from within the app.

11 Conclusion

Overall, the project has executed successfully having performed a comprehensive analysis of the communication structure and security of the *UYIKOO Spy Clock Camera 140°HD 1080P* and its associated systems, matching the overarching goal of the project as specified in section 1.2. Throughout the course of the investigation, effective project management has proved pivotal in driving success in spite of uncertainties and roadblocks encountered. The multitude of vulnerabilities discovered provides proof of the lack of sufficient security measures utilised by IoT manufacturers and attempts to bring more attention to the issue. Additionally, this underscores the need for additional research in the field to expose further vulnerabilities in devices and formulate mitigation strategies, improving the security of the sector as a whole.

11.1 Project outcomes

Comparing to the original MoSCoW prioritised objectives, it is evident that the project has succeeded and, indeed, exceeded the key requirements outlined. Of the outlined objectives, all those in the **Must** and **Should** categories have been completed successfully with all **Won't** categorisations having been avoided, meeting the legal and ethical considerations made. Of the **Could** categorisation, only the following objectives were not met; these objectives were only extensions to the primary goals and hence do not subtract from the success of the project:

- Outline a testing methodology that can be used to perform security analysis in a systematic and repeatable manner on other IoT devices.
- Test vulnerabilities from a previous report on an IP camera and report whether the camera being tested in this project is vulnerable to these previously known attacks.

Through the **Must** objectives, the app has been reverse engineered and the communication structures of the system have been successfully derived. By doing so,

a foundation for vulnerability enumeration was formed, additionally providing foundational research which further studies can draw information from. Building upon this foundation, vulnerabilities were enumerated and verified through the **Should** objectives.

Over the course of the investigation, a range of vulnerabilities were discovered with implications ranging from minor flaws in system software to severe compromises of user information. Attacks exploiting these vulnerabilities were further demonstrated such as the command execution scripts, evidencing the fact that the discovered vulnerabilities not only posed a threat to security, but could be actively exploited. In several cases, the barrier to entry of exploiting vulnerabilities was found to be low, being able to be compromised through simple attack vectors. Furthermore, attacks were contextualised by providing scenarios in which a malicious actor would be able to apply each form of attack. As such, the lack of adequate security measures employed by the camera and its related systems were revealed, reinforcing the need for security research in IoT devices to continue to be conducted.

Resulting from the findings of the attacks, general mitigation strategies were able to be proposed, despite not being part of the formal requirement of the project. However, although both compromises and mitigations have been suggested, resolutions cannot be implemented by any other than the manufacturer. Presenting the findings in a clear manner provides information with which manufacturers can implement fixes. Moreover, the secondary camera testing validates the transfer-ability of the findings, illustrating how the results presented have implications beyond the security of the specific camera tested, extending to IP camera security as a whole.

These outcomes highlight how the dangers posed by insecure devices prove to be tangible risks to user security and the emphasis that needs to be imposed on the need for heightened security measures in IoT devices.

11.2 Limitations

Reflecting on the technical results highlights how the inherent uncertainties related to security analysis resulted in several challenges being encountered such as direct replay attacks not proving to be viable. Despite this, due to the efficacy of the agile principles employed, resolutions to such issues were easily able to be devised and incorporated into the project. The MoSCoW prioritisation method further aided

in task scheduling in light of the challenges presented.

Although the project proved successful, the investigation did not occur without limitations. Primarily, these existed in the form of time constraints resulting from specific challenges causing delays. Specifically, the issues related to using an emulated device as was proposed in the initial technical methodology caused the greatest loss of time and resulted in methodology changes needing to be implemented before the investigation could further proceed. Additionally, the lack of evidence supporting any specific encoding format being used for the video feed led to the feed not being decoded as it was considered not the most beneficial use of available time.

More research earlier on at the initiation stage of the project would have aided in resolving some of the timing issues encountered. By gaining additional information and potentially practical experience as to how to perform security testing on internet connected devices and android apps, the impact of some challenges faced could have been minimised, furthermore increasing the velocity at which discoveries could be made.

11.3 Further work

Aside from the **could** objectives highlighted earlier, the discoveries resulting from this investigation pose additional areas for further research. Even within the scope of the primary camera tested, the lack of security presently demonstrated suggests a likelihood that other vulnerabilities exist within the system. As such several targets for further work are outlined.

- **Decoding video feed** - Attacks covered in this investigation provide complete compromise of the command and control system of the camera. Recovery of the video feed either by decoding packet captures or directly intercepting the stream would provide full compromise of the camera system.
- **Extension of command execution attacks** - Based on the attacks presented, resolving the port numbering issue would improve the success rate of the same network attack. Likewise, reconstructing the required packets for communication with the connection server could allow for attacks to be launched with as little information as only the user token. This would substantially increase the threat posed by the different networks attack.

- **Exploration of app level vulnerabilities** - Issues within the app itself were not explored in this investigation but could lead to information disclosure resulting directly from the app installation.
- **Further testing of secondary camera** - Building upon the information determined regarding the secondary camera system, further investigation into the device could expose additional vulnerabilities and reveal more shared weaknesses with the primary camera.

11.4 Author's Assessment of the Project

This project makes significant technical contribution through conducting a systematic security analysis of the UYIKOO Spy Clock Camera and its accompanying software. Performing this included thorough examination of the communication structure of the system, a process which consisted of both reverse engineering of the *Android* app, and analysis of communications. For analysis of communications to be performed, packet capturing techniques were required, with manual examination of multiple captures used to construct a view of the communication structures employed. Additionally, this project enumerates, verifies, and documents discovered vulnerabilities within the camera system; the implications of which are proved to extend beyond the tested device through manner of the secondary camera comparison made.

The success of this investigation aids in addressing the issues regarding insufficient security in the field of IoT devices. As the field continues to expand, the proliferation of insecure endpoints has increasing consequences in the field of computer science as a whole. By conducting a security analysis, this project demonstrates the practical implications of substandard security in networked ecosystems. Performing such research is paramount to providing a platform from which future security advances can be made.

Leveraging the results discovered during the course of this investigation, a basis is formed from which further research into vulnerabilities of similar systems can be conducted. Through replication of the attacks presented, numerous other systems may be discovered to have similar security flaws. Additionally, through extension of the vulnerabilities discovered, other more sophisticated attacks may be devised. Resulting from this, manufacturers and researchers can work to enhance the security of future products and infrastructure. Furthermore, the detailed communication

structure presented allows for future system comparisons to other devices.

By successfully meeting all primary objectives, the project illustrates the capability to address and overcome technical challenges within the realm of cyber security, and deliver new research to the sector from which further progress can be made. Over the course of the project, new areas of expertise had to be developed, particularly in the realm of packet sniffing and analysis, required to derive the communication structures and discover potential points of weakness. For this, it was necessary to learn to use new tools such as *Wireshark*.

Despite the accomplishments of the project, the scope and depth of research proved limited by time constraints. As such, the search for vulnerabilities within the device was far from exhaustive, excluding the app and hardware from the scope of vulnerability enumeration. Due to roadblocks faced at different stages of the investigation, delays led to certain stages of the project taking longer than initially anticipated although this was, in part, mitigated through agile re-planning and restructuring of the investigation.

Bibliography

1. Foote, K. D. *A Brief History of the Internet of Things* 2022. https://www.dataversity.net/brief-history-internet-things/.
2. Google. *IoT* https://trends.google.com/trends/explore?date=all&q=IoT&hl=en-GB.
3. Statista. *IoT average sensor costs 2004-2020* Accessed: 2024-4-24.
4. Muncaster, P. *The spy who rented to me? Throwing the spotlight on hidden cameras in Airbnbs* Nov. 2022. https://www.welivesecurity.com/2022/11/01/spy-who-rented-to-me-hidden-cameras-airbnbs/.
5. Shodan. *Shodan* shodan.io.
6. UYIKOO. *Wifi Hidden Camera, UYIKOO Spy Clock Camera 140°HD 1080P Spy Hidden Camera Wifi Clock Camera Support App Remote View* https://www.amazon.co.uk/Hidden-Camera-UYIKOO-140%C2%B0HD-Support/dp/B08NJQXBL1/ref=sr_1_3?keywords=hd+wifi+clock+camera&qid=1688653537&sprefix=hd+wifi+clock+camera%2Caps%2C71&sr=8-3.
7. Brush, K. *MoSCoW method* Mar. 2023. https://www.techtarget.com/searchsoftwarequality/definition/MoSCoW-method.
8. Seralathan, Y. *et al. IoT security vulnerability: A case study of a Web camera* in (2018), 172–177. doi:10.23919/ICACT.2018.8323686.
9. Howarth, J. *80+ Amazing IoT Statistics (2023-2030)* https://explodingtopics.com/blog/iot-stats.
10. Wurm, J., Hoang, K., Arias, O., Sadeghi, A.-R. & Jin, Y. *Security analysis on consumer and industrial IoT devices* in. hardware flaws in IoT devices (2016), 519–524. doi:10.1109/ASPDAC.2016.7428064.
11. Biondi, P., Bognanni, S. & Bella, G. *Vulnerability Assessment and Penetration Testing on IP camera* in. testing of a camera example (IEEE, Dec. 2021), 1–8. doi:10.1109/IOTSMS53705.2021.9704890.

12. Bugeja, J., Jonsson, D. & Jacobsson, A. *An Investigation of Vulnerabilities in Smart Connected Cameras* in. using shodan and cve to find globally insecure cameras (IEEE, Mar. 2018), 537–542. doi:10.1109/PERCOMW.2018.8480184.

13. Culture, M. S. D. F. D. *Code of Practice for Consumer IoT Security* 2018. https://assets.publishing.service.gov.uk/media/60576f54e90e0724c0df4631/Code_of_Practice_for_Consumer_IoT_Security_October_2018_V2.pdf.

14. El-Gendy, S. & Azer, M. A. *Security Framework for Internet of Things (IoT)* in. iot security framework suggestion (IEEE, Dec. 2020), 1–6. doi:10.1109/ICCES51560.2020.9334589.

15. Kurose, J. F. & Ross, K. W. *Computer networking : a top-down approach* 7th ed., 852. ISBN: 9780133594140 (Pearson).

16. Schulzrinne, H., Rao, A., Lanphier, R., Westerlund, M. & Stiemerling, M. *Real-Time Streaming Protocol Version 2.0* Dec. 2016. doi:10.17487/RFC7826.

17. U.Farooq, M., Waseem, M., Mazhar, S., Khairi, A. & Kamal, T. A Review on Internet of Things (IoT). *International Journal of Computer Applications* **113,** 1–7. doi:10.5120/19787-1571 (1 Mar. 2015).

18. Sharma, N., Shamkuwar, M. & Singh, I. in (eds Kumar, V., Raghvendra, K., E., K. M. B. V. & Solanki) 27–51 (Springer International Publishing, 2019). doi:10.1007/978-3-030-04203-5_3.

19. Nilsson, R., Skolan, K., Elektroteknik, F. & Datavetenskap, O. *Penetration testing of Android applications*

20. Shah, K. *Penetration Testing Android Applications* http://www.mgovworld.org/topstory/mobile-applications-market-to-reach-9-billion-by-2011.

21. Alanda, A., Satria, D., Mooduto, H. & Kurniawan, B. Mobile Application Security Penetration Testing Based on OWASP. *IOP Conference Series: Materials Science and Engineering* **846,** 012036. doi:10.1088/1757-899X/846/1/012036 (1 May 2020).

22. Oracle. *VirtualBox* https://www.virtualbox.org/.

23. Google. *Android Studio* https://developer.android.com/studio.

24. Buchanan, E. *GPU Passthrough for Virtual Machines: KVM vs VirtualBox* 2023. https://devicetests.com/gpu-passthrough-kvm-vs-virtualbox.

25. Qualihome. *Qualihome Mini Spy Cameras Hidden 1080P HD Wireless Camera with Night Vision Motion Detection, WiFi Camera Home Security Nanny Surveillance Cam Perfect Video Bady Camera for Indoor and Outdoor* https://www.amazon.co.uk/dp/B0CP733873?psc=1&ref=ppx_yo2ov_dt_b_product_details.

26. Inc, A. N. *AWUS036NHA* https://www.alfa.com.tw/products/awus036nha?variant=36473966166088.

27. Sabih, Z. *Learn Ethical Hacking from Scratch* 1st ed., 4–4 (Packt, July 2018).

28. Home, C. *HDlivecam* https://play.google.com/store/apps/details?id=com.hytech.yuncam.viewer.googleplay&hl=en&gl=US.

29. Limited, O. S. *Kali Linux* 2024. https://www.kali.org/.

30. Foundation, W. *Wireshark* https://www.wireshark.org/.

31. Foundation, P. S. *Python* 2024. https://www.python.org/.

32. Microsoft. *Visual Studio Code* 2024. https://code.visualstudio.com/.

33. Inc, G. *GitHub* 2024. https://github.com/.

34. Overleaf. *Overleaf* 2024. https://www.overleaf.com/.

35. skylot. *jadx* https://github.com/skylot/jadx.

36. oblique. *create_ap* https://github.com/oblique/create_ap.

37. *Computer Misuse Act 1990* Feb. 2023. https://www.legislation.gov.uk/ukpga/1990/18.

38. Sliger, M. *Agile project management with Scrum* in (Project Management Institute, 2011).

39. Aroral, H. K. Waterfall process operations in the fast-paced world: project management exploratory analysis. *International Journal of Applied Business and Management Studies* **6,** 91–99 (1 2021).

40. Home, C. *Care Home* https://play.google.com/store/apps/dev?id=9101274103348469146&hl=en&gl=US.

41. smartcloudconn. *smartcloudconn* https://smartcloudcon.com/.

42. Ezzat, A. *Hacking & Fuzzing Home Surveillance Camera* Jan. 2021. https://bitthebyte.medium.com/hacking-fuzzing-home-surveillance-camera-edf2fe0b4e5.

43. Malinen, J. *Hostapd* Jan. 2013. http://w1.fi/hostapd/.

44. Licel. *Decompilation and modification* 2024. https://licelus.com/resources/guide-to-mobile-application-protection/threats/decompilation-and-modification.

45. APKCombo. *APKCombo* https://apkcombo.com.

46. Mauthe, N., Kargen, U. & Shahmehri, N. *A Large-Scale Empirical Study of Android App Decompilation* in (IEEE, Mar. 2021), 400–410. doi:10.1109/SANER50967.2021.00044.

47. Dexon. *What are RGB and YUV color spaces?* Apr. 2022. https://dexonsystems.com/blog/rgb-yuv-color-spaces.

48. Team, Q.-E. *Quark Engine book* https://quark-engine.readthedocs.io/en/latest/#.

49. NSA. *Ghidra* https://ghidra-sre.org/.

50. Nmap. *nmap* https://nmap.org/.

51. Davidson, J. *Voice Over IP Fundamentals* ISBN: 9781587052576. https://books.google.co.uk/books?id=MLxfy6W8bxgC (Cisco Press, 2006).

52. Sichuan. *AI-Link* https://www.ailinkiot.com.cn/index.html?lang=en_US.

53. VideoLAN. *VLC media player* https://www.videolan.org/.

54. Gillis, A. S. *DHCP (Dynamic Host Configuration Protocol)* Jan. 2023. https://www.techtarget.com/searchnetworking/definition/DHCP.

55. Foundation, W. *Wireshark Frequently Asked Questions* https://www.wireshark.org/faq.html#promiscsniff.

56. Prasad, A., Verma, S. S., Dahiya, P. & Kumar, A. A Case Study on the Monitor Mode Passive Capturing of WLAN Packets in an On-the-Move Setup. *IEEE Access* **9,** 152408–152420. doi:10.1109/ACCESS.2021.3127079 (2021).

57. Aircrack-ng. *Airodump-ng* https://www.aircrack-ng.org/doku.php?id=airodump-ng.

58. team, A. *Computer terms unwrapped: What is BSSID?* 2022. https://www.atera.com/blog/computer-terms-unwrapped-what-is-bssid/.

59. Klassen, F. & AppNeta. *Tcpreplay - Pcap editing and replaying utilities* https://tcpreplay.appneta.com/.

60. Howard, M. & Lipner, S. *The security development lifecycle : SDL, a process for developing demonstrably more secure software* ISBN: 9780735622142 (Microsoft Press, 2006).

61. Shostack, A. *Experiences Threat Modeling at Microsoft*

62. AppBrain. *SHIX ZHAO* `https://www.appbrain.com/dev/SHIX+ZHAO/`.