DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2021

Are modern smart cameras vulnerable to yesterday's vulnerabilities?

A security evaluation of a smart home camera

JESPER A LARSSON

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Are modern smart cameras vulnerable to yesterday's vulnerabilities?

A security evaluation of a smart home camera

JESPER A. Larsson

Degree Programme in Computer Science and Engineering
Date: June 10, 2021

Supervisor: Pontus Johnson
Examiner: Mathias Ekstedt
School of Electrical Engineering and Computer Science
Swedish title: Undviker dagens smarta kameror gårdagens sårbaraheter?
Swedish subtitle: Utvärdering av säkerheten hos en smart hem kamera

Abstract

IoT cameras can allow users to monitor their space remotely, but consumers are worried about the security implications. Their worries are neither unfounded as vulnerabilities repeatedly have been found in internet connected cameras. Have modern cameras learned from the mistakes of their predecessors?

This thesis has performed a case study of a consumer smart camera popular on the Swedish market. The camera was evaluated through a pentest. The evaluation found that the camera's cloud centric design allowed it to side step issues present in earlier models. However, it was demonstrated that it is possible to detect potentially sensitive events, e.g. when the camera notice motion, by just inspecting the amount of traffic it sends. Other tests were not able to demonstrate vulnerabilities though. Based on these findings it was concluded that the camera were more secure than it's predecessors, which supports that the market has improved.

Keywords: Security, Penetration testing, Threat modelling, Internet of things, Smart cameras

Sammanfattning

Konsumenter kan med IoT kameror på distans överse sin egendom. De är dock oroliga över hur säkra kamerorna är. Denna oro existerar inte utan god anledning. Sårbarheter har upprepade gånger påvisat finnas i internetuppkopplade kameror. Har dagens kameror lärt sig av deras föregångares misstag?

Detta examensarbete har testat en smart kamera som är populär på den svenska marknaden. För att fastställa hur säker kameran är genomfördes ett penetrationstest. Undersökning fann att kameran lyckats kringgå tidigare vanliga sårbarheter genom att förlita sig på molnet. Undersökning kunde dock konstatera att en motståndare kan läcka potentiellt känslig information, t.ex. när kameran upptäcker rörelse, bara genom att mäta hur mycket nätverkstrafik kameran sänder. Undersökningen kunde dock inte påvisa andra sårbarheter. Baserat på dessa resultat fann studien att denna kamera är säkrare än sina föregångare, och att detta stödjer tesen att marknaden som helhet förbättrats.

Nyckelord: Säkerhet, penetrationstestning, hotmodellering, Sakernas internet, Smarta kameror

Acknowledgments

I would like sincerely thank and dedicate this thesis to my ever supporting parents. You have believed in me when I did not do so myself. You have supported and pushed me when I needed so. Thank you.

I would also like to thank my supervisor and examiner from KTH that kindly have worked with me to realize this thesis. Your flexibility and understanding have been greatly appreciated this unusual past year. Thank you.

Stockholm, June 2021
Jesper A. Larsson

Contents

List of Figures

List of Tables

Listings

Glossary

Amazon Alexa Amazon's AlexaAmazon Alexa is a home automation ecosystem by Amazon.

Apple Homekit Apple's HomeKit is a home automation ecosystem by Apple.

application stack A group of software that work togheter toward a common goal

domain name A domain name is a string that serves to identifiy some resource or realm of authority on the internet. `example.com` is an example.

DREAD Risk assessment model. The word is mnemonic for *damage*, *reproducibility*, *exploitability*, *affected users*, and *discoverability*.

Google Nest Google's Nest is a home automation ecosystem by Google.

pentest Short for penetration test

pentester A person that performs a pentest

port scan A port scan is a technique where automated tools are used to probe network ports on one or multiple host to determine if any of the ports on the hosts are open and running services.

Acronyms

API Application Programming Interface

CVD Coordinated Vulnerability Disclosure

CVE Common Vulnerability or Exposure

CVSS Common Vulnerability Scoring System

DNS Domain Name System

DoS Denial of Service

DPI Deep packet inspection

EoP Elevation of Privilege

FTP File Transfer Protocol

HTTP Hypertext Transfer Protocol

HTTPS HTTP over TLS

ICMP Internet Control Message Protocol

ID Information Disclosure

IoT Internet of Things

IP Internet Protocol

ISP Internet Service Provider

MAC Media Access Control

mDNS Multicast DNS

MitM Man in the Middle

NAS Network-attached storage

NTP Network Time Protocol

OS Operating System

OWASP Open Web Application Security Project

RF Radio Frequency

RTP Real-Time Transport Protocol

RTSP Real Time Streaming Protocol

SCTP Stream Control Transmission Protocol

SFTP Secure FTP

SNI Server Name Indication

SRT Secure Reliable Transport

SSH Secure Shell

STRIDE Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege

TCP Transmission Control Protocol

TLS Transport Layer Security

TTL Time To Live

UDP User Datagram Protocol

WebRTC Web Real-Time Communications

XMPP Extensible Messaging Presence Protocol

XSS Cross Site Scripting

Chapter 1

Introduction

Internet connected devices and smart homes are becoming more and more prevalent.[12] While users of home automation products mention several benefits, they also raise concern, especially regarding remote access to locks and cameras. [13] Smart home cameras does provide consumers with useful features such as allowing them to remotely monitor their spaces, and even notifying them when there is moment in the monitored area. Some even allow the camera owner to talk back through the camera. Unfortunately, the security concerns mentioned by users are not unfounded. As smart home products introduce new features the threat surface increases compared to traditional versions of the products. Kalbo et al. surveyed video surveillance systems, the attacks against them, and the potential consequences of such attacks. They found that an adversary could be able to illicitly watch or manipulate the cameras' feeds. [14] The risks are neither merely theoretical. In the last decade we can readily find examples of severe vulnerabilities being present in consumer cameras. [15, 16, 17, 18, 19, 20, 21]

Should consumers today that would like to enjoy the benefits of monitoring their spaces with home camera be concerned that they would be inviting hackers to view their private spaces? The time requirement to look at the entire home camera market would be too great in project. Instead, this thesis will contribute in addressing this concern, through a pentest of a smart home camera available to today's consumers as a case study. The goal is that this study may contribute to an understanding of the security landscape of smart home cameras today, and together with other studies may provide insight in to how that landscape has shifted over time. Concretely it will contribute to answering the following question:

In what capacity are today's smart home cameras more secure

than those in the previous decade?

1.1 Delimitations

As mentioned in the previous section, completely answering the stated question would require time and resources beyond this project. To make the thesis feasible, while still providing a valuable contribution towards its answer a deliberate delimitations are required. As noted, time is factor in this choice. Delimitations were also made based on the author's field of study and with regard to legal concerns.

A delimitation (somewhat) implicit from the question itself is that the subject of study is security. Privacy is a closely related concern. If an adversary exploit a vulnerability that allow unauthorized access to a camera's video stream, then both the products security and the owners privacy are breached. But one does not necessarily imply the other. E.g. a manufacturer might collect data sensitive from a privacy perspective, without breaking the products security. Of course, one might argue that the manufacturer can collect that data because the product is designed to be selectively insecure. For the purposes of this thesis however, it is assumed that the adversary is not the cameras owner nor manufacturer, but a third party.

Another major delimitation is that this thesis will only cover a single camera. This limitation is unfortunate since the more cameras that are evaluated, the better the central question could be answered.

Further, this work will focus on the network, firmware and the user exposed hardware aspects of the evaluated cameras. Internal hardware will be neglected. This limitation is based on two orthogonal reasons. First, exploits that rely on opening and tampering with the cameras circuits does attract an additional level of attention beyond operating the power cord. Thus such are attacks less likely to succeed. Secondly, this thesis is the capstone of master specialised in software. The stated goal is (with author's emphasis):

> The aim of the degree project is for the student to apply and deepen knowledge, understanding, abilities and approach *within the context of the education.* [22]

Finally, there are some juridical concerns. These are discussed at length in subsection 9.3.2. In summary, Swedish law impose some limitation on what can be done as part of pentest. One such limitation is that one may only attack products or services with permission. An implication for this thesis is that the associated cloud services is out of bounds. Another implication of concern is

that reverse engineering (with a few exceptions) could be considered a breach of immaterial right.

1.2 Report outline

This thesis is divided into ten chapters as well as a section for references and four appendices. This chapter, chapter 1 Introduction, introduces the topic and the central question. The next chapter, chapter 2 Background, present the literature on which the methodology is based on. The methodology itself is then presented in chapter 3 Methodology. The methods implementation and result is then presented in the chapters chapter 4 Selected systems, chapter 5 Related vulnerabilities, chapter 6 Information gathering, 7, and chapter 8 Penetration test.

In chapter 4 Selected systems the selected camera is presented. chapter 5 Related vulnerabilities then presents earlier work that have found relevant vulnerabilities. In chapter 6 Information gathering we get familiar with the selected camera. That understanding is then advanced by constructing a threat model in 7. The identified threats are then examined in chapter 8 Penetration test.

Thereafter, the results are then discussed in chapter 9 Discussion. A conclusion is finally drawn in chapter 10 Conclusion & Future Work.

It worth noting that this thesis somewhat deviates from the standard thesis structure. The reader may be familiar with the following outline:

1. Introduction
2. Background
3. Method
4. Results
5. Discussion
6. Conclusion

Them most obvious deviation may be that the there is no chapter explicitly dedicated to results. This content is instead spread multiple chapters that describe the result of each large step. A perhaps more confusing deviation is that each threat tested has an associated background, method, result and focused discussion. These are separated from their traditional positions as

to deliver information and have focused discussions in a relevant context. A holistic discussion is then held in chapter 9 Discussion that builds upon on the focused counterparts.

Chapter 2

Background

This section will be dedicated to presenting theory and other background that is relevant for this thesis. To start off, Internet of Things will be presented and placed in context. Pentesting methodologies and related theory will then be presented. The presentation will in general aim to illuminate different views described in the literature where relevant, and compare them. This section is however only intended to summarize the available approaches. An argumentation for what methodology would be most appropriate to answer the question of this thesis is rather conducted in the Methodology section.

2.1 Internet of Things

Internet of Things (IoT) is an umbrella term for networked objects or systems with sensing and/or actuation capabilities, as well as a vision that these (often on their own limited) things may collaborate to provide emergent experiences or value. [23] Light bulbs, heaters, air conditions and motorized blinds are concrete examples of domestic things that can be actuated. Sensing devices such as thermostats, clocks, switches, motion detectors, etc. may, according to some set of rules, trigger the actuatable things. These devices and rules could for example turn on the lights when the sun goes down and senses that someone is home, or turn off heating when no one is home. Consumers may be familiar with few different ecosystems such as Apple Homekit, Google Nest, or Amazon Alexa. IoT have a wide array of applications outside of the home as well, such as making cities or power grids smart, making farming more efficient or in industry. [23]

IoT has over the last few years become much more prevalent. [24] The core ideas however have been around for a few decades. In the early 1990's Mark

Weiser described the vision of ubiquitous computing in the 21st century in works he did at Xerox PARC that bear a close resemblance to the IoT of today. [25, 26] Weiser's vision, that applications may seamlessly wander between computers, boards, tablets and pads, may not have been realized though. While the vision is still alive, IoT today is dominated by vertically integrated solutions dependent on the cloud. [27, 23, 1] Figure 2.1 contain a diagram of this architecture.

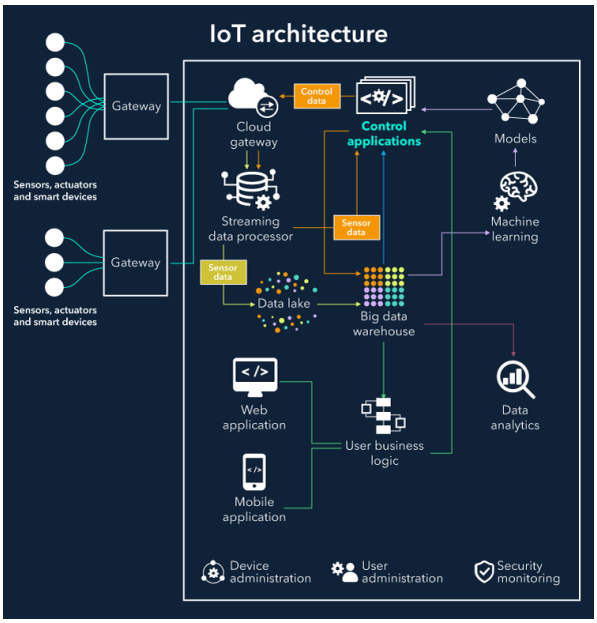


Figure 2.1: Common modern IoT architecture. [1]

IoT cameras follow a similar architecture. A general overview architecture for such a device can be found in Figure 2.2. This architecture have been synthesized by comparing product pages and user manuals for IoT cameras. [2, 3, 4] These products often store the surveillance locally. It is common to store this data either on an SD-card in simpler systems or on a networked video recording device in larger more configurable systems. The user can then usually access the live or recorded surveillance stream from an smartphone, either directly over the local network or via the cloud. (Some more complex systems have additional means of accessing the data such as web portals.)

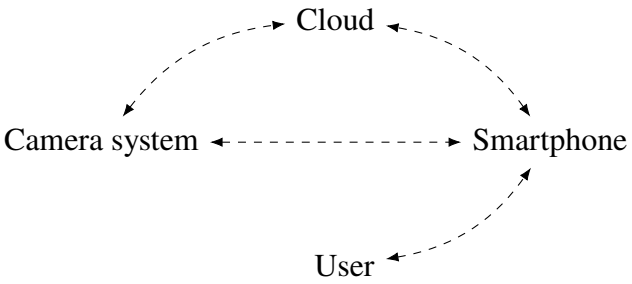


Figure 2.2: A general architecture for IoT cameras from an overview perspective. (Synthesized from IoT camera product manuals. [2, 3, 4])

2.2 Penetration testing

This section presents a pentesting methodology described by Georgia Weidman and how it relates to process described by other authors in the field.

Georgia Weidman describes in her book *Penetration Testing: A Hands-On Introduction to Hacking* penetration testing consisting of seven stages. [28] The stages are:

1. pre-engagement,
2. information-gathering,
3. threat modeling,
4. vulnerability analysis,
5. exploitation,
6. post-exploitation, and
7. reporting.

Weidman defines **pre-engagement** as the first stage and it involves discussing the coming pentest with the client in order to reach an understanding about the what, when, how, and who of the tests. This can include answering what is the pentester allowed to attack, what actions they may perform there, what kind of test is expected*, when are they allowed to operate, who should be contacted during the test if required, etc.

* Test kinds are discussed in Kinds of testing.

Then comes the **information-gathering** stage, according to Weidman. This step is about finding and collecting information that is freely available about the system under test. This could entail reading manuals or running tools such as port scanners.

The knowledge gathered is then used to build a threat model in the **threat modeling**. This model serves to identify assets within the target, and possible avenues to attack those assets. The assets can include anything that is important to the client from customer data to proprietary software. The model is used to develop strategies to attack the system. *

The pentester then proceed to the **vulnerability analysis** stage where they start actively exercise the system under test to discover vulnerabilities. The strategies developed in the previous stage is here used to guide the explorations. Weidman highlights that the pentester may use automated tools such as vulnerability scanners, but emphasize that manual analysis still is critical to verify the automated results and look in blindspot of the tools.

Once the pentester found some vulnerabilities, they go on to the **exploitation** stage where they try to exploit the vulnerabilities they found. Many vulnerabilities are common and may be attacked using pre-built exploits. Others may require new or custom exploits to be developed. Weidman highlights the tool Metasploit as valuable for this stage.

When the pentester successfully exploits a vulnerability the **post-exploitation** stage begins. During this stage the pentester survey what their exploit has gained them. Have they gained access to sensitive files? Have they gained further access to the target network? Can they now reach additional vulnerabilities? Weidman highlights the possibility of elevating privileges or using the compromised system as a pivot to attack another one. All this information together with the difficulty of the exploit is used to produce a risk assessment of the vulnerability for the client.

The final stage in Weidman's model is **reporting**. Here the pentester collect and package their findings in a manner that is meaningful to the client. This report is essentially feedback on how well the client's security posture and should be presented as any other feedback, focusing both on what is done right and what areas would need improvement.

Another author, Aditya Gupta echoes this methodology by outlining a similar one in his book on pentesting IoT devices *The IoT Hacker's Handbook*.[29] Weidman's model has also previously been used by Madeleine Berner in her thesis on pentesting a smart garage. [30]

* An exploration into the state of the art of threat modeling is done in the Threat modeling section.

2.3 Kinds of testing

How a pentester approaches a target varies depending on what information about it they have access to. These approaches can be classified as *white-*, *grey-*, or *black-box* tests depending on the level of information that is available.

Aaron Guzman and Aditya Gupta defines the terms in their book *IoT Penetration Testing Cookbook*. [5] They define black-box testing as an approach where the tester has no previous knowledge of the system under test. White-box they define as the opposite; in this approach the pentester is given full access to the inner workings of the target, e.g. source code, network-, architecture- and data-flow diagrams. Grey-box is defined as an intermediary between the extremes of the others. Knowing the application stack of the target is used as an example by Guzman and Gupta.

2.4 Threat modeling

Threat modeling is, according to A. Shostack, about using models to find security problems. [8, p. 3] Many authors, including Shostack, have written about threat modeling from different perspectives. Of particular interest for this thesis is the work by Aaron Guzman and Aditya Gupta on threat modeling in an IoT pentest setting. [5, c. IoT Threat Modeling] Their approach is similar to Shostack's, but focused on applying it in pentesting compared to Shostack's which is primarily intended for software engineers or system administrators. [8, p. xxiv] This section is dedicated to describing Guzman and Gupta's approach to threat modeling.

Guzman and Gupta's approach consists of the following six steps:

1. identify IoT assets,
2. create an IoT device architecture overview,
3. decompose the IoT Device,
4. identify threats,
5. document threats, and
6. rate the threats.

In the first step, **identifying assets**, the pentester go through what they know about the system under test and pick out it's assets. Shostack defines an asset

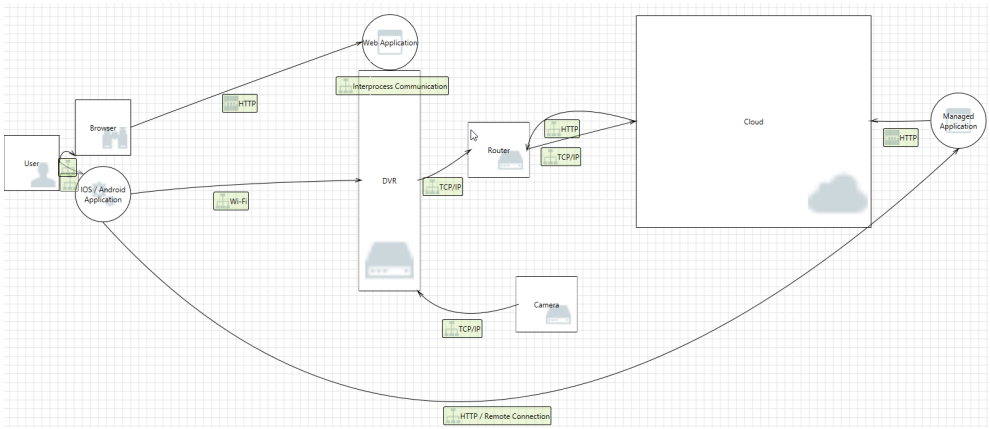


Figure 2.3: An example of an architecture diagram created in the second step. (From Guzman and Gupta's book) [5]

as something that: 1) an attacker want, 2) the owner want to protect, and 3) is a stepping stone to either. [8, p. 37] Guzman and Gupta provide the following examples (among others) in the context of IoT: 1) firmware, 2) web applications, 3) mobile applications, 4) device hardware, and 5) radio communications.

The next step in Guzman and Gupta's approach is to **create an IoT device architecture overview**. They state that the goal of this step is to document the functionality, applications and implementation of the system under consideration. To start off, the authors recommend creating a couple of common use cases. These use cases are then documented in an architectural diagram which shows details about the components and data flow in the system. (See figure 2.3.) With the diagram drawn, the pentester goes on to identify what technologies are used to implement the components in the diagram. This can include identifying what OS devices run, what protocols are used for communications or common libraries used by the software.

The third step, **decompose the IoT device**, focuses on analyzing application and protocol data flow through the system. Of special interest is trust boundaries and when a information flow across one. The diagram created in the previous step is here updated with the information uncovered in this step. (See figure 2.4.)

Once, the pentester has a good sense of the system under test they are ready to start the step of **identifying threats**. The idea here is not to find actual threats, but possible threats. There exist multiple techniques for finding threats in a system, such as *STRIDE*, *Attack Trees* or *Attack Libraries*. [8, pt. II] A more thorough presentation of such techniques can be found in the section Threat Identification. In Guzman and Gupta's approach STRIDE is used.

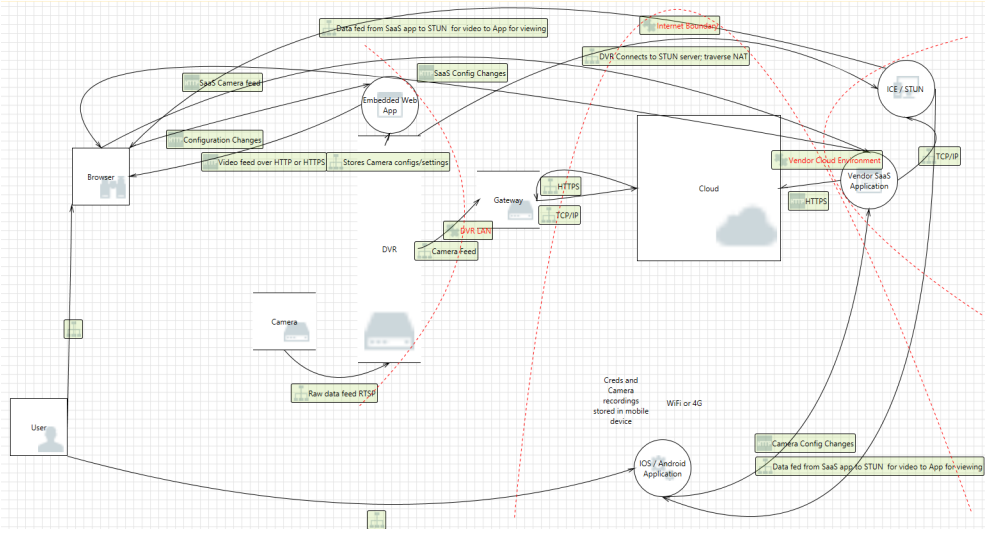


Figure 2.4: An example of an updated architecture diagram after step three. (From Guzman and Gupta's book) [5]

In the penultimate step, **document threats**, the pentester methodically documents the possible threats found in the previous step. Guzman and Gupta suggest listing the threats together with an description of the threat, what the threat targets, how the threat could be attacked and what countermeasures could be in-place.

And finally, in the **rate threats** step, each threat is given a risk rating. Like how there exist multiple methodologies for finding threats, there exist multiple methodologies to rate them, e.g. *DREAD*, *CVSS*. These and others are presented in more detail in section Threat rating. Guzman and Gupta use DREAD in their approach.

It is common that threat modelling is talked about as a linear process in the literature[8, 5]. But threat modelling can take on other forms. Some prescribe that the process should be repeated[31], others talk about an *iterative* threat modelling process[32, 33]. The major benefit of the iterative model describe is it's ability to continuously refine one's model as new facts are discovered about the system under consideration.

2.5 Threat Identification

Threat identification is the process of finding potential threats in a system. It is often done in the process of threat modelling. (See the section 2.4 for more on

threat modelling.)

There exist multiple approaches for identifying threats. Each approach have it's own distinct advantages and disadvantages. This section will present a few approaches common in the literature, namely: *STRIDE*, *Attack Trees*, and *Attack Libraries*. Each approach will be covered in a separate subsection that details how it works, as well as containing a summary of the approach's advantages and disadvantages.

2.5.1 STRIDE

STRIDE is a acronym defined by Microsoft's security task force in an article published in 1999 written by Loren Kohnfelder and Praerit Garg. The acronym stands for *Spoofing*, *Tampering*, *Information Disclosure (ID)*, *Denial of Service (DoS)*, and *Elevation of Privilege (EoP)*. [34] There are variations on these six elements. Drew van Duren and Brian Russel present a variation for IoT systems with three additional elements: *Physical security bypass*, *Social engineering*, and *Supply chain issues*. These additional elements are not used in the authors example of the methodology however. [35]

Kohnfelder and Garg writes that "Security threats fall into the six major categories listed below." while referring to the elements of STRIDE. Adam Shostack however note that some threats are not necessarily neatly described by a single STRIDE element, which makes classification by the system more complex. Classifying the threats might be unnecessary though, as the goal is identification. Thus, he argues, that STRIDE is a useful tool to think about and elicit threats for a system, but that the threats that is found does not need to be classified by it. [8, p. 64]

The core idea in STRIDE is to consider each element of a system for threats of each element in STRIDE. This could look like considering what spoofing-, tampering-, ID-, DoS-, or EoP-threats there are for each process, data store, or data flow in a architectural diagram*. [8, p. 10] However, there may be multiple threats that may be elicited by a given STRIDE element for a given component. The question of when to stop therefore arise. Shostack mentions three exit criteria:

1) a threat for each STRIDE element has been elicited,
2) a threat for every component of the system has been elicited, and
3) a criteria specific for the STRIDE variant used.
   (Variants discussed a few paragraphs further down.)

* Described in section 2.4

Shostack notes that stricter criteria that require more threats to be found naturally are harder to meet, but consequently are more comprehensive. However, no matter the criteria, having met it does not imply that all threats have been found.*. [8, p. 85]

There exist multiple variations of the traditional STRIDE defined by Kohnfelder and Garg. Shostack details STRIDE-per-element and STRIDE-per-interaction. These versions often prescribes what elements to iterate over, and which STRIDE elements to consider for each type of element. The purpose of these variation are to make STRIDE more prescriptive and have some meaningful exit criteria. [8, p. 78]

STRIDE-per-element recognize that all STRIDE elements are not relevant for all kind of elements in the architectural diagram, e.g. it might not make sense in some context for a DoS-attack to be made against external entities such as users. This insight allows for skipping irrelevant STRIDE elements for some diagram elements, and thus making the method easier to apply. An example table indicating what STRIDE elements are relevant for a given diagram element can be seen in Table 2.1. Some analysis is however required to determine which STRIDE elements are relevant for each diagram element in a given context. Shostack list this as one of two major issues with this variant. The other issue he acknowledge is that the same threat may appear in multiple places for a given model. The method though has the advantage of being prescriptive without telling the user to look for certain threats, such as XSS, SQL-injection, etc. It also has a somewhat natural exit criteria. Shostack judges that one is doing "resonably well" once a threat for every relevant STRIDE element for every diagram element is found. He notes that one can do a bit better if for each mitigation for each threat another threat is found. [8, p. 78]†

STRIDE-per-interaction builds upon STRIDE-per-element, but acknowledge that threats do not manifest for some element in a vacuum, but rather in the interaction between elements. The items considered in STRIDE-per-interaction therefore are tuples of $(origin, destination, interaction)$. This leads to that what STRIDE elements that are relevant and should be focused on need to be specified in terms of these tuples. This therefore exhibit the same issue as STRIDE-per-element that some judgement needs to be made about what STRIDE elements are relevant for each interaction. Further, due to number of

* In a way this statement can be thought of as a reformulation of the following quote by Dijkstra: "Program testing can be used to show the presence of bugs, but never to show their absence!" [36, p. 6]

† STRIDE-per-element seem to have been first published by Larry Osterman in 2007 in his blog post titled *Threat Modeling Again, STRIDE-per-element*.[37] Shostack's description of the method appear to be a few iterations down the line though.

	S	T	R	I	D	E
External Entity	x		x			
Process	x	x	x	x	x	x
Data Flow		x		x	x	
Data Store		x		x	x	

Table 2.1: An example of a table indicating what STRIDE elements are important for a given diagram element from Microsoft.[8, p. 78]

ways the interaction tuple can be constructed, such reference table is naturally larger than one for STRIDE-per-element. Despite this, Shostack notes that this method leads to the same number of threats as STRIDE-per-element, although they may have additional context making them easier to understand. Shostack issue the same judgment regarding exit criteria for STRIDE-per-interaction as for STRIDE-per-element, i.e. that having found one threat per cross in the reference table is good, but it is better if a threats for mitigations also are found. Shostack attributes Larry Osterman and Douglas MacIver at Microsoft for creating this variant.[8, p. 80]

Since STRIDE does not specify what threats to look for, but only types of threats, it relies to some extent on the person using it's experience of the system under consideration and what can go wrong with it. While this may make it harder to use for people unfamiliar with the area, the open ended nature can allow the users to be more creative about what threats they find. This could manifest as being able to find novel threats in a specific system context, as one example. [8, p. 61]

2.5.2 Attack Trees

Attack trees is a methodology to break down how some asset might be compromised. [38, 6] The core idea is to represent goals and attacks as a tree. The root node represents the goal of the attack. Nodes other than the root node represents direct prerequisites for gaining access to that node's parent. [6] An example of an attack tree can be found in figure 2.5. In that figure we find `Open Safe` as the root node and the goal. By reading the children to that node we find that we can open the safe by either picking it's lock, learn it's combination, cut it open or that it is installed improperly. Further down the tree we find the node `Eavesdrop`. It is marked with `(and)` to indicate that we must both listen to a conversation and get the target to state the combination. These exemplify the

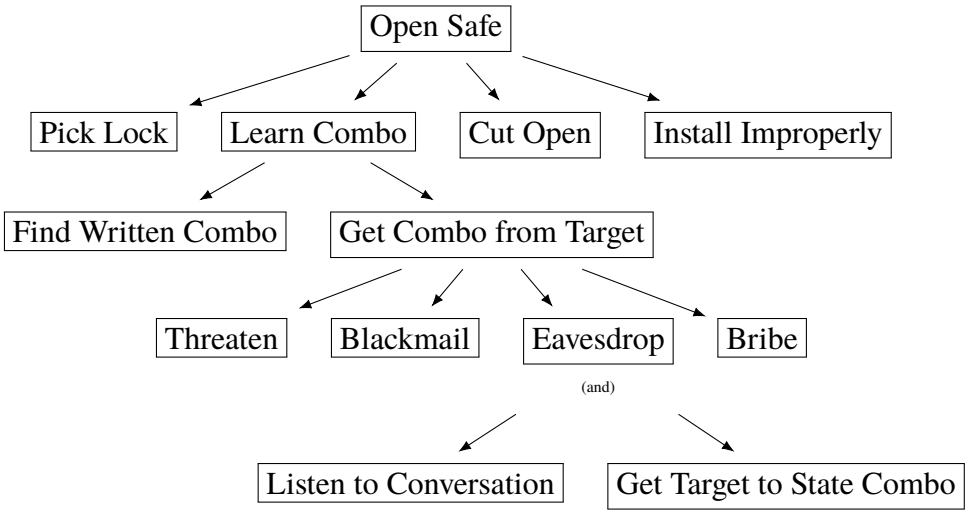


Figure 2.5: An example attack tree.
(Taken from Schneier's introduction on attack trees.[6])

concept of *or-* and *and-nodes* in attack trees. Their children represent a set of steps that either independently or collectively respectively achieve a goal. [6]

Once an attack tree has been built, the nodes can be marked with different properties to help get a better understanding of the attacks in the tree. These properties can help answer wether a particular attack requires specialized skills or tools, is expensive, is time consuming, etc. In his introduction on the subject, Schneier proposes a method of assigning these values to the nodes. He proposes that leaf nodes are assigned some value and the value of non-leaf-nodes are derived from their children. An example is to give the leaf nodes some boolean value, such as whether the node is possible. The values could then be propagated up the tree by assigning an or-node as possible if any of it's children is possible, and that an and-node is possible if all of their children are possible. Figure 2.6 contain a version of figure 2.5 where the nodes have been marked as possible or impossible. The values assigned does not need to be boolean, but can be chosen from other discrete or even continuous sets. Functions for assigning the values of or- and and-nodes must however be chosen to match the selected set of values and their context, e.g. if the leaf nodes are assigned an approximate cost the function could be the minimum or sum of the cost of it's children for or- and and-nodes respectively. These value assignments can be used to answer simple questions, such as whether the root goal is possible or not, or more complex questions such as what the cheapest possible attack is by combining value assignments. [6]

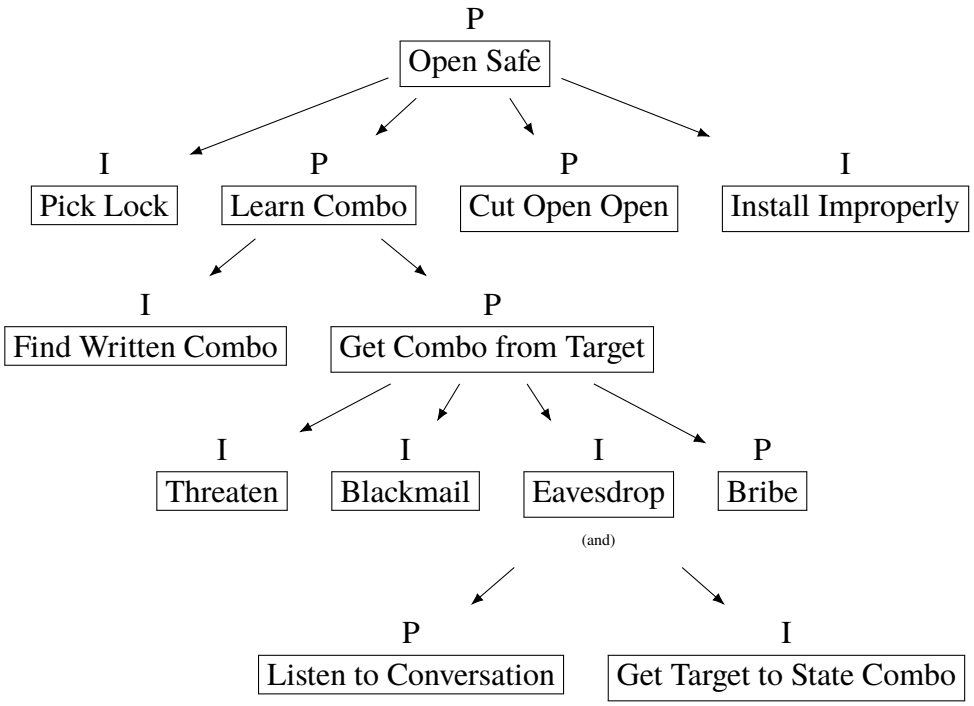


Figure 2.6: The same attack tree as in figure 2.5 but each node is marked as either possible (P) or impossible (I). The leaf nodes were given their value manually. The value of the or- and and-nodes were then derived based on their children. Or-nodes were given the value of possible if any child node were assigned as possible. For and-nodes all their children had to be assigned as possible for it to be assigned as possible.
(Taken from Schneier's introduction on attack trees.[6])

When performing threat identification one could use an already constructed attack tree that applies to the system under consideration or construct a new custom one if no suitable pre-made is available. Schneier outlines a simple recursive process to create these trees. The steps of this process are: 1) identify goals (each goal will be the root of an attack tree), and 2) for every non-trivial not expanded node identify possible attacks and let those be new children nodes to the current node. He also highlights that available pre-made trees can be used as sub-trees where suitable. [6]

Attack trees can be useful for finding threats and organizing them in a structured manner. That structure can then be used to answer complex questions about the threat landscape. They are not a panacea however. Shostack lists the following three challenges: 1) completeness, 2) scoping, and 3) meaning. Completeness is a challenge due to issues of knowing whether all threats to some asset are found. Shostack advises that this challenge could be countered by additional brainstorming, literature review, or augmenting the attack tree method with another threat identification method such as STRIDE. The challenges of scoping and meaning concern the usability of attack trees. Shostack writes that attack trees tend to find many issues that are not actionable for system developers and thus can have issues regarding scope. (However, an attacker of the system might of course still be interested in threats that that are not actionable for the developers of said system.) The last challenge, meaning, stems from a lack of standardization of representation of attack trees, which could make them hard interpret for those who are unfamiliar. [8, c. 4]

2.5.3 Attack Libraries

An attack library is a collection or taxonomy of attacks or areas of attacks. An attack library may have different levels of abstractions, e.g. Adam Shostack put checklist of concrete attacks and STRIDE as both attack libraries, but of different level of detail. Attack libraries may also have different scopes. Some may target attacks against the web, mobile apps, IoT, networks, etc. Other consideration might also influence the design of attack libraries. Shostack comments that different variations could be useful in different situations, but that there might be a sweet spot that are more prescriptive than the abstract STRIDE, but not too rigid to inhibit creative and critical analysis of a system that checklist might. [8, c. 5]

Shostack describes a few attack libraries in his book on threat modeling. He discusses checklist and literature reviews as generic basic attack libraries but also CAPEC and OWASP Top Ten as specific examples. Shostack writes

that a checklist can be a useful tool if there are some common problems that frequently are missed. However, checklists are rather poor at finding threats not on them. They are therefore of limited use as a tool for encouraging critical and creative thinking about security. Nevertheless, their ability to prevent common issues means they can be used as a complement to other techniques in threat identification. Literature review is the other general attack library approach Shostack writes about. The idea in this approach is that the threats encountered by others, that worked on similar systems, may be relevant for the current system under consideration. It may therefore be helpful to read or otherwise learn about what others have done and what threats they may have faced. Shostack notes that threats found through this approach may be specific to original system's situation or be an instance of class of issues, and that some effort may be necessary to abstract these threats, e.g. if an earlier system have suffered from that database information could be leaked through a search query, one may deduce that SQL injection attacks should be considered. Finally Shostack describes two specific attack libraries, what domains they can be useful in and how they can be used. The OWASP Top Ten is a library focused on the web. CAPEC on the other hand is a comprehensive list of common attack patterns. In the discussion he describes that the items in an attack library often are more abstract than those in an checklist. The attack library could thus encourage elicitation of entire classes of threats, and avoid limiting that only specific threats are considered. The lines between a checklist and an abstract attack library is not always clear though. It is therefore important to review any library to ensure it can fulfill the intended purpose. Shostack concludes that attack libraries relevant for the system under consideration can be valuable, if available. [8, c. 5] Another attack library of note not mentioned by Shostack is ATT&CK by MITRE. ATT&CK is structured around what is called *tactics*. Tactics describe why an adversary would like to perform some action, e.g. discovery, lateral movement, or privilege escalation. ATT&CK then, for each tactic describe a number of *techniques* an adversary can use to achieve that tactic. [39]

In summary, attack libraries can be great tools to prevent common and known problems. A specific library such as a checklist may be used to prevent specific issues. A library that is a bit more abstract, such as CAPEC, may be on the other hand be more useful to elicit new threats of known kinds. A literature review could allow one to learn from others, but some effort may be required to adapt their knowledge to the particular system under consideration. Common for all attack libraries is the tradeoff between ease of use and ability to elicit new threats. A specific library such as checklist may be easy to implement.

Contrary, a general framework such as STRIDE may encourage elicitation of new threats, but the result is dependent on the experience of the user.

2.6 Threat rating

The threat modeling process may find many potential threats. Therefore, once potential threats have been identified, they can be rated. This rating can be used to prioritize the threats, i.e. it is probably sensible to start vulnerability analysis searching for the most serious threats.

There exist multiple methods for rating threats. DREAD is a common risk assessment model, recommended for example by Guzman and Gupta in their book on IoT pentesting. [5] The name DREAD is an acronym from the following terms: 1) damage potential, 2) reproducibility, 3) exploitability, 4) affected users, and 5) discoverability. Each term is supposed to address a specific area of risk. Damage potential is about how serious the consequences would be if a threat would be realized. The reproducibility would reflect how easy it is to make a threat a reality in terms of reliability; a threat reliant on an unlikely race condition would typically be less reproducible than an exploit that it is deterministic. Exploitability in turn measure the required resources and/or knowledge for an attacker to realize a threat. When giving a rating for affected users, one should estimate how many users would be affected if the threat was realized. Discoverability is the measure of how likely it is that the threat is found.* When using this method each identified threat receive a rating for each term. The score given to the five terms are then averaged to give a overall risk rating for the threat. Each term is typically scored on scale from 1 to 10, where 10 is worst in context of the term, e.g. highest damage potential or the most affected users. [31]

Adam Shostack however comments that DREAD might be obsolete based on that it can be subjective. [8, p. 180] He comments the fact that the method assigns number without having some standardized scale can make the risk assessment seem more algorithmic than it is. [40, p. 7] He instead advocates approaches such as a bug bar, probability/impact assessment, or FAIR. Bug bar is an approach that has an a few different categories of criticality from low to critical with associated sets of criteria. If a threat matches the criteria for a category it is classified to have that risk.[41, 8, p. 181] In probability/impact assessment the probability of a risk and the impact if the risk actualizes are taken into account. The assessed risk increases if the likelihood and/or the

* Often it is good to assume that the threat will be found!

impact of the risk increase.[42] FAIR is highly structured method that can achieve reproducible assessments. Broadly it focuses on identifying an asset at risk, evaluating how often the risk would occur and estimating the impact of the risk occurring in a prescribed manner.[8, p. 182]

Guzman and Gupta mention Common Vulnerability Scoring System (CVSS) as a fifth approach to threat rating in their book on IoT pentesting. [5] CVSS is a granular framework for assessing the severity of software vulnerabilities. It has three groups of metrics 1) base, 2) temporal, and 3) environmental. The metrics in the base groups aim to capture how vulnerable the system under consideration is irrespective of time or environment. The metrics in the temporal and environmental groups can then modify the base score depending on factors that may change over time or between different deployments. Maturity of exploits or remediation are examples of temporal factors. Deployed safeguards (or lack thereof) in the environment such as firewalls can contribute to the environment group. It also considers the required level of security, i.e. some problems are catastrophic for an organization while others can go unnoticed and only cause limited harm for a long while.[43]

2.7 Summary

This chapter begun covering the basics of IoT. The following sections then covered multiple authors perspective on pentesting and it's intricacies. It started with an overview of the process based on work by G. Weidman and A. Gupta. The threat modelling step of that process concealed an, itself, immense field. Again after an overview, this time substantiated by literature from A. Shostack, A. Guzman and A. Gupta, the chapter concluded with explorations of the threat modelling steps threat identification and ranking.

This wide exploration of the field of pentesting provides the foundation for the method selection performed in the next chapter.

Chapter 3

Methodology

This section discuss the methodology selected to answer this thesis's research question. The chosen method should be achievable given the time and resource constraints afforded this thesis while answering the question as completely, precisely and accurately as possible.

The choice of kind of security review done in this study is dominated by the fact that the thesis is done independently. This mean that design documents, firmware source code and the like are not available. Hence, black-box testing and pentesting become the necessary choice. The other choices are discussed in the coming subsection.

3.1 Camera Selection

The central question of this thesis covers an entire market segment. To evaluate, or even find, every smart home camera on the market would be infeasible. Sadly, time is limited. Therefore, this thesis is limited to a case study of a single camera. The result thus lose some generalizability. That is, this thesis can not definitely disprove the existence of vulnerabilities in products in the market. However, it can show their existence. Furthermore, the impact of the conclusion also depends on how well the tested camera represents the market. Conclusions about a camera with a higher market share do have a larger impact than ones about a camera that is barley used. To maximize the impact of the result, the camera selected ought to be as representative of the market as possible. As a proxy popularity was used. The Swedish price comparison site Prisjakt was used for their popularity metric. To further improve the impact of the camera selection, a camera that is not well studied ought to be selected.

3.2 Pentesting methodology

Georgia Weidman's pentesting methodology or variations of it is the prevalent choice in the literature as discussed in section 2.2. The methodology is catered to a security professional being contracted to do a review for some company though. Some slight modifications are thus required to adapt it. More precisely, the pre-engagement step has a focus on defining scope together with the client. Discussing scope with a client is not applicable for the thesis and the study's scope is defined by other means. Thus, the pre-engagement step is dropped. The final step, reporting, also exhibit the issue of being client focused in the Weidman model. Scientific endeavours still need to communicate their findings (e.g. this report), but having an step in the methodology explicitly dedicated to writing a report is superfluous. The remaining steps:

1. information gathering,
2. threat modelling,
3. vulnerability analysis,
4. exploitation, and
5. post-exploitation

are kept unmodified. Most of these steps are explained thoroughly in Weidman's model, but section 3.3 will explore the step of threat modelling and related choices in greater detail.

The basic structure of Weidman's model is modified to incorporate the ideas of iterative threat modelling*. It lies in the nature of black-box pentesting that information about the system under consideration is revealed over time. An iterative approach is therefore a natural fit. Thus, instead of completing the vulnerability analysis and then proceeding developing exploits for all vulnerabilities found, this thesis instead analyse one threat, and if the vulnerability is found, directly proceeded with the exploitation step. The post-exploitation step, which is described as taking stock of how ones position have shifted after the exploit of a vulnerability, is then a natural place to update the threat model and opportunity to re-prioritize as well as adding or removing potential threats. If the analysis concludes that the potential vulnerability imagined by in threat in fact does not exist, then vulnerability analysis can commence directly with the next threat. Figure 3.1 contains a diagram over the modified process.

* Discussed in section 2.4.

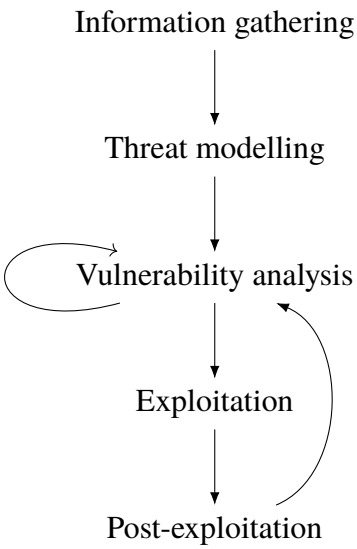


Figure 3.1: Chosen pentest process for thesis.

This process can then proceed until all threats discovered in the threat model or subsequent post-exploitation-update have been explored, or (more likely) until the available time is spent.

3.3 Threat modelling

Again, the literature is overall in agreement as discussed in section 2.4. Different authors describe the threat modelling process differently, but each version mostly consists of the same components in mostly the same order. The differences of kind between the process described by the different authors can be ascribed to the different intended audiences. The process described by Guzman and Gupta is tailored for IoT hacking and is thus the most actionable in the context of this study. Therefore it is selected as the threat modelling method of choice for this thesis.

Madeleine Berner also used Guzman and Gupta's threat modelling process as well Weidman's pentesting method in her excellent master thesis where she performed a pentest on a connected garage.[30] She argued that the sixth step, threat rating, of threat modelling process where redundant given that the pentesting method's vulnerability analysis stage. I do not concur on the basis that, as I understand it, they serve different purposes. As I have interpreted the literature, the threat rating step function is to prioritize the hypothetical threats found in the threat modelling process, while the vulnerability analysis role is

to examine whether the hypothetical threats are real.

The Guzman and Gupta threat modelling method was thus employed unmodified in this thesis. The method do require method selection for both the threat identification and threat rating steps, which will be discussed in section 3.4 and section 3.5. Further, the second to last step, document threats, do not have it's dedicated section as the other steps, as the documentation comes naturally in the form of this report.

3.4 Threat identification

An exploration of the literature were made in section 2.5. That exploration revealed that the choice of threat identification is not as given as choosing methodologies for pentesting or threat modeling. STRIDE, attack trees, and attack libraries are possible alternatives, that can be used either as stand alone solutions or together.

Attack libraries has the advantage of being simple to apply, if there is a suitable one available. The simplicity in application though also is one of this methods weaknesses, as elicitation of new attacks may be overlooked. If a suitable attack library is not available, then it is possible to create one, by literature review for example. But such task requires great care and expertise as omitting an attack or kind of attack means that the created library will do so too. As noted in subsection 2.5.3 Attack Libraries, there are multiple attack libraries that might be applicable in this thesis, i.e. CAPEC, OWASP IoT top ten 2018 and ATT&CK. While both CAPEC and ATT&CK are comprehensive, OWASP IoT top ten 2018 has the advantage of being simple and concise.

Attack trees are similar to attack libraries, both positive and negative aspects. Like attack libraries are attack trees easy to apply, encourage reuse, but they also risk missing vulnerabilities if the user do not carefully consider the context of the system under consideration. The literature on attack trees does seem to encourage such critical considerations though, but still emphasize the possibility to slot in earlier trees as sub-trees in the current context. Attack trees most significant advantage is their structure though. Given an accurate and complete attack tree, additional aspects of the attacks may be modelled (e.g. cost, complexity, etc.), which enable complex queries regarding the attack surface to be made.

STRIDE can, from one point of view, be regarded as an abstract attack library since STRIDE at it's core is a collection of types of vulnerabilities to look for. However, the difference in level of abstraction becomes a difference in kind as the STRIDE elements guide the user in their search for threats,

rather than enumerating which are possible. This focus on elicitation has the implication that the required expertise and knowledge of the user is greater than they are with attack libraries. Although, STRIDE, and especially it's variants such as STRIDE-per-element, does give more guidance in eliciting threats than both attack trees and attack libraries. Further, STRIDE seem to be the preferred method in the literature. It is presented as the default in Adam Shostack's book on threat modelling, [8, p. 10] and as the methodology of choice by Guzman and Gupta in their book on IoT pentesting. [5]

In the context of these other methods, attack trees does not bring new capabilities regarding threat identification. STRIDE has superior guidance in eliciting novel threats, while attack libraries shine in ensuring common issues are not overlooked. The structure of an attack tree does provide capability of querying found threats that the others can't, although such capability does not help find the vulnerability. A combination of STRIDE and an attack library is therefore a compelling option. OWASP IoT top ten 2018 is the favoured over CAPEC and ATT&CK for this role since simplicity and conciseness lends itself to prevent specific common issues. Such a combination would benefit of the work of others that has compiled the attack library, which would help ensure that common issues are not overlooked, but would also be able to leverage STRIDE to consider the specifics of the current system under consideration.

The threats listed in the attack library was complemented with an exploration of earlier work describing vulnerabilities. As mentioned above, attack libraries omit every threat they do not list. Therefore, in the pursuit of being comprehensive, earlier work related to the selected camera, IoT cameras in general and IoT in general was examined. The goal to summarize what cameras and IoT historically been vulnerable to. This, along with an exploration of attack libraries is presented in their own chapter (chapter 5 Related vulnerabilities). The threat identification phase of the threat model can then consider the threats summarized there in the specific context of the camera under consideration, as well as elicit new threats.

As discussed in subsection 2.5.1, STRIDE has multiple variations. The question of which exit criteria to use also become pertinent. The three variations discussed are: 1) STRIDE-per-element, 2) STRIDE-per-interaction, and 3) every STRIDE element for every diagram element. * Given that the primary goal of threat identification in the context of this thesis is to elicit possible vulnerabilities, the extra scaffolding necessary for STRIDE-per-interaction compared to STRIDE-per-element appear to not be required. The potential

* The third variation is really a special case of the first, but is mentioned as separate as it is the original formulation of STRIDE.

extra context for the found vulnerabilities may be a great boon for larger team, but forgoing it is not concern for this thesis. The thesis use the STRIDE focus matrix used by Microsoft (Table 2.1) as a starting point. Selecting STRIDE-per-element also gives a natural choice for exit criteria, namely striving for finding a possible threat for every entry in the threat focus matrix.

3.5 Threat rating

The literature surrounding threat rating suggest multiple possible methods. The ones discussed in section 2.6 Threat rating are:

1. DREAD,
2. bug bar,
3. probability/impact assessment,
4. FAIR, and
5. CVSS.

Bug bar, FAIR and CVSS are structured approaches, that generally should be able to give consistent ratings from different practitioners. DREAD may seem structured given it's categories, but is just guidance to achieve an assessment. The methodology lack standardized criteria for how different scenarios should be scored, and can thus be dependent on the particulars of how it is applied. Probability/impact assessment is fundamentally similar, but has less guidance. Adam Shostack argues that the guidance given in DREAD may be confused for the predictability that the structured methods can provide, and that the less prescriptive probability/impact assessment method may avoid such pitfall. But as the main purpose for rating the possible threats in the context of this thesis is for prioritization, having a method capable of delivering predictable result across time and users is not the main concern. For the purpose of ordering the found threats, their rating only need to be comparable, and not have much meaning beside that. The simplicity, but guidance of DREAD thus become the natural option.

3.6 Methodology summary

In summary, the process that was used in this thesis is a modified version of Weidman's pentesting methodology. A study of earlier vulnerabilities however

proceed this. As deliberated in section 3.4 Threat identification, this section summarize vulnerabilities historically relevant for the current pentest and attack libraries. For the threat modelling step, Guzman and Gupta's process was used. The threat identification method of choice is STRIDE, with special care to account for the vulnerabilities from the related vulnerabilities chapter, beyond eliciting novel ones. DREAD was used for threat ranking. Finally, the camera to evaluate shall be selected based on popularity as a proxy for market relevance. An outline of the complete pentest method can be found in figure 3.2.

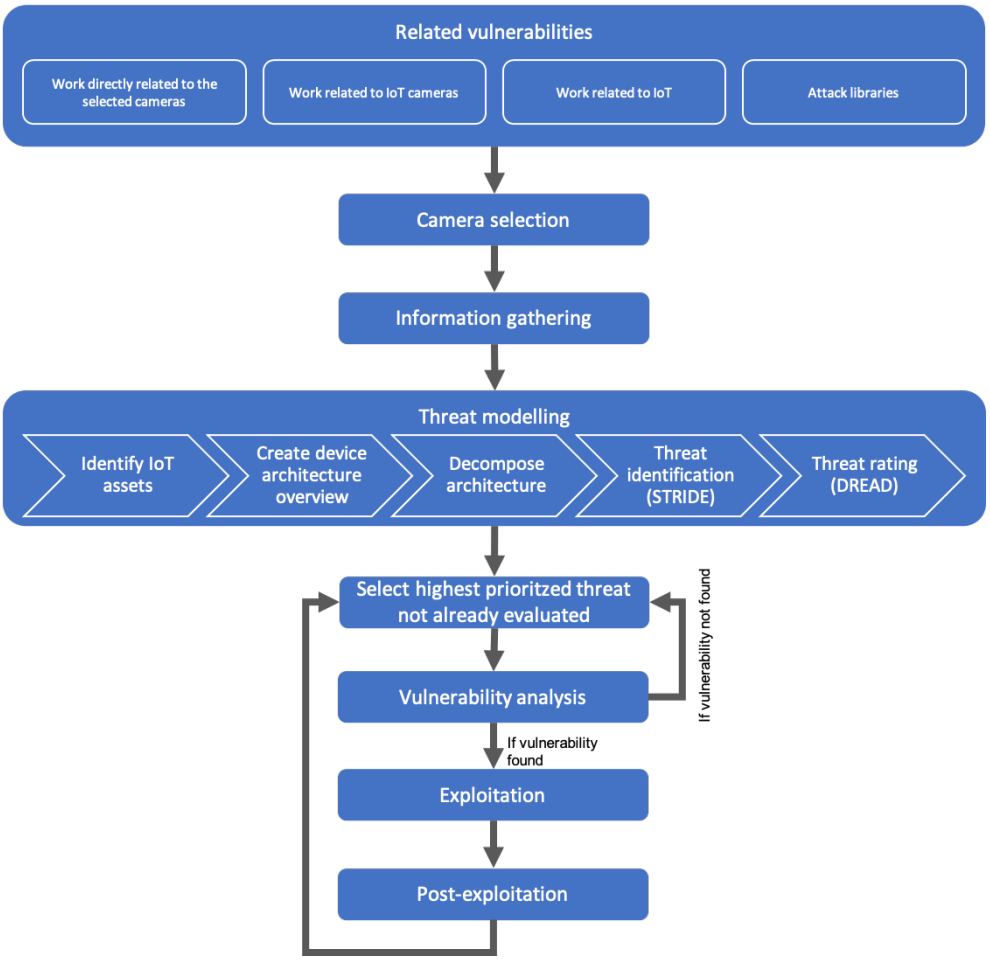


Figure 3.2: Outline of chosen pentest method.

Chapter 4

Selected systems

As discussed in section 3.1, the Swedish price comparison site Prisjakt is used for their popularity metric. At the time of selection, the top 10 most popular cameras were, in order:[44]

1. TP-Link KC200,
2. Xiaomi Mi Home Security Camera 360,
3. Arlo Pro VMS4230P,
4. TP-Link KC120,
5. Arlo Pro 3 VMS4240P (2st),
6. TP-Link Tapo C200,
7. Eufy eufyCam (2st) + HomeBase,
8. Netatmo Presence,
9. Arlo Essential, and
10. Arlo Pro 3 Floodlight Camera.

While the TP-Link KC200 according to this list is the most popular camera, it cost significantly more than than the second entry, Xiaomi Mi Home Security Camera 360. It also advertise being able to turn the camera 360°, which TP-Link KC200 does not. Preliminary research indicates that there is no significant body of work evaluating this camera from a security perspective either.

The Xiaomi Mi Home Security Camera 360 is selected as the camera to be considered in this thesis due to it's high popularity, competitive price and features. An image of the camera can be seen in Figure 4.1.

Figure 4.1: The selected camera.

Chapter 5

Related vulnerabilities

This chapter will explore previous work relevant for the work at hand and the vulnerabilities they have found. The purpose of this study is to reveal vulnerabilities that affect might the camera evaluated in this thesis. The most obvious relevant work are it that is directly related to the selected camera. Almost as relevant is work related to any and all security cameras, since all cameras likely have had to solve the same or similar problems.

5.1 Directly related

CVE Details has no records of known vulnerabilities or exposures for Xiaomi Mi Home Security Camera 360° 1080P. There were however an issue late 2019/early 2020 where an issue allowed some users that had connected their camera to the Google Nest hub could view other users camera feed on said hub. [45] In a statement from Xiaomi they say:

> Upon investigation, we have found out the issue was caused by a cache update on December 26, 2019, which was designed to improve camera streaming quality. ([45])

That the issue reportedly were caused by a cache update indicates that there were not a issue with the camera itself or it's firmware, but only with the related cloud infrastructure.

The final directly related previous work found are two GitHub repositories which allegedly contain modified firmwares that expose additional services, such as SSH, SFTP, RTSP, and more. [46, 47]

5.2 Related to IoT cameras

There have been a lot of work examining the security of other smart home cameras, IP cameras, or other similar variations. [16, 17, 18, 19, 20, 21] Data transmissions with weak or non-existent encryption, and exposed services with standard or easily cracked passwords were the two most common vulnerabilities across these works, occurring in [21, 20, 18, 19] and [16, 20, 19, 17] respectively. The next most common vulnerabilities were having unencrypted data storage (on device or mobile phone) and being susceptible to accessing confident data by directly browsing some address space of brute-forcing addresses, which were demonstrated in [18, 20] and [20, 16] respectively. A few other vulnerabilities were mentioned by [16, 21]. A summary of vulnerabilities identified in these works can be found in Table 5.1.

	Vulnerability	**From**
1	Weak or non-existent encryption	[21, 20, 18, 19]
2	Default or easily cracked passwords	[16, 20, 19, 17]
3	Unencrypted storage	[18, 20]
4	Guessable or brute forceable address space	[20, 16]
5	Authentication bypass	[16]
6	Privilege escalation	[16]
7	XSS	[16]
8	Packet flood attack	[21]
9	Replay attacks	[21]
10	DNS spoofing	[21]
11	Reflection attacks	[21]

Table 5.1: Vulnerabilities found in works on IoT cameras in general.

Further, Li et al. have also demonstrated that just analysing data transmission rate (i.e. just the amount of data transferred at a given time) can be enough to infer sensitive information, such as whether a camera owner leaves the home as usual or, in certain circumstances, when the activity captured by the camera changes. [48] Notable is that most of these vulnerabilities are enumerated on the OWASP IoT top 10[9].

5.3 Related to IoT

Here we look at work related to vulnerabilities in general.

Open Web Application Security Project (OWASP) is an organization that does a lot of work related to spread information around common vulnerabilities in a rang of different domains, such as web and (of special interest here) IoT. Table 5.2 contains OWASP's top 10 IoT threats of 2018. It is not uncommon that IoT devices incorporate some web technology in some manner. OWASP's top 10 web threats can therefore also be of interest. These can be found in Table 5.3 While this list is intended to address threats common in the web space, some of it's entries (e.g. 1) Injection and 2) Insecure Deserialization) are relevant threats for most software.

	Threat Title
1	Weak, Guessable, or Hardcoded Passwords
2	Insecure Network Services
3	Insecure Ecosystem Interfaces
4	Lack of Secure Update Mechanism
5	Use of Insecure or Outdated Components
6	Insufficient Privacy Protection
7	Insecure Data Transfer and Storage
8	Lack of Device Management
9	Insecure Default Settings
10	Lack of Physical Hardening

Table 5.2: The OWASP IoT top 10 threats 2018.[9]

	Threat Title
1	Injection
2	Broken Authentication
3	Sensitive Data Exposure
4	XML External Entities
5	Broken Access Control
6	Security Misconfiguration
7	Cross Site Scripting (XSS)
8	Insecure Deserialization
9	Using Components with Known Vulnerabilities
10	Insufficient Logging & Monitoring

Table 5.3: The OWASP web top 10 threats 2017.[10]

Neshenko et al. explored and summarised a large swathe of IoT security research

in 2019. They looked at a diverse set of topics, including the layers of an IoT architecture, attacks against it, the security impact of them, and more. Their research yielded nine classes of IoT vulnerabilities. These are summarised in Table 5.4. [11]

	Vulnerability Class
1	Deficient physical security
2	Insufficient energy harvesting
3	Inadequate authentication
4	Improper encryption
5	Unnecessary open ports
6	Insufficient access control
7	Improper patch management capabilities
8	Weak programming practices
9	Insufficient audit mechanisms

Table 5.4: Classes of IoT vulnerabilities identified by Neshenko et al. [11]

5.4 Summary

While the identified research and other work directly related to the camera under test are at most of very limited supply, there are a rich body of work related to IoT security cameras in specific and IoT in general. In this chapter have we summarised six works that are related to IoT cameras, two attack libraries from OWASP as well as one article summarising security research for the general field of IoT. The congruence in the identified vulnerabilities between the works is striking. Not only does the same vulnerabilities show up in multiple surveys of cameras, but they are also mentioned in the context of wider IoT, indicating that IoT camera security does not differ to a large extent from generic IoT security. Table 5.5 contain a coalesced summary of the vulnerabilities and threats identified in this chapter.

	Threat/Vulnerability	From
1	Deficient physical security	Table 5.4: 1, Table 5.2: 10
2	Insufficient energy harvesting	Table 5.4: 2
3	Inadequate authentication	Table 5.4: 3, Table 5.1: 5,10 Table 5.3: 2
4	Improper encryption	Table 5.4: 4, Table 5.2: 7 Table 5.1: 1,3
5	Unnecessary open ports	Table 5.4: 5, Table 5.2: 2
6	Insufficient access control	Table 5.4: 6, Table 5.2: 1 Table 5.1: 2,4,6, Table 5.3: 5
7	Improper patch management capabilities	Table 5.4: 7, Table 5.2: 4
8	Weak programming practices	Table 5.4: 8, Table 5.1: 7,11,9 Table 5.3: 1, Table 5.3: 4,7,8
9	Insufficient audit mechanisms	Table 5.4: 9, Table 5.3: 10
10	Insecure Ecosystem Interfaces	Table 5.2: 3
11	Use of Insecure or Outdated Components	Table 5.2: 5, Table 5.3: 9
12	Insufficient Privacy Protection	Table 5.2: 6, Table 5.3: 3
13	Lack of Device Management	Table 5.2: 8
14	Insecure (Default) Settings	Table 5.2: 9, Table 5.3: 6
15	Packet flood attack	Table 5.1: 8

Table 5.5: Summary of threats and or vulnerabilities found in earlier works.

Chapter 6

Information gathering

This section will present the first step of the penetration testing process, i.e. information gathering. Information gathering, as discussed in section 2.2 Penetration testing, is about gathering freely available information. This could be done by reading manuals or running tools such as port scanners or network loggers. The knowledge learned doing these activities then inform the later steps.

Before commencing, the specific camera model used is `MJSXJ05CM` and it ran firmware version `3.5.1_0052`.

6.1 Manual & product website

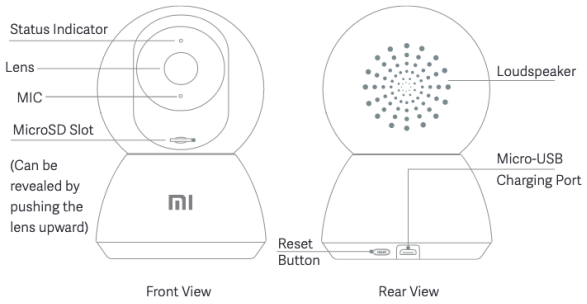


Figure 6.1: *Xiaomi Mi Home Security Camera 360° 1080P* external overview. [7]

Reading the manual[7] for the *Xiaomi Mi Home Security Camera 360° 1080P* we learn that the camera is equipped with:

- a camera,

- a status indicator light,
- a microphone,
- a MicroSD slot,
- a loudspeaker,
- a Micro-USB charging port, and
- a reset button.

Further, it explains that that the camera powers on when connected to power as well as stating that the camera is (only) connected via Wi-Fi IEEE 802.11 b/g/n 2.4 GHz. The manual also contain explanation of what different states of the indicator light means. Table 6.1 contain the complete mapping provided by the manual. Further, the manual notes that the camera can be reset to factory settings by pressing and holding the reset button. (The data saved to the MicroSD card is supposedly not deleted by this procedure.)

Light status	**Camera state**
Steady blue	Normal operation
Flashing blue	Network error
Flashing orange rapidly	Waiting for network connection
Flashing orange slowly	System upgrade in progress

Table 6.1: *Xiaomi Mi Home Security Camera 360° 1080P* indicator light explanation.[7]

The manual finally contain explanation and instruction how to use the cameras features. The features detailed are:

1. real-time surveillance,
2. night-vision mode,
3. recording & playback,
4. automatic surveillance, and
5. shared access.

The real-time surveillance feature allow the user to view the video feed and control the camera manually in real-time through the companion app. The night-vision mode is enabled by infrared light emitting elements, allowing the camera to see in the dark without illuminating the scene in visible light. The recording and playback feature is enabled when a MicroSD card is installed. When enabled, the camera can record its video feed, which the user then can playback through the companion application. The product website mentions that the camera should support storing to a local NAS device. Attempts to enable it in the companion application however failed, thus this feature is ignored.* The automatic surveillance feature allow the user to configure a schedule where the camera automatically sweep through camera angles. In automatic mode, the camera can also detect movement, and when it does, record a video and send a notification to the owner. The final feature detailed in the manual is the shared access feature, which enable the camera's owner to invite family or friends to view the camera remotely through the app. The camera box also makes it clear that it works with the third-party services Google Nest and Amazon Alexa.

A noteworthy exception from the manual is that it does not explain how the camera update process (that is alluded to in the explanation of the indicator light) works. However, the update mechanism became apparent when using the companion mobile application. When an update where available, the user would be prompted to install it when opening the application. The user could also manually check for updates through the application. It is also possible to configure the device to automatically install updates.

6.2 Port scanning

A comprehensive port scan were then made. The network setup for the scan was as can be seen in Figure 6.2. Notably, the network setup does not contain a firewall or similar measures between the scanning computer and the scanned camera. Further, there is only one hoop, i.e. the router, between the computer and the camera. The likelihood that a package sent by the computer to the camera or vice versa is not delivered as it should in a timely manner is negligible. The tool `nmap` were used to run the port scan. Exactly how it were used can

* Three attempts were made where three different computers acted as the NAS device. One ran Windows, another Mac OS and the last were a router. The first setup were discoverable in the application, but failed with the following error once credentials had been supplied: "Couldn't get storage location, try again?". The shares from the other two setups were not even discoverable from within the application.

be seen in Listing B.1. In short, a TCP SYN, UDP and SCTP INIT scan over the entire port range were attempted. All TCP SYN scan attempts resulted in resets, indicating that the port were closed. All attempts when scanning in SCTP INIT mode got either no response or an protocol unreachable ICMP message, again, indicating that the ports were closed. All but three ports received port unreachable ICMP message during the UDP scan. The ports `5353`, `32761`, and `54321` did not respond to the probe at all, which, in the UDP context, is an indication that some services may be running on those ports.[49]

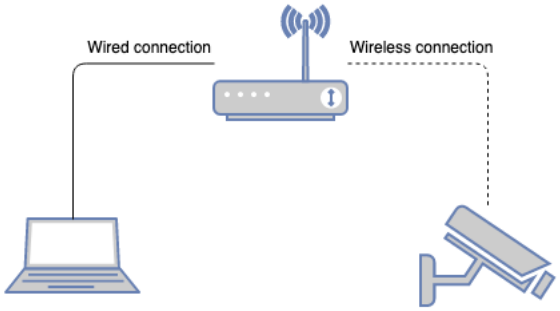
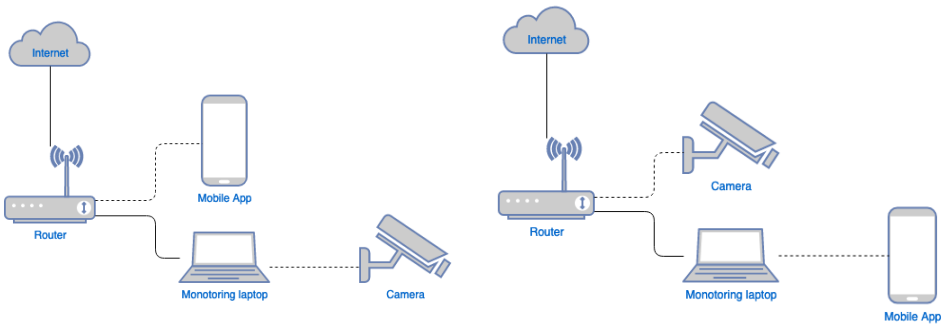


Figure 6.2: Network setup for port scanning.

6.3 Monitoring network traffic

Network traffic capturing and dumping were performed to get an understanding of whom the camera and mobile application spoke to during common operations and what were said. The traffic were monitored in two network setups (which can be seen in Figure 6.3). In both setups, two scenarios were tested, namely 1) starting the monitored subject, and 2) connecting to the cameras video stream from the mobile app. The subject under test in Figure 6.3a were the camera, and starting it means the power on and boot sequence. The boot sequence were determined to have finished when the network reached a steady repeating pattern of traffic. The mobile application were the subject in Figure 6.3b. Starting it involves just opening it from a fully closed state. Again, traffic were monitored until the traffic reached a steady repeating pattern.

A summary and highlights of details can be seen in Appendix A. Analysing the traffic going to and from the camera (i.e. as captured when monitoring the camera) reveal that most of the the majority of the traffic were held in an unidentified protocol over UDP. There seems to be a lot of redundancy



(a) (b)

Figure 6.3: Network setup for capturing and dumping network traffic to and from camera and mobile app.
(Solid lines are wired connections. Dotted are wireless.)

in the messaging structure, e.g. sending the same message to multiple hosts. This might be to combat the fact that UDP does not guarantee the delivery of a message. Analysis of the traffic going to and from the mobile application uncover a different story. Here most of the traffic is TCP based instead. The protocol in some of these conversation can be identified as HTTP, others are likely HTTPS (based on port number and the use TLS).

6.4 FCC filings

The camera is listed in the FCC filings database under with the ID `2APA9-IPC009A`. The filings contains 1) the product manual, 2) letters pertinent to the bureaucratic process, 3) external & internal photos of the camera, and 4) documentation of the Radio Frequency (RF) test setup & results. Not available to view are the products 1) block diagram, 2) schematics, and 3) operational description. One of the letters is a request to keep those documents confidential, which probably was granted. Another letter mentions that the Micro-USB port on the camera is "…not capable of transferring data but only used for charging the internal battery." It is unclear what battery the letter refers to; they probably meant to say that the port is capable of powering the camera.

Chapter 7

Threat model

The threat model for the system under test will be constructed step by step in this section. At the end of this section a list of potential threats will have been identified and rated. To accomplish this the methodology discussed in section 3.3 Threat modelling will be employed.

7.1 Asset identification

Using the information learned in chapter 6 Information gathering and chapter 5 Related vulnerabilities assets pertinent to the system under consideration can be elicited. The identified assets can be found in Table 7.1. Each assets is described and assigned an id.

Asset-ID	Description
01	Video & audio stream
03	Video & audio stored on Micro-SD card in camera
04	Firmware
05	Mobile applications (iOS and Android)
06	Rate of data transmission

Table 7.1: Identified assets of Xiaomi Mi Home Security Camera 360° 1080P.

7.2 Architecture overview

As the methodology prescribes, a set of use cases for the camera were created. The use cases where built around combinations of an actor doing an action

at/from a place. These factors can be found in Table 7.2. These factors can be combined into use cases, e.g.

> Camera owner access the real time stream on mobile device from the same network as the camera.

Some of theses combination may however be infeasible or otherwise nonsensical, and are thus discarded.

Actor	**Action**	**Place**
Camera owner	Access real time stream	Same network as camera
User with shared access	Access recorded stream	Different network than camera
	Receive notification about detected motion	

Table 7.2: Actors, actions and place factors in use cases.
(Columns are independent!)

Table 7.3 contain the resulting combinations. They are further detailed in Appendix C Use Cases.

Based upon these, use cases an architectural diagram has been constructed for the entire system. This diagram can be seen below in Figure 7.1.

In the information gathering step some technologies were identified to be used in the system. These are collated in Table 7.4. The details for some key components are however unknown at this point in the pentest.

ID	Description
01	Camera owner accesses real time stream on mobile device from same network as camera.
02	Camera owner accesses real time stream on mobile device from different network than camera.
03	Camera owner accesses recorded stream on mobile device from same network as camera.
04	Camera owner accesses recorded stream on mobile device from different network than camera.
05	User with shared access accesses real time stream on mobile device from same network as camera.
06	User with shared access accesses real time stream on mobile device from different network than camera.
07	User with shared access accesses recorded stream on mobile device from same network as camera.
08	User with shared access accesses recorded stream on mobile device from different network than camera.
09	Camera owner receives motion detected notification on mobile device from same network as camera.
10	Camera owner receives motion detected notification on mobile device from different network than camera.

Table 7.3: Summary of use cases for Xiaomi Mi Home Security Camera 360° 1080P.

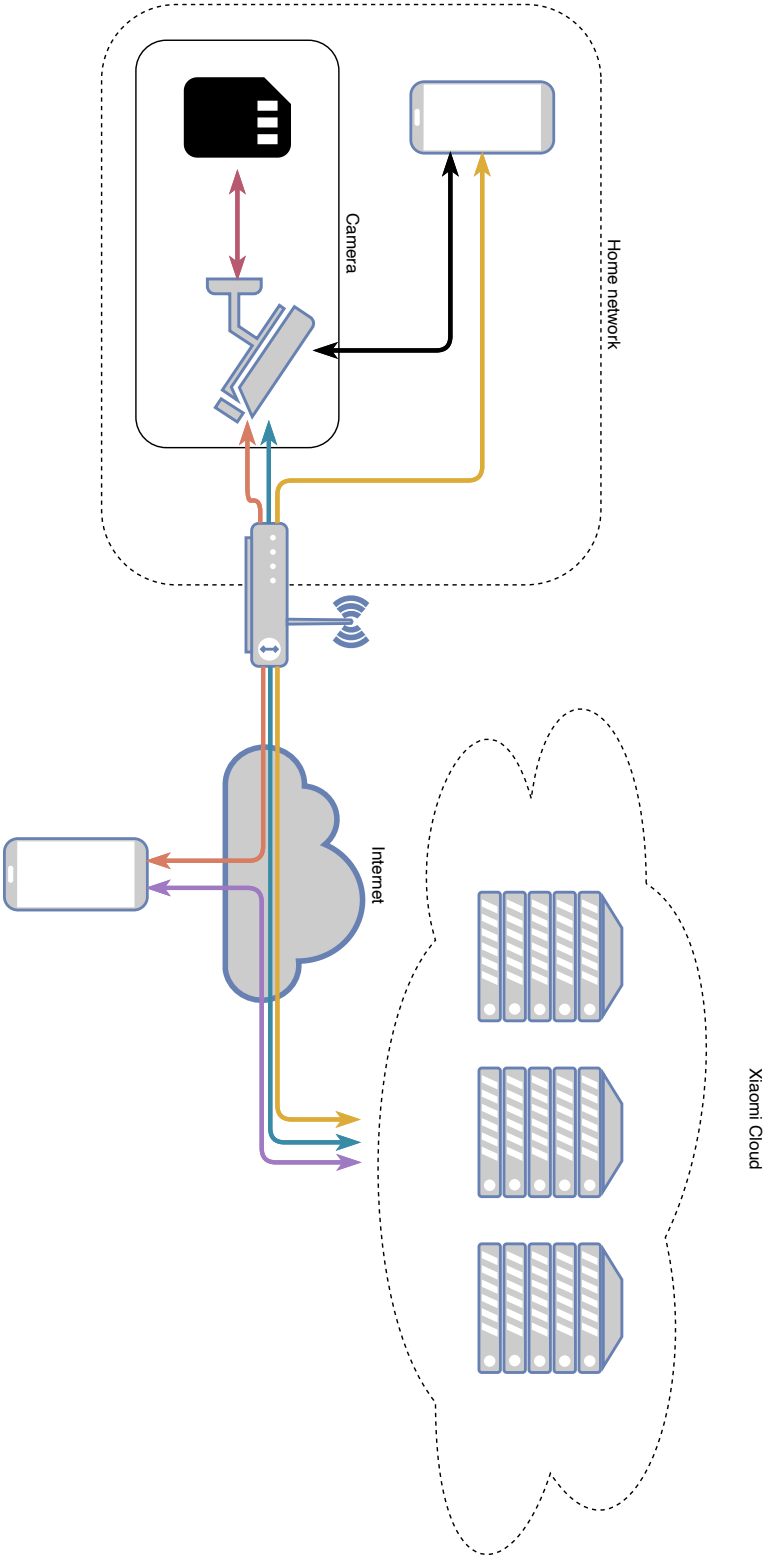


Figure 7.1: Architectural overview of Xiaomi Mi Home Security Camera 360° 1080P.

Technology	Details
Camera	OS unknown. Communicates mostly over UDP/IP. A fewTLS (suspected HTTPS) messages are also exchanged. Also observed to perform DNS queries.
Mobile applications	Android and iOS applications that connect to the Xiaomi cloud and the camera. Communicates with cloud mostly over TLS (suspected HTTPS), but a few conversations were with the unencrypted HTTP. Communicates with the camera over UDP.
Communication protocol: UDP	User Datagram Protocol (UDP) is a fire and forget protocol for sending a datagram over IP. Other protocol may use UDP as a transport layer. Most of the traffic observed in the information gathering were UDP but the inner protocol could not be identified.
Communication protocol: TLS	Transport Layer Security (TLS) is a protocol to encrypt and authenticate traffic over a TCP connection. Is commonly used to secure HTTP traffic. All observed TLS connections terminated at TCP port 443, which again is commonly used for HTTPS traffic. It is therefore likely that most if not all of this traffic in reality is HTTPS traffic. This is the second most common traffic type observed in the information gathering step.
Communication protocol: HTTP	Hypertext Transfer Protocol (HTTP) request/response protocol for exchanging data on the web. Only a few (8) packets identified as this were seen.
Communication protocol: HTTPS	HTTP over TLS (HTTPS).
Communication protocol: DNS	Domain Name System (DNS) is a protocol on top of UDP for resolving data associated with domain names, mostly their IP address.
Communication protocol: mDNS	Multicast DNS (mDNS) is a protocol similar to DNS, but for resolving domain names on the local network.
Communication protocol: XMPP	Extensible Messaging Presence Protocol (XMPP).

Table 7.4: Technologies used by Xiaomi Mi Home Security Camera 360° 1080P.

7.3 Architecture decomposition

This section describes the identified system trust boundaries and system entry points. The system data flow were already identified in chapter 6 Information gathering and displayed in Figure 7.1.

The identified entry points are listed in Table 7.5. The identified trust boundaries of the system can be seen in Figure 7.2.

ID	Entry point	Details
01	Micro-SD card	The SD-card slot can according to [46, 47] be used to update the firmware, which could be exploited by a person with physical access to compromise the camera.
02	Network communications	The camera's network traffic could divulge confidential information, e.g. by weak or non-existent encryption or traffic pattern analysis.
03	Mobile applications	The mobile applications are the only manner through which the user may interface with the camera.

Table 7.5: Technologies used by Xiaomi Mi Home Security Camera 360° 1080P.

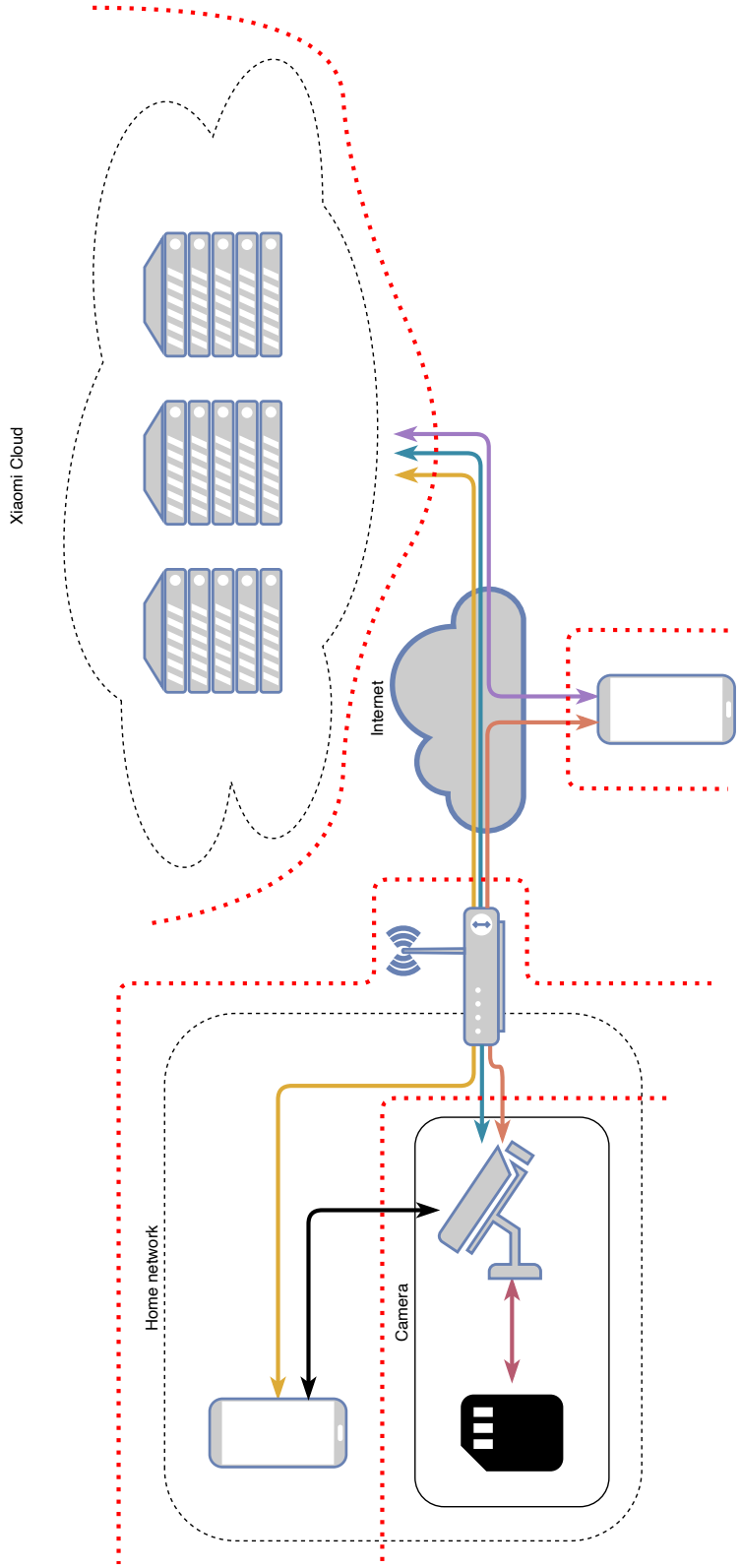


Figure 7.2: Architectural overview of Xiaomi Mi Home Security Camera 360° 1080P with trust boundaries.

7.4 Threat identification

In this section the result of the threat identification will be presented. As discussed in section 3.4 Threat identification STRIDE-per-element is used to identify threats. The focus matrix used can be found in Table 2.1. The threat elicitation is based upon the vulnerabilities identified in chapter 5 Related vulnerabilities, but also expand on them. The vulnerabilities found in the related vulnerabilities chapter are covered, except in cases where information revealed in chapter 6 Information gathering demonstrate that a particular vulnerability do not apply. One example of such a vulnerability is *Unnecessary open ports*, which safely can be disregarded since the system under test does not have any open ports as described in section 6.2 Port scanning.

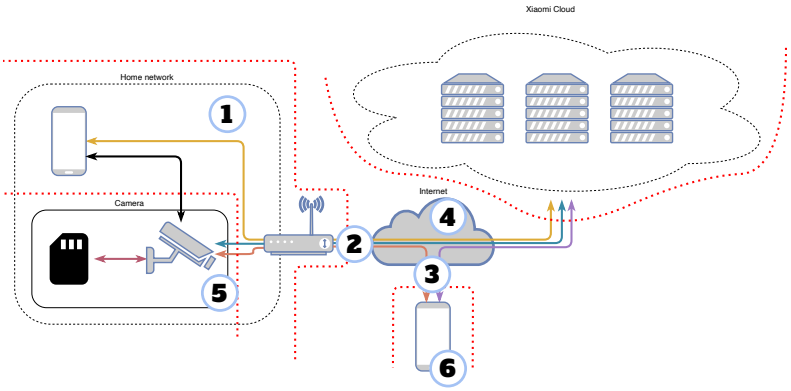


Figure 7.3: Architectural overview with adversary positions.

Marker	Position description
1	On the same WiFi as camera
2	ISP like position.
3	Between phone and general internet. (Not on the same WiFi as camera though.)
4	Anywhere on the internet.
5	Physical access to camera.
6	Physical access to phone.

Table 7.6: Potential adversary positions. Legend for Figure 7.3.

Figure 7.3 contain the architecture overview again, but with labels for different positions an adversary may operate in. These are referenced in the enumeration of the identified threats below.

Final note, some of the identified flows are similar but not identical, e.g. orange and black. STRIDE does demand that threats are identified for each element in the architecture. Thus, similar elements may elicit similar threats.

7.4.1 Data flow

This section present identified threats against data flows.

Threat 1: Video stream replaced

Target	Orange		
	Confidentiality	**Integrity**	**Availability**
Attacks		✓	
Adversary position	2, 3		
Description	An adversary that controls some part of the path between the camera and the phone could replace the feed going to the phone.		
Rel. Work Vuln.	Table 5.5: 4, 8		
Sources	[50]		

Threat 2: Video stream improperly encrypted

Target	Orange		
	Confidentiality	**Integrity**	**Availability**
Attacks	✓		
Adversary position	1, 2		
Description	Improper encryption could allow an adversary with ability to intercept the video traffic to snoop on it.		
Rel. Work Vuln.	Table 5.5: 4		
Sources	[51]		

Threat 3: Traffic amount analysed

Target	Orange		
	Confidentiality	**Integrity**	**Availability**
Attacks	✓		
Adversary position	1, 2		
Description	An adversary with ability to intercept the video traffic could analyse the amount of traffic to infer secrets without (decrypting and) reading the plaintext.		
Rel. Work Vuln.			
Sources	[48]		

Threat 4: Video stream discarded

Target	Orange		
	Confidentiality	**Integrity**	**Availability**
Attacks			✓
Adversary position	2, 3		
Description	An adversary that controls some node that the traffic flows through could drop the video stream.		
Rel. Work Vuln.			
Sources	[50]		

Threat 5: Video stream lost in noise packets

Target	Orange		
	Confidentiality	**Integrity**	**Availability**
Attacks			✓
Adversary position	4		
Description	An adversary that have identified the video stream could attempt to add noise packets to either overwhelm the app processing the stream, or corrupt the result.		
Rel. Work Vuln.	Table 5.5: 3, 8, 15		
Sources	[52, 53]		

Threat 6: Video stream replaced

Target	Black		
	Confidentiality	**Integrity**	**Availability**
Attacks		✓	
Adversary position	1		
Description	An adversary that controls some part of the path between the camera and the phone could replace the feed going to the phone.		
Rel. Work Vuln.	Table 5.5: 4, 8		
Sources	[50]		

Threat 7: Video stream improperly encrypted

Target	Black		
	Confidentiality	**Integrity**	**Availability**
Attacks	✓		
Adversary position	1		
Description	Improper encryption could allow an adversary with ability to intercept the video traffic to snoop on it.		
Rel. Work Vuln.	Table 5.5: 4		
Sources	[51]		

Threat 8: Traffic amount analysed

Target	Black		
	Confidentiality	**Integrity**	**Availability**
Attacks	✓		
Adversary position	1		
Description	An adversary with ability to intercept the video traffic could analyse the amount of traffic to infer secrets without (decrypting and) reading the plaintext.		
Rel. Work Vuln.			
Sources	[48]		

Threat 9: Video stream discarded

Target	Black		
	Confidentiality	**Integrity**	**Availability**
Attacks			✓
Adversary position	1		
Description	An adversary that controls some node that the traffic flows through could drop the video stream.		
Rel. Work Vuln.			
Sources	[50]		

Threat 10: Video stream lost in noise packets

Target	Black		
	Confidentiality	**Integrity**	**Availability**
Attacks			✓
Adversary position	1		
Description	An adversary that knows about the path the video stream takes across the internet could try target some critical node or edge with a flood of packets.		
Rel. Work Vuln.	Table 5.5: 15		
Sources	[52, 53]		

Threat 11: Messages from cloud replaced

Target	Purple		
	Confidentiality	**Integrity**	**Availability**
Attacks		✓	
Adversary position	3		
Description	An adversary that controls some part of the path between the cloud and the phone could replace the messages going to the phone.		
Rel. Work Vuln.	Table 5.5: 4, 8		
Sources	[50]		

Threat 12: Communications with cloud improperly encrypted

Target	Purple		
	Confidentiality	**Integrity**	**Availability**
Attacks	✓		
Adversary position	3		
Description	Improper encryption could allow an adversary with ability to intercept the messages between the cloud and the phone to snoop on them.		
Rel. Work Vuln.	Table 5.5: 4		
Sources	[51]		

Threat 13: Amount of traffic with cloud analysed

Target	Purple
	Confidentiality **Integrity** **Availability**
Attacks	✓
Adversary position	3
Description	An adversary with ability to intercept the messages between the phone and cloud could analyse the amount of traffic to infer secrets without (decrypting and) reading the plaintext.
Rel. Work Vuln.	
Sources	[48]

Threat 14: Traffic with cloud discarded

Target	Purple
	Confidentiality **Integrity** **Availability**
Attacks	✓
Adversary position	3
Description	An adversary that controls some node that the traffic flows through could drop the messages.
Rel. Work Vuln.	
Sources	[50]

Threat 15: Messages from cloud replaced

Target	Yellow
	Confidentiality **Integrity** **Availability**
Attacks	✓
Adversary position	1, 2
Description	An adversary that controls some part of the path between the cloud and the phone could replace the messages going to the phone.
Rel. Work Vuln.	Table 5.5: 4, 8
Sources	[50]

Threat 16: Communications with cloud improperly encrypted

Target	Yellow		
	Confidentiality	**Integrity**	**Availability**
Attacks	✓		
Adversary position	1, 2		
Description	Improper encryption could allow an adversary with ability to intercept the messages between the cloud and the phone to snoop on them.		
Rel. Work Vuln.	Table 5.5: 4		
Sources	[51]		

Threat 17: Amount of traffic with cloud analysed

Target	Yellow		
	Confidentiality	**Integrity**	**Availability**
Attacks	✓		
Adversary position	1, 2		
Description	An adversary with ability to intercept the messages between the phone and cloud could analyse the amount of traffic to infer secrets without (decrypting and) reading the plaintext.		
Rel. Work Vuln.			
Sources	[48]		

Threat 18: Traffic with cloud discarded

Target	Yellow		
	Confidentiality	**Integrity**	**Availability**
Attacks			✓
Adversary position	1, 2		
Description	An adversary that controls some node that the traffic flows through could drop the messages.		
Rel. Work Vuln.			
Sources	[50]		

Threat 19: Messages from cloud replaced

Target	Blue		
	Confidentiality	**Integrity**	**Availability**
Attacks		✓	
Adversary position	1, 2		
Description	An adversary that controls some part of the path between the cloud and the camera could replace the messages going to the phone.		
Rel. Work Vuln.	Table 5.5: 4, 8		
Sources	[50]		

Threat 20: Communications with cloud improperly encrypted

Target	Blue		
	Confidentiality	**Integrity**	**Availability**
Attacks	✓		
Adversary position	1, 2		
Description	Improper encryption could allow an adversary with ability to intercept the messages between the cloud and the camera to snoop on them.		
Rel. Work Vuln.	Table 5.5: 4		
Sources	[50]		

Threat 21: Amount of traffic with cloud analysed

Target	Blue		
	Confidentiality	**Integrity**	**Availability**
Attacks	✓		
Adversary position	1, 2		
Description	An adversary with ability to intercept the messages between the camera and cloud could analyse the amount of traffic to infer secrets without (decrypting and) reading the plaintext.		
Rel. Work Vuln.			
Sources	[48]		

Threat 22: Traffic with cloud discarded

Target	Blue		
	Confidentiality	**Integrity**	**Availability**
Attacks			✓
Adversary position	1, 2		
Description	An adversary that controls some node that the traffic flows through could drop the messages.		
Rel. Work Vuln.			
Sources	[50]		

7.4.2 Data store

This section present identified threats against data stores.

Threat 23: Modified camera firmware installed

Target	SD-card		
	Confidentiality	**Integrity**	**Availability**
Attacks		✓	
Adversary position	5		
Description	An adversary left alone with the device could power off the device, swap the SD-card to one with a modified firmware, and power it back on to install it.		
Rel. Work Vuln.	Table 5.5: 1, 7		
Sources	[46, 47]		

Threat 24: SD-card stolen

Target	SD-card		
	Confidentiality	**Integrity**	**Availability**
Attacks	✓		
Adversary position	5		
Description	An adversary with unsupervised access to the camera could eject and steal the SD-card. They could also swap it with another one if they'd like.		
Rel. Work Vuln.	Table 5.5: 1, 4		
Sources			

Threat 25: Saved stream deleted

Target	SD-card		
	Confidentiality	**Integrity**	**Availability**
Attacks			✓
Adversary position	5		
Description	With unsupervised access and sufficient time, an adversary could eject the SD-card, modify the contents, e.g. to remove parts, and the reinsert it.		
Rel. Work Vuln.	Table 5.5: 1		
Sources			

7.4.3 Process

This section present identified threats against processes.

Threat 26: Cloud spoofed

Target	Cloud		
	Confidentiality	**Integrity**	**Availability**
Attacks	✓		
Adversary position	4		
Description	An adversary that could spoof the cloud could fool a phone or camera to send it messages instead of to the proper cloud.		
Rel. Work Vuln.	Table 5.5: 3, 4		
Sources	[54]		

Threat 27: Camera spoofed

Target	Camera		
	Confidentiality	**Integrity**	**Availability**
Attacks			✓
Adversary position	1,2,4		
Description	An adversary that can spoof the camera could send faked status messages and/or video stream. Exactly how the camera could be spoofed is unclear, but would likely involve exploiting how the camera announces its presence.		
Rel. Work Vuln.	Table 5.5: 3, 4, 10		
Sources	[54]		

Threat 28: Camera covered

Target	Camera		
	Confidentiality	**Integrity**	**Availability**
Attacks			✓
Adversary position	5		
Description	An adversary puts a blanket over the camera.		
Rel. Work Vuln.	Table 5.5: 1		
Sources			

Threat 29: Camera power unplugged

Target	Camera		
	Confidentiality	**Integrity**	**Availability**
Attacks			✓
Adversary position	5		
Description	With access to the camera, an adversary could unplug the camera's power cord.		
Rel. Work Vuln.	Table 5.5: 1		
Sources			

Threat 30: Camera flooded with packets

Target	Camera		
	Confidentiality	**Integrity**	**Availability**
Attacks			✓
Adversary position	1		
Description	If an adversary could generate enough traffic to the camera, it might be overwhelmed.		
Rel. Work Vuln.	Table 5.5: 15		
Sources	[52, 53]		

Threat 31: Phone spoofed

Target	Phone
	Confidentiality **Integrity** **Availability**
Attacks	✓
Adversary position	1, 4
Description	If an adversary could trick the cloud to believe their phone to be another users, they could either receive status updates, or video stream not destined for them. The adversary could also send spoofed control messages to the camera.
Rel. Work Vuln.	Table 5.5: 3, 4, 10
Sources	[54]

Threat 32: Malicious app read app private storage

Target	Phone
	Confidentiality **Integrity** **Availability**
Attacks	✓ ✓
Adversary position	4, 6
Description	If an adversary could install a malicious app they could try to sniff intents, or abuse some application's privilege to read private information belonging to the camera application.
Rel. Work Vuln.	Table 5.5: 4, 6, 8, 11
Sources	[55, 56, 57]

Threat 33: Video stream access denied

Target	Phone
	Confidentiality **Integrity** **Availability**
Attacks	
Adversary position	6
Description	A user or adversary access the video stream or controls the camera when they are not supposed to, but lack of logs mean that they could deny the access.
Rel. Work Vuln.	Table 5.5: 9
Sources	

Threat 34: Malicious app extracts info from IPC error messages

Target	Phone		
	Confidentiality	**Integrity**	**Availability**
Attacks	✓		
Adversary position	4, 6		
Description	Mobile OSes contain provisions for applications to communicate with or use features from each other. A malicious application without access could try to extract information from the camera's companion application through IPC errors.		
Rel. Work Vuln.	Table 5.5: 3, 8		
Sources	[55, 56, 57]		

Threat 35: Phone flooded with packets

Target	Phone		
	Confidentiality	**Integrity**	**Availability**
Attacks			✓
Adversary position	1, 4		
Description	An adversary flood the phone with packets to overwhelm it and make it unable ot receive legitimate packets.		
Rel. Work Vuln.	Table 5.5: 15		
Sources	[52, 53]		

Threat 36: Malicious app abused companion app

Target	Phone		
	Confidentiality	**Integrity**	**Availability**
Attacks	✓	✓	
Adversary position	4, 6		
Description	A malicious app abuse IPC features provided by the camera's companion application to fool it to the malicious applications biding.		
Rel. Work Vuln.	Table 5.5: 3, 8		
Sources	[55, 56, 57]		

7.4.4 Notes

The identified threats cover most of the threats identified in chapter 5 Related vulnerabilities. Five threats have not been covered. These are:

- Insufficient energy harvesting
- Unnecessary open ports
- Insufficient Privacy Protection
- Lack of Device Management
- Insecure (Default) Settings

As mentioned in the introduction to this section, *Unnecessary open ports* were disregarded due to device not having any open ports. *Insecure (Default) Settings* were disregarded on similar grounds. This threat refers that devices have been observed to be shipped with weak default passwords or other insecure settings. As it has been discovered in chapter 6 Information gathering that the camera isn't accessed for setup via a password protected interface but rather by linking the camera to ones account via a generated QR code, and thus the camera side steps having an insecure default password. Further, as there are no exposed services running on the camera, there are no insecurely configured services exposed either. *Insufficient energy harvesting* were disregarded because the device operates connected to mains, and thus is not power constrained. In section 1.1 Delimitations it were mentioned that the objective of the thesis is to explore the security of the camera, and not how well it handles privacy. *Insufficient Privacy Protection* were therefore disregarded too due to being out of scope. The threat of *Lack of Device Management* on the other hand were considered, but did not elicit a threat. Device management is of great importance in larger deployments, such as industrial applications. There it is important to be able to monitor large swaths of devices, have routines around secure decommissioning, and robust update management. But the camera in question is marketed for a domestic settings, which have less strict requirements. The main requirement is that the device should be easy to update, which the camera and accompanied application met by prompting the user to install updates or installing them automatically.

7.5 Threat rating

As discussed in section 3.5 Threat rating, DREAD were chosen for it's simplicity. There it was also noted that the employed rating system is required to be able to order the found potential threats. Furthermore, while the exact scale used may be arbitrary (as the value of the final score does not matter), the scale for the different terms in DREAD ought to be balanced with each

other to prevent that the score of one term is the sole decider of the final score. Therefore, this section starts by briefly detailing the scales used (Table 7.7), before moving on to reporting the scores and rankings of the threats (Table 7.8).

DREAD term	**Score**	**Description**
Damage potential	0	Nothing.
	2	Authorised users denied control of camera position.
	3	Integrity of control of camera position compromised.
	6	Authorised users denied access to camera stream.
	8	Confidentiality of camera stream breached.
	9	Full access to view camera stream and control camera position,.
	10	Complete system compromise, e.g. ”ownership” transferred.
Reproducibility	0	Extremely unlikely to succeed (even for users already with high privilege in the system).
	5	Vulnerability can be exploited ca. half the time.
	10	Guaranteed to succeed.
Exploitability	1	Unlikely to succeed, even with intimate knowledge and custom tooling.
	4	Specialised knowledge and/or tools required, available only to authenticated users.
	6	Specialised knowledge and/or tools required, available to non-authenticated users.
	8	Specialised knowledge and/or tools *not* required, available only to authenticated users.
	10	Specialised knowledge or tools *not* required, available to non-authenticated users.
Affected users	0	None.
	5	Dependent on deployment details.
	10	Every deployment susceptible.
Discoverability	10	Always assumed to be 10.

Table 7.7: Scales used in ranking threats.

Rank	ID	Threat title	D	R	E	A	D	Avg.
1	23	Modified camera firmware installed	10	8	7	8	10	8,6
2	2	Video stream improperly encrypted	8	8	6	10	10	8,4
3	7	Video stream improperly encrypted	8	8	6	10	10	8,4
4	29	Camera power unplugged	6	10	10	6	10	8,4
5	5	Video stream lost in noise packets	6	8	9	8	10	8,2
6	10	Video stream lost in noise packets	6	8	9	8	10	8,2
7	24	SD-card stolen	5	8	10	8	10	8,2
8	35	Phone flooded with packets	6	8	9	7	10	8
9	3	Traffic amount analysed	7	8	6	8	10	7,8
10	8	Traffic amount analysed	7	8	6	8	10	7,8
11	12	Communications with cloud improperly encrypted	5	8	6	10	10	7,8
12	16	Communications with cloud improperly encrypted	5	8	6	10	10	7,8
13	20	Communications with cloud improperly encrypted	5	8	6	10	10	7,8
14	21	Amount of traffic with cloud analysed	6	8	6	8	10	7,6
15	28	Camera covered	6	10	10	2	10	7,6
16	13	Amount of traffic with cloud analysed	5	8	6	8	10	7,4
17	17	Amount of traffic with cloud analysed	5	8	6	8	10	7,4
18	25	Saved stream deleted	5	8	6	8	10	7,4
19	30	Camera flooded with packets	6	6	7	7	10	7,2
20	33	Video stream access denied	1	6	8	10	10	7
21	6	Video stream replaced	7	6	6	5	10	6,8
22	32	Malicious app read app private storage	9	4	5	5	10	6,6
23	27	Camera spoofed	6	5	5	6	10	6,4
24	31	Phone spoofed	4	5	7	6	10	6,4
25	1	Video stream replaced	7	3	6	5	10	6,2
26	9	Video stream discarded	6	4	6	5	10	6,2
27	14	Traffic with cloud discarded	4	3	9	5	10	6,2
28	18	Traffic with cloud discarded	4	3	9	5	10	6,2
29	22	Traffic with cloud discarded	4	3	9	5	10	6,2
30	34	Malicious app extracts info from IPC error messages	2	6	6	7	10	6,2
31	4	Video stream discarded	6	3	6	5	10	6
32	36	Malicious app abused companion app	3	5	5	7	10	6
33	11	Messages from cloud replaced	4	3	7	5	10	5,8
34	15	Messages from cloud replaced	4	3	7	5	10	5,8
35	19	Messages from cloud replaced	4	3	7	5	10	5,8
36	26	Cloud spoofed	4	5	3	6	10	5,6

Table 7.8: Threat ranking for threats sorted by averaged score in descending order.

Chapter 8

Penetration test

8.1 Threat 23: Modified camera firmware installed

8.1.1 Introduction

This threat is inspired by the work done by T. Marques and M. Cabral on alternative firmware for the camera under consideration. [46] This work was discussed in section 5.1. According to the instructions in their git repositories, it should be possible to modify the firmware of the camera under consideration by having a specifically prepared SD card inserted in the camera when it boots. If this is true, an adversary with physical access to a camera could conceivably, with a few moments unsupervised, quickly unplug the camera, exchange the SD card, plug it back in, move away and wait for the modified firmware to be installed. An adversary could gain physical access in several manners. If the camera is inside a home, then a dining guest or repairman might have enough of an opportunity to perform the manoeuvre. Having the is camera placed outside the cover of dark might be enough an opportunity for an adversary.

The instructions in Marques and Cabral's repositories does however mention that the camera is required to run a specific firmware version for their modified firmwares to be able to apply. They do however mention the possibility of downgrading the firmware by the same method as mentioned above. An adversary's method would thus only need to be slightly modified. Instead of only swapping the SD card once in a unsupervised moment, they would need to do so twice. Once to ensure the correct firmware runs on the camera and then once to install their modified firmware.

8.1.2 Vulnerability analysis method

To test the viability of the approach laid out by Marques and Cabral, we attempt to follow their instructions. More specifically, we follow the instructions by Cabral which are for the specific model of camera that is under consideration, i.e. `MJSXJ05CM`. [46]

Cabral's instructions for downgrading the firmware are as following:

1. Grab the recovery file.*
2. Put the file in the root of an SD card.
3. Power down the camera and insert the SD card.
4. Power on the camera and wait. The status indicator light will be a solid yellow while the firmware is flashing. (Might take several minutes.)
5. When the camera asks for a QR code, the downgrade is complete.

To verify that this step worked, we check the cameras firmware version. The firmware should, according to the instructions, be `3.4.2_0062`. His instructions for installing his modified firmware are:

1. Ensure the camera is configured with the official companion application.
2. Grab the latest modified firmware.*
3. Copy the contents of the *sdcard* folder in the modfied firmware to the root of an SD card.
4. Power down the camera and insert the SD card.
5. Power on the camera.
6. Find the IP address of your camera.
7. Open the web configuration interface for the modified firmware.*

* See [46] for details on how to do this.

8.1.3 Vulnerability analysis results

The firmware of the camera before the experiment is `3.5.1_0052`. As the camera is not on the correct firmware we perform the downgrade step. It however failed. After having performed it, the firmware version has not changed. The instructions mentioned that the procedure might not work with some SD card. The step is therefore performed again with another card, but alas without any luck.

Despite being on the wrong firmware, the step of installing the modified firmware is attempted, but again without success.

8.1.4 Discussion

The fact that the method outlined by Marques and Cabral failed in this case might have several explanations. As noted, their instruction do warn that the procedure might not be compatible with all SD cards. It is possible that both used in here were incompatible and that another might have worked. It could also be that the firmware running on the camera have countermeasures to defend against the downgrade step that the instructions does not consider.

While the method failed on the camera under considerations, a determined adversary might still be able to create an SD card that could modify the cameras firmware, if present during boot. The adversary could purchase a camera themselves and SD cards from multiple manufactures and perform the downgrade step until they manage to create a compatible one. But even if they managed to get a camera and SD card combination to work it is not guaranteed to work on their targets camera, if variables such as camera firmware is a factor, since that is a variable under control of the target.

8.2 Threat 2 & 7: Video stream improperly encrypted

If the video stream is improperly encrypted, then an adversary that can eavesdrop on the communication might be able to decrypt the stream and break it's confidentiality.

A exploratory network capture of when the camera was streaming video reveals many large (ca. 1000 bytes) UDP datagrams being sent to the phone. Given that this stream transmits the most data (by two orders of magnitudes) in the captured window and that video takes a lot of data, this stream is presumed to be the video stream. Figure 8.1 contain an visualisation of the bytes sent

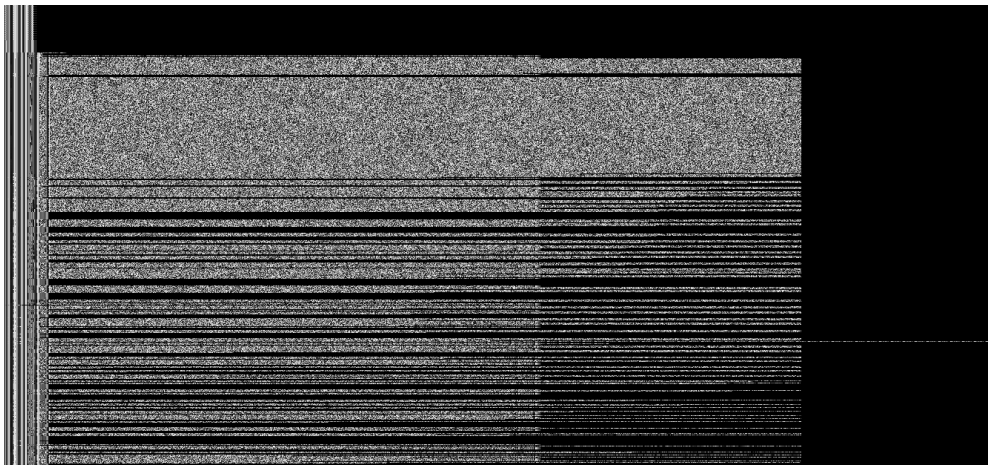


Figure 8.1: Visualisation of the bytes in the payload of the captured video stream. Each horizontal line is one UDP payload. In each line, the first pixel represents the first pixel in the payload, the second pixel the second byte, and so on. The lines are ordered in time order, i.e. the first line is the first packet sent, and the last line is the last packet sent. The packets displayed are all UDP packets going from the camera to the phone.

from the camera to the phone. On the left side of the figure (at the start of the packets) there seems to be some structure consistent across packets. This could be a header. The protocol used in the stream could however not be determined by Wireshark.

8.2.1 Background

This section lays out some background about the topics relevant for examining this threat.

Protocol identification

Tools like Wireshark can recognize some common encrypted protocols, such as TLS. For these common protocols there exist documented exploits against common vulnerabilities in the literature. G. Weidman for example describe how the tool Ettercap can be used to perform a TLS Man in the Middle (MitM) attack. [28] However, the task is non trivial if some data or message is encoded with an unknown protocol. It can be possible to reverse engineer an unknown protocol, but it is a daunting task, especially if it is encrypted. [58]

Information entropy

There exist some techniques that can indicate that it's unlikely that a message is encrypted. Shannon entropy from information theory [59] can be used as a measure for this purpose. Shannon entropy is defined as:

$$H(p_1, p_2, \ldots, p_n) = \sum_{i=1}^{n} p_i log(p_i) \tag{8.1}$$

Encryption is a concept in data processing that produce message with high entropy. It is not the only process though, compression is another. [60] That is:

$$M \text{ is encrypted} \implies H(M) \text{ is high} \tag{8.2}$$

The contrapositive, i.e.:

$$H(M) \text{ is low} \implies M \text{ is unencrypted} \tag{8.3}$$

then gives a simple test for if a message likely is unencrypted.

8.2.2 Vulnerability analysis method

As noted in the introduction, Wireshark could not identify the protocol used to stream. If the protocol used could be identified, then analysis tailored to it could be performed. There are some protocols commonly used for streaming video over UDP: [61]

- Secure Reliable Transport (SRT)
- Web Real-Time Communications (WebRTC)
- Real Time Streaming Protocol (RTSP)
- Real-Time Transport Protocol (RTP)

Even if Wireshark may not be able to automatically be able to detect the correct protocol, it is possible to explicitly ask it to decode the stream as a certain protocol. Deep packet inspection (DPI) tools like Netify are designed to be able to identify encrypted as well as unencrypted network communications. [62] It can be used to analyse network traffic from a capture file in pcap format. Appendix D demonstrates how this can be done.

However, if the protocol employed cannot be discerned, then it would be too time consuming to reverse engineer it to fit in this thesis. The entropy test

could still be employed though. Shannon entropy is defined as sum over the probabilities of a set of possible symbols in a message. To approximate the entropy for a given message it's bytes could be thought of the symbols. The probabilities that any given byte value occurring in the message is unknown but can be approximated by calculating the frequencies at which the values occur in the actual message. [63] The algorithm is outlined in the following pseudocode:

Algorithm 8.2.1: SHANNONENTROPYESTIMATION($bytes$)

local $counts, entropy$
$counts \leftarrow [0; 256]$
for each $byte \in bytes$
  do $counts[byte] \leftarrow counts[byte] + 1$
$entropy \leftarrow 0$
for each $count \in counts$
  do if $count > 0$
  then $\begin{cases} p \leftarrow count/length(bytes) \\ entropy \leftarrow entropy + p * log_2(p) \end{cases}$
return $(entropy)$

Using this method we would expect to see a maximum entropy of 8 if every byte value were equiprobable. An entropy of 0 would mean that the stream would consist of a single byte value.

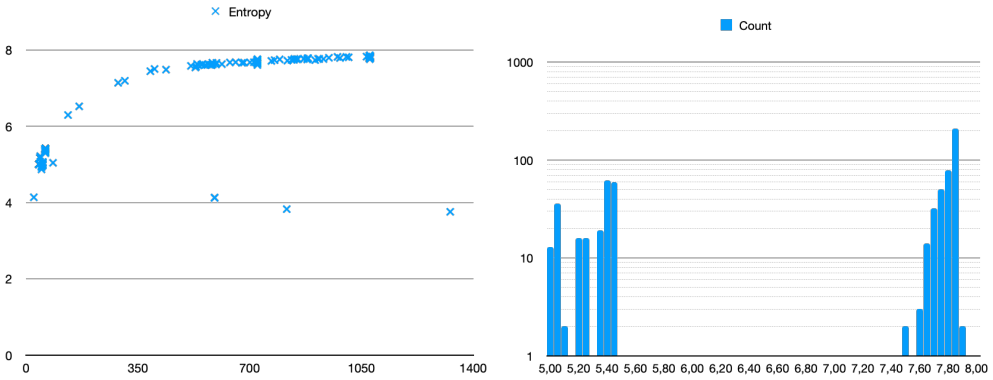
It is worth noting that the accuracy for this algorithm is limited for short messages, as it would only be a small sample of the underlying true distribution of frequencies. Say that we have a message of ten unique byte values generated from a uniform distribution of all byte values. Just looking at the message we would estimate the probability for the containing byte values at $\frac{1}{10}$ instead of the $\frac{1}{256}$ true value. The unseen byte values would conversely be estimated to have a probability of 0. This skew would result in a lower estimated entropy of ca. $3,3$ instead of 8. We would therefore expect that this measure would skew low for short messages. As the messages become longer, the law of large numbers tell us that the observed frequencies would become a more accurate estimation. Fortunately are most of the packets of the captured video stream long (ca. 1000 bytes).

8.2.3 Vulnerability analysis results

Protocol identification

Unfortunately, the protocol used in the video stream could not be identified. Attempting to explicitly ask Wireshark to decode the stream as the above listed protocols resulted at best in incoherent streams. SRT worked the best. Each individual packet on its own superficially made sense. But fields like *sequence number* did not change between packets while the field *destination socket ID* would change. The attempts to decode the stream as the other protocols produced mostly nothing; in the case of RTP a protocol version identifier of 1 was produced but nothing else. The tool Netify did not manage to identify the protocol either, only labelling it as unknown.

Figure 8.2 displays the estimated entropy of the payloads of the UDP packets in the stream. Figure 8.2b demonstrates that the estimated entropy is distributed bimodally for the captured video stream. Most (ca. $\frac{2}{3}$) payloads do have an entropy above to 7.4. Looking at Figure 8.2a we can see that the estimated is positively correlated with payload length, as would be expected from this measure if the underlying distribution has high entropy. There are however a few outliers denoting messages that are relative long but have low entropy.



(a) Scatter plot of estimated entropy compared to payload length.

(b) Distribution of estimated entropy. Log y-axis.

Figure 8.2: Estimated entropy of video stream.

8.2.4 Discussion

While Figure 8.1 indicates that the packets in the stream contains a header, the protocol used in the video stream could not be identified. The captured video

stream also exhibited estimated high entropy, meaning we cannot exclude that the stream is encrypted. Of course, there might be other explanations for the high entropy, such as the stream being compressed (which would be likely that it is given it being video).

No breaches of confidentiality for stream have been found. That does not mean that they do not exist. Further reverse engineering of the protocol is likely required to determine if such exist, which sadly is out of scope for this thesis. An issue with further reverse engineering is however that such practices may be illegal. This issue discussed further in subsection 9.3.2 Ethics.

8.3 Threat 5 & 10: Video stream lost in noise packets

8.3.1 Introduction

This/these threats are concerned with the possibility of an UDP garbage flood attack, where an attacker floods an host with UDP packets with garbage payloads. The intention is to saturate some resource (network or processing power) utilisation to deny legitimate traffic. [53] In this context the attack manifest itself as either consuming the network resources for the mobile phone running the companion application or consuming its processing resources.

8.3.2 Background

This section lays out some background about the topics relevant for examining this threat.

Networking layers and receiving packets

Networking is commonly logically divided into four layers: [64]

1. Link (Ethernet, Wi-Fi, etc.)
2. Internet (IP)
3. Transport (TCP, UDP)
4. Application (HTTP, NTP, etc.)

The general idea is that the lower layers solve simpler problems than the higher layers. The higher layers can then use the lower layers to solve those harder problems. The link layer is concerned with delivering packets from one host one ”hop”, i.e. hosts that are directly connected via e.g. Ethernet or Wi-Fi. On this level are hosts identified with MAC addresses. The internet layer is then responsible for routing packets multiple ”hops”. Hosts are identified with an IP address on this level. The fundamental function of the transport layer is to handle routing of packages between applications (on different or the same host). Applications are identified with *ports*. The combination of IP and port is called a *socket*, which then identifies a specific application on a specific host. The result is a kind of Russian doll of packets. A link layer packet may contain a internet layer packet which in turn may contain an transport layer packet which itself in turn may contain an application layer packet. [65] The three lowest layers are commonly implemented by the operating system. It then provides an interface for applications to send and receive packets to and from other applications. [66]

In general, when a host receives a packet from the network it verifies that it was destined for itself. If it is, the OS then unwraps the internet layer and checks what port the packet was destined for. If there is an application listening on that port, then it is delivered to that application. [66]

Spoofing packets

When sending data out on the network, it is possible to create packets with false information in some or all fields. [52] This may lead the recipient host to believe that a packet came from somewhere other than it actually did, and therefore send potential responses there. Spoofing UDP packets are often easier compared to spoofing TCP packets, since UDP is a connectionless protocol.

Message authentication

There exist methods that the mobile application could employ to verify the authenticity of the messages it receives. These methods can be used to quickly verify that the messages are sent by the stated sender and/or the integrity of the message. These are methods such as *Message Authentication Code* or *Digital Signature*. [67] The mobile application may employ these techniques to quickly discard messages from an improper source.

Magic numbers

File formats and protocols may contain what is called an *magic number*. They are used to identify the protocol or format. Protocols and formats specify where these numbers should be located in them (usually close to the start) and what their values should be. [68]

8.3.3 Vulnerability analysis method

Since the test case that overwhelm network resources would not exercise any particulars of the system under test, but rather the underlying network, it is not considered. Instead this pentest is focused on attempting to trigger the companion mobile application to waste resources processing malicious data.

To test if the companion mobile application is susceptible to UDP garbage flood we designed a program to generate spoofed packets. Appendix E contains the program source code.

The program is capable of generating and sending spoofed UDP packets to a chosen socket. The spoofing done by the program consists of supplying the sent packet with a chosen source IP address and port. The program generates the data to use as payload in these packets with a Markov chain. This Markov chain is trained with data from captured an actual video stream. Markov chains were favoured for generating the payload data over using pure random data to fill it due to that the chain would be more likely to favour byte combinations that could be beneficial for bypassing some payload verification, e.g. checking for a magic number. The program have been observed to be able to emit ca. 1700 packets per second on the computer used in the test*, were each packet is in the range 600-1400 bytes long. To be able to differentiate the malicious packets compared to the genuine ones (e.g. in a capture file) the TTL field of the IP packet were used. This field was chosen as a differentiator as it should not affect how the packet is handled once received, since TTLs may naturally vary depending on which route a packet took through a network. It was filled with a value that differed from TTLs observed on genuine packets.

In the test, the network were ordered as can be seen in Figure 8.3. A video stream were initiated from the camera. The computer labelled *Adversary* eavesdropped on the traffic. The source and target sockets in the video stream were identified. The above discussed program were then started with these parameters.

The hypothesis here is that, since the spoofed packets should resemble

* MBPr 13” 2015, 2,7GHz Dual-Core i5 with 8GB memory.

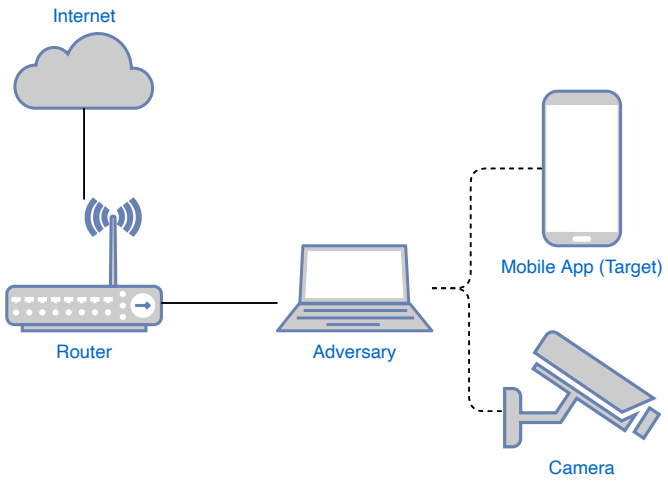


Figure 8.3: Network setup

the authentic packets up to the transport layer, they should be delivered to the mobile application. If the mobile application perform any processing, e.g. decompressing or decrypting, before checking that the packet is genuinely from the camera, then the mobile's processing power should quickly be consumed. If no verification were to be done by the application, and the packets instead were implicitly trusted and attempted to be rendered, then we could either exhibit a corrupt stream or that some expected invariant being broken by the crafted package leading to an application crash.

8.3.4 Vulnerability analysis results

The malicious traffic in the experiment (identified with TTL value) were observed to account for ca. 80% of traffic to the target socket on the mobile, i.e. the UDP garbage flood traffic outnumbered the genuine traffic 4 to 1.

The affect on the playback of the video stream or the application itself were however minor at most. The playback would occasionally freeze for short periods of time (ca. 10 seconds) but would quickly resume again. The rest of the app were during these hiccups of the playback still responsive, and notably did not crash. The mobile phone did not either get perceivably warm as one would have expected if it would have been under significant computational stress.

8.3.5 Discussion

While the supplied garbage data indeed were a flood compared to the genuine traffic, the application did not exhibit any signs that it were vulnerable to this attack. Perhaps the application quickly identifies and discards the malicious data via magic numbers or message authentication. A deeper understanding of the protocol used perhaps could be useful to craft malicious garbage data. However, if message authentication is used and implemented properly, then it probably is close to impossible to spoof messages. In conclusion, the companion mobile application does not seem vulnerable to the UDP garbage flood attack.

8.4 Threat 3, 8 & 21: Traffic amount analysed

8.4.1 Introduction

This threat is inspired by the work of Li et al. [48] As discussed in section 5.2, they have shown that it is possible to deduce sensitive information, by just observing how the camera's traffic amount varies in different scenarios. In this section we will investigate whether the camera under test is susceptible to this attack.

8.4.2 Background

This section lays out some background about the topics relevant for examining this threat.

Side channel attacks

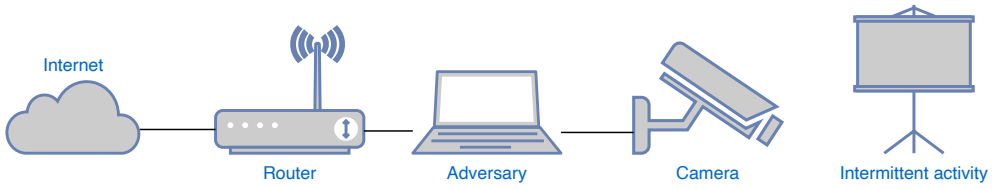
A side channel attack is an exploit that relies on a vulnerability that stems from a misalignment between the security model used to develop the system and the necessity of reality. If the security model omits to account for a possible attack vector, e.g. by relying on an assumption that a host platform is secure, the system may be only secure on paper. The time consumed by an operation, a system's power consumption and emitted electromagnetic radiation have all been used to glean secrets from otherwise secure systems. To achieve a truly secure system these factors must taken into account in both the security model and implementation. Counters to these attacks depend on the specific vector. For example timing attacks can be countered by designing algorithms such that their run time either is fixed, or do not depend on secrets. [69, pt. Side-Channel Attacks]

The vulnerability presented by Li et al. [48] is another type of side channel attack, where the amount of traffic sent by the camera is used to deduce secrets about what it is surveying. They in particular show that they are able to tell when the camera they used detected motion as it then initiated an upload of the captured video to the cloud and thus produced an observable surge in the traffic rate. They further noted that, since the traffic surge were triggered because the camera detected motion, the observed surge could be correlated with specific regular motion (such as the camera owner leaving the house) in the surveyed area.

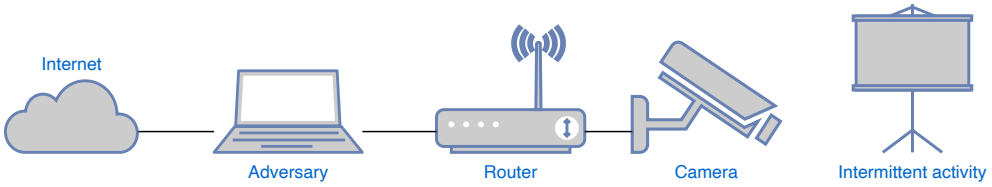
8.4.3 Vulnerability analysis method

To test whether the camera under consideration in this thesis is susceptible to leak when a motion event occurs to an eavesdropper just by the amount of traffic sent by the camera the following experiment were carried out. In this experiment two events will be attempted to be detected, namely: the camera detects motion and a user starts a video stream.

The network were setup as can be seen in Figure 8.4a. In the imagined scenario the adversary could either be in the position inside the network, as in the experiment, or outside the network, as in Figure 8.4b, as long as the traffic originating from the camera could be isolated. Alternative 1 were chosen for the experiment due to the traffic being easier to isolate in that version.



(a) Alternative 1



(b) Alternative 2

Figure 8.4: Network setup

The camera were pointed at an area to be monitored. When attempting to determine when the camera detects motion, a video with activity once every

five minutes were played in a loop as an analogue for an area with intermittent activity. The camera were configured to detect motion and when doing so, send a push notification. In the scenario where detecting the start of a video stream the motion detection of the camera were turned off. A user then starts and end a few video stream over the captured period. In both scenarios the camera starts turned on but in idle mode.

The cameras traffic were recorded with `tshark`, a terminal version of Wireshark. Using a *capture filter* that only traffic originating from the camera were captured. The traffic were saved to a Pcap file. The structure of the command used can be seen in Listing 8.1.

```
tshark -i {INTERFACE_OF_CAMERA} -F pcap \
       -f "src host {CAMERA_IP} -w {FILE}"
```

Listing 8.1: Structure of command to capture camera traffic.

The captured packets were then partitioned into buckets. If there were n buckets, the partition were done so that the packets that occurred for the first $\frac{d}{n}$ seconds of the trace was placed in the first bucket, where d is the duration of the entire trace in seconds. The packets in the second time fraction where placed in the second bucket, and so on. For each bucket, the bytes in the packets in the bucket where summed up to calculate the amount of traffic in each time slice. These rates were then normed, where the slice with the maximum traffic were set as the norm. These normed rates were then plotted against the start time of each slice, expressed as seconds since start of the trace. In this plot, the times the camera reported motion detected events where also added.

8.4.4 Vulnerability analysis results

The resulting plots can be seen in Figure 8.5 and Figure 8.6. In both plots we can see a distinct traffic surge shortly after all events of interest have occurred. In the case where the camera detects motion in its area of surveillance, the traffic surge is abrupt but short lived. The traffic surge related to a video stream is also abrupt but the traffic is sustained.

8.4.5 Discussion

Both types of activity (camera detects motion and video stream initiation) are followed by distinct traffic surges. The two events however differ in how long the higher traffic levels are sustained. The traffic is short lived when the camera detects motion. In comparison, the traffic is sustained when video streaming is

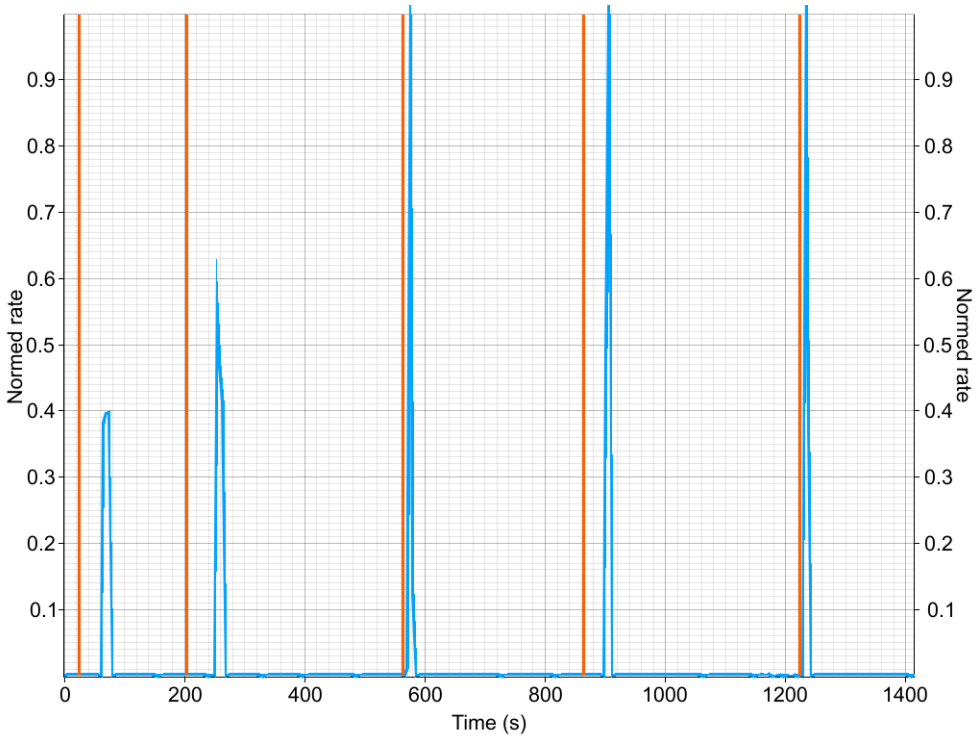


Figure 8.5: Normed traffic rates during a period where the camera detects motion. Traffic rates are drawn in blue. The orange lines are the times the camera reported that it detected motion.

initiated. These patterns could allow an adversary to detect and differentiate these events.

As discussed by Li et al., being able to detect these events might leak sensitive private information. Given enough regular data an adversary might be able to make inferences about what happens in the area surveyed by the camera. [48] For example a motion event that occurs every weekday morning might signify that the owner is leaving their house, while a stream event in the afternoon might indicate that the owner is checking the cameras in their home before leaving work. Such inference however requires that the motion and stream events are correlated with other observations of interest.

It is conceivable that this vulnerability partly could be mitigated. In the traffic rate diagrams we can see that the camera continuously send a small amount of data. If these events could be hidden in this background data it would harder or impossible for an adversary to distinguish them. Hiding the video stream might however be impossible due to the amount of traffic involved. Continuously sending enough background traffic so that a video stream could be

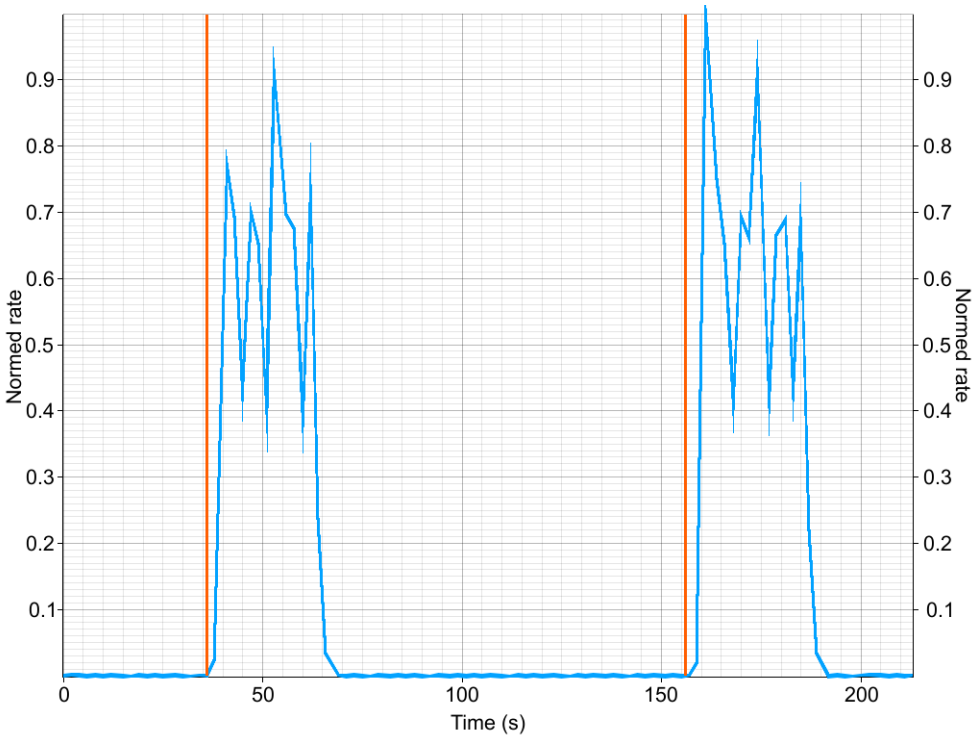


Figure 8.6: Normed traffic rates during a period where video streams are initiated. Traffic rates are drawn in blue. The orange lines are the times the stream were initiated in the companion application.

hidden would likely demand what would be an unjustifiable amount of dedicated bandwidth for a domestic environment. The brief traffic spike associated with an motion detection event perhaps could be hidden though. Admittedly, is is unknown what data is contained in the associated traffic spike, but one could conceive that a minimal amount of data, i.e. just enough to indicate that an event has occurred, when it happened, and what camera detected it could be encoded in 3 integers. This small amount of data ought to hidable in the background traffic. If additional data, e.g. a thumbnail of the event, is desired, perhaps it could be requested by the user client as needed.

Chapter 9

Discussion

This chapter will discuss three topics. First, it will be discussed how well the chosen methodology served to explore the research question: *In what capacity are today's smart home cameras more secure than those in the previous decade?* Then, the results of the individual pentests will be summarised and discussed in context of the research question. Finally, this chapter will discuss this thesis from a ethical and sustainability perspective.

9.1 Reliability and Validity

The methodology (which can be found in chapter 3) chosen in this thesis were a combination of methods, processes and techniques presented in chapter 2 Background. None of the authors or works described there offered a process complete or thorough enough, which presented the challenge of combining the available sources into a suitable methodology. Although an amalgamation, the chosen method served well to answer the research question.

The selected methodology does emphasise the threat modelling step, but not unnecessarily. Time is always a limited resource and there is not time to explore every conceivable threat, but the threat model is a structured process that aims to ensure that the threats that are investigated are the most important. Of course, one could ask if it is really justified spending precious time crafting a thorough threat model, if that time could have been used to investigating more vulnerabilities? The threat model provides two valuable artefacts though. First, possible threats are identified as a part of it. Regardless of the process, threats do need to be identified, lest there would be none to investigate. Secondly, threat modelling produces a threat ranking. The threat model can thus in a structured manner prioritize the identified threats. This prioritization allows the

most severe threats to be considered first. Thus, the threats that were identified but not investigated are those that are deemed to be less severe. Of course, some threats might be miss-prioritized. But at least a structured approach can be scrutinized for systematic errors. Spending time on proper threat modelling therefore can improve the relevancy of each threat actually tested.

To modify the method described by G. Weidman to fit an iterative proved somewhat mute, since only a single vulnerability analysis found a vulnerability. The modification were still justified though. Just because the iterative capability in the end were not used in this thesis, does not imply that the flexibility granted were unwise to include. As argued in chapter 3 Methodology, the ability to adjust priorities as new informations is learned do contribute to the relevancy of every threat tested.

In summary, the selected method has contributed to ensure that the threats investigate are the most relevant and interesting.

9.2 General Observations of Results

Overall, the Xiaomi Mi Home Security Camera 360° tested in this thesis demonstrates an impressive level of security compared to cameras examined in works from just a few years ago. In chapter 5 Related vulnerabilities a few issues were identified to be common in cameras tested between 2015 and 2020, including unnecessarily exposed services, insecure default credentials and improper encryption.

Already in the chapter 6 Information gathering it became apparent that the Xiaomi camera completely sidesteps some of these issues. For example, the port scan performed during this step demonstrated that this camera does not have any services exposed. Having no services exposed is a great way of having no insecure services exposed. Furthermore, non-existent services do not require any credentials, and thus many opportunities for weak default credentials are eliminated. Issues with insecure default credentials in the configuration process are again side stepped, by configuring the camera once through a QR code. Scanning the QR code provides the camera with Wi-Fi credentials as well as links the camera to a user.

The camera is able to side step these issues as it is relying on the cloud. As we have seen, there are benefits of relying on the cloud, but they do come with some drawbacks. We have an actual example. As presented in chapter 5 Related vulnerabilities, in late 2019 there was a confidentiality breach of owners camera streams as an issue in the cloud services resulted in some users being able to see other users video streams. Another potential point of contention is

that the cloud provider may collect usage data, which could be sensitive from a privacy standpoint. The level of privacy provided by the tested solution is however outside the scope of this thesis and has thus not been tested. Another disadvantage is that an outage of the cloud service might have the potential of preventing owners using their cameras as intended. So cloud reliance might beneficial for local security, but has its own trade-offs and is certainly not a panacea.

Beyond having positive results in the information gathering stage, the camera under consideration also held up well in three of the four targeted pentests. The only vulnerability the thesis has been able to demonstrate is that an adversary could detect and differentiate a few events, just by monitoring the amount of traffic transmitted by the camera. As discussed by Li et al., this side channel may leak sensitive information, but does require some time and resources to correlate the events with other observations (e.g. owners leaving for work) to become actionable. On it's own, it could be compared to the information provided by a lit room, in an otherwise dark house, to those on the street outside. So, while certainly not perfect, the camera does hold up well from a local security perspective, especially compared to earlier models.

Of course, this evaluation is based on the tests performed in this thesis. It is then therefore natural to ask if the right tests have been performed. Time is an unfortunate limitation, and not all identified threats could be tested. As discussed in the previous section, the selected methodology have promoted the most relevant threats to be tested. It is however inevitable that some vulnerabilities may hide in the threats that were not tested due to time constraints. Secondly, the constructed threat model may contain mistakes, and thus some threats may either not have been identified or miss-classified. To prevent that important threats has been missed, related work has been explored to guide the threat identification process. Further, to promote that each test achieve the best result, each is based on the findings in associated background research. Of course, the performed tests contain limitations. As an example, the results for Threat 2: Video stream improperly encrypted, Threat 7: Video stream improperly encrypted, Threat 5: Video stream lost in noise packets, and Threat 10: Video stream lost in noise packets could have been more certain if the protocol used had been better understood. But as discussed in 9.3.2, there might be some legal questions needing attention if one were to attempt reverse engineer it. Finally, as discussed in section 1.1, the extent to which the research question can be answered is limited by that only one camera has been evaluated.

9.3 Sustainability & Ethics

This section discuss how the result of this works affect sustainability and the ethics aspect of the work done.

9.3.1 Sustainability

Sustainability is a multifaceted concept that encapsulates environmental, societal and economical aspects.

The environmental aspects of this work is negligible. The camera that has been under consideration is the single thing acquired for this thesis*. Further, this camera will go back to the departments lab so that other experiments could be performed with it in the future, meaning that the resources used to produce and transport it can potentially be amortized over multiple projects.

The small environmental impact is likely outweighed by the economic benefits. That this camera now has been tested and that no serious vulnerabilities have been found does lend some credibility to this specific camera and, in somewhat less degree, the manufacturer Xiaomi. This work can be thus hopefully help future camera owners choose a more secure option.

Societal ramifications of this work are more tenuous. The social pillar of sustainability is concerned with issues like (but not limited to) inequality, poverty, and health. One might argue that more secure cameras could improve surveillance capabilities, which could be used for a 1984-esque argument in which the camera is a tool aiding subjugation of populations and thus promoting inequality. However secure cameras could also prevent actors from eavesdropping on presumed private areas, and thus prevent private cameras being used as Orwellian spy-infrastructure. Consequently, it is possible to argue that secure cameras can both negatively and positively affect social development.

9.3.2 Ethics

The act of pentesting and hacking is always an exercise in ethics as information about potential vulnerabilities could valuable. Vulnerabilities can be disclosed in different manners. *Full disclosure* is a method were the vulnerability is made public to it's full extent. A product's users, developer and attackers can all learn about the vulnerability at the same time. A party that found a vulnerability may also choose to keep it to themselves or sell it. Such behaviour is classified as *non disclosure*. If a discoverer like to give the developer of a product the

* Of course, beside the copious amount of snacks that have fuelled the author.

opportunity to fix the vulnerability before it is published, then they can perform a *Coordinated Vulnerability Disclosure (CVD)* together with the developer. The Dutch National Cyber Security Centre recommend that vulnerabilities are disclosed with CVD.[70]

The findings of this thesis is however of such nature that they do not prompt the need for CVD. The majority of the exploration and testing of the camera under consideration have not been able to demonstrate vulnerabilities. That the camera is susceptible to leak a small amount of information through the amount of network traffic it generates is the exception. While this vulnerability likely is widespread, it is deemed that the damage potential is low.

Beyond the concerns about how to disclose a vulnerability a pentester need to concern themselves with what they are permitted to do in order to find them. Swedish law lay down a few limitations. Here are three laws of concern. First, Brottsbalken prohibits attack or tamper devices or services without permission. It also prohibited to without proper permission read messages in computer networks.[71] Furthermore, it is prohibited to disclose trade secrets unless justified by protecting public interest. [72] Finally, Swedish copyright law have consequences for pentesting in Sweden. Of note is that reverse engineering may be considered a form of copying. There are paragraphs permitting reverse engineering if the purpose is achieve interoperability with other programs. There are however no such allowances for security research, and thus reverse engineering has, *at best*, dubious legal status when done as part of a pentest. Further investigations into Threat 2: Video stream improperly encrypted and Threat 7: Video stream improperly encrypted as well as Threat 5: Video stream lost in noise packets and Threat 10: Video stream lost in noise packets may therefore be infeasible unless done in collaboration with Xiaomi, the manufacturer behind the camera.

Chapter 10

Conclusion & Future Work

This chapter will further reconnect with the research question with conclusions founded on the pentest results and related discussion. The chapter and thesis will end with a section reflecting on how the question could be addressed further in future work.

10.1 Conclusion

The Xiaomi Mi Home Security Camera 360°, while not perfect, set a high security standard for home consumer cameras. In particular, the cloud central design allow the camera to bypass issues common in earlier models. Moreover, the evaluation made here within, focused on the areas deemed to be of highest risk, have been able to demonstrate a single vulnerability, which while likely is wide spread, arguably has low damage potential. Naturally, vulnerabilities may hide in areas neglected in this thesis. There are also outstanding questions raised by the evaluation, e.g. how its cloud reliance affect privacy. However, in summary, this thesis supports that this camera is secure against third-party adversaries.

This conclusion provide information to help answer the central question. That question asked in what capacity smart home cameras today are more secure than earlier attempts. The evaluated camera were selected for its popularity as proxy for market relevancy, and thus maximize the impact on the question's answer. However, as noted already in the Introduction, more cameras need to be evaluated to answer the question thoroughly. The conclusions that this camera has a strong security posture support that modern cameras have improved upon their predecessors. Further, considering that the camera's cloud centric design avoided previous mistakes lends credence to that such design may improve

local security.

10.2 Future Work

The tests made in this project were those that, based on the threat model, were deemed to be to most relevant. Unfortunately, the list of identified threats is long. The evaluated camera may thus still be understood better by analysing more threats in depth. Further, future work may also evaluate more cameras to, without a doubt, help provide a firmer conclusion in regards to how IoT camera security develops over time.

Moreover, while a cloud centric design may help avoid local security issues, how do it affect other concerns? Are there privacy concerns associated? Could a decentralized approach provide the same security benefits? Furthermore, pentesting the cloud have been out of scope of this thesis. Its security posture is therefore an open question. Investigation into these in future work may expand the understanding of the trade-offs made in the cloud centric design.

References

[1] A. Grizhnevich, "IoT architecture: building blocks and how they work," *ScienceSoft*, Oct 2020. [Online]. Available: https://www.scnsoft.com/blog/iot-architecture-in-a-nutshell-and-how-it-works

[2] "Product page: Mi Home Security Camera 360° 1080P," Feb 2021, [Online; accessed 24. Feb. 2021]. [Online]. Available: https://www.mi.com/global/camera-360

[3] "UniFi Protect - Getting started," Feb 2021, [Online; accessed 24. Feb. 2021]. [Online]. Available: https://help.ui.com/hc/en-us/articles/360058454253-UniFi-Protect-Getting-started

[4] "Kasa Spot - User Guide," Feb 2021, [Online; accessed 24. Feb. 2021]. [Online]. Available: https://help.ui.com/hc/en-us/articles/360058454253-UniFi-Protect-Getting-started

[5] A. Guzman and A. Gupta, *IoT Penetration Testing Cookbook: Identify vulnerabilities and secure your smart devices*. Packt Publishing Ltd, 2017.

[6] B. Schneier, "Attack trees," 1999.

[7] *Mi Home Security Camera 360° 1080p User Manual*, Xiaomi, Room 001A, Floor 11, Block 1, No. 588 Zixing Road, Minhang District, Shanghai, China, [Online; accessed 1. Mar. 2021]. [Online]. Available: http://go.buy.mi.com/uk/servicecenter/file/Mi_Home_Security_Camera_360%C2%B0_1080p_uk?publicationId=20572&namespaceId=2&binaryId=12128

[8] A. Shostack, *Threat modeling: Designing for security*. John Wiley & Sons, 2014.

[9] "OWASP Internet of Things - Top 10," OWASP, Tech. Rep., dec 2018, [Online; accessed 2. Mar. 2021]. [Online]. Available: https://owasp.org/www-pdf-archive/OWASP-IoT-Top-10-2018-final.pdf

[10] "OWASP Web - Top 10," OWASP, Tech. Rep., 2017, [Online; accessed 23. Apr. 2021]. [Online]. Available: https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_%28en%29.pdf.pdf

[11] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, "Demystifying iot security: an exhaustive survey on iot vulnerabilities and a first empirical look on internet-scale iot exploitations," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2702–2733, 2019.

[12] P. Cerwall, "Ericsson mobility report," Ericsson, Tech. Rep., 2017.

[13] A. B. Brush, B. Lee, R. Mahajan, S. Agarwal, S. Saroiu, and C. Dixon, "Home automation in the wild: challenges and opportunities," in *proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2011, pp. 2115–2124.

[14] N. Kalbo, Y. Mirsky, A. Shabtai, and Y. Elovici, "The security of ip-based video surveillance systems," *Sensors*, vol. 20, no. 17, p. 4806, 2020.

[15] "CVE-2017-11632." Available from MITRE, CVE-ID CVE-2017-11632., Feburary 2018. [Online]. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-11632

[16] M. Stanislav and T. Beardsley, "Hacking iot: A case study on baby monitor exposures and vulnerabilities," *Rapid7 Report*, 2015.

[17] B. Cusack and Z. Tian, "Evaluating ip surveillance camera vulnerabilities," 2017.

[18] Y. Seralathan, T. T. Oh, S. Jadhav, J. Myers, J. P. Jeong, Y. H. Kim, and J. N. Kim, "Iot security vulnerability: A case study of a web camera," in *2018 20th International Conference on Advanced Communication Technology (ICACT)*. IEEE, 2018, pp. 172–177.

[19] A. Tekeoglu and A. S. Tosun, "Investigating security and privacy of a cloud-based wireless ip camera: Netcam," in *2015 24th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2015, pp. 1–6.

[20] P. A. Abdalla and C. Varol, "Testing iot security: The case study of an ip camera," in *2020 8th International Symposium on Digital Forensics and Security (ISDFS)*. IEEE, 2020, pp. 1–5.

[21] F. Loi, A. Sivanathan, H. H. Gharakheili, A. Radford, and V. Sivaraman, "Systematically evaluating security and privacy for consumer iot devices," in *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy*, 2017, pp. 1–6.

[22] "Course description for degree project in computer science and engineering." [Online]. Available: https://www.kth.se/student/kurser/kurs/DA231X?l=en

[23] S. Cirani, G. Ferrari, M. Picone, and L. Veltri, *Internet of things: architectures, protocols and standards*. John Wiley & Sons, 2018.

[24] "Google Books Ngram Viewer," Feb 2021, [Online; accessed 23. Feb. 2021]. [Online]. Available: https://books.google.com/ngrams/graph?content=internet+of+things&year_start=1950&year_end=2019&corpus=26&smoothing=0&case_insensitive=true#

[25] M. Weiser, "The computer for the 21 st century," *Scientific american*, vol. 265, no. 3, pp. 94–105, 1991.

[26] ——, "Hot topics-ubiquitous computing," *Computer*, vol. 26, no. 10, pp. 71–72, 1993.

[27] F. Wortmann and K. Flüchter, "Internet of things," *Business & Information Systems Engineering*, vol. 57, no. 3, pp. 221–224, 2015.

[28] G. Weidman, *Penetration testing: a hands-on introduction to hacking*. No Starch Press, 2014.

[29] A. Gupta, *The IoT Hacker's Handbook*. Springer, 2019.

[30] M. Berner, "Where's my car? ethical hacking of a smart garage," 2020.

[31] M. Howard and D. LeBlanc, *Writing secure code*. Pearson Education, 2003.

[32] "Threat Modeling | OWASP," Aug 2020, [Online; accessed 16. Apr. 2021]. [Online]. Available: https://owasp.org/www-community/Threat_Modeling

[33] B. W. J.D. Meier, Alex Mackman, "Threat Modeling Web Applications," May 2005, [Online; accessed 16. Apr. 2021]. [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff648006(v=pandp.10)?redirectedfrom=MSDN

[34] L. Kohnfelder and P. Garg, "The threats to our products," *Microsoft Interface, Microsoft Corporation*, vol. 33, 1999.

[35] B. Russell and D. Van Duren, *Practical Internet of Things Security: Design a security framework for an Internet connected ecosystem*. Packt Publishing Ltd, 2018.

[36] O.-J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, *Structured programming*. Academic Press Ltd., 1972.

[37] L. Osterman, "Threat Modeling Again, STRIDE per Element," Mar 2021, [Online; accessed 1. Mar. 2021]. [Online]. Available: https://docs.microsoft.com/en-us/archive/blogs/larryosterman/threat-modeling-again-stride-per-element

[38] C. Salter, O. S. Saydjari, B. Schneier, and J. Wallner, "Toward a secure system engineering methodolgy," in *Proceedings of the 1998 workshop on New security paradigms*, 1998, pp. 2–10.

[39] "ATT&CK 101," May 2018, [Online; accessed 27. May 2021]. [Online]. Available: https://www.mitre.org/capabilities/cybersecurity/overview/cybersecurity-blog/attck-101

[40] A. Shostack, "Experiences threat modeling at microsoft." *MODSEC@ MoDELS*, vol. 2008, 2008.

[41] "Appendix N: SDL Security Bug Bar (Sample)," Feb 2021, [Online; accessed 17. Feb. 2021]. [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/windows/desktop/cc307404(v=msdn.10)

[42] "Probability-impact assessment - Praxis Framework," Feb 2021, [Online; accessed 17. Feb. 2021]. [Online]. Available: https://www.praxisframework.org/en/library/probability-impact-assessment

[43] "CVSS v3.1 Specification Document," Jan 2021, [Online; accessed 18. Feb. 2021]. [Online]. Available: https://www.first.org/cvss/specification-document

[44] "Övervakningskameror - jämför priser och omdömen hos prisjakt," https://www.prisjakt.nu/c/overvakningskameror, (Accessed on 10/28/2020).

[45] "Google shuts down Xiaomi access to Assistant following Nest Hub picking up strangers' camera feeds (Update: Fully resolved)," *Android Police*, Jan 2020, [Online; accessed 12. Mar. 2021]. [Online]. Available: https://www.androidpolice.com/2020/01/17/uh-oh-xiaomi-camera-feed-showing-random-homes-on-a-google-nest-hub-including-still-images-o

[46] cmiguelcabral, "mjsxj05cm-hacks," Mar 2021, [Online; accessed 12. Mar. 2021]. [Online]. Available: https://github.com/cmiguelcabral/mjsxj05cm-hacks

[47] telmomarques, "xiaomi-360-1080p-hacks," Mar 2021, [Online; accessed 12. Mar. 2021]. [Online]. Available: https://github.com/telmomarques/xiaomi-360-1080p-hacks#features

[48] J. Li, Z. Li, G. Tyson, and G. Xie, "Your privilege gives your privacy away: An analysis of a home security camera service," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 387–396.

[49] G. F. Lyon, *nmap(1) - Linux man page*.

[50] "Man-in-the-Middle (MITM) Attacks: Techniques and Prevention," [Online; accessed 28. Apr. 2021]. [Online]. Available: https://www.rapid7.com/fundamentals/man-in-the-middle-attacks

[51] J. Forshaw, *Attacking network protocols: a hacker's guide to capture, analysis, and exploitation*. No Starch Press, 2017.

[52] I. Vanney, "hping3 flood ddos," 2020, [Online; accessed 28. Apr. 2021]. [Online]. Available: https://linuxhint.com/hping3

[53] "UDP Garbage Flood," [Online; accessed 5. May 2021]. [Online]. Available: https://kb.mazebolt.com/knowledgebase/udp-garbage-flood

[54] P. R. Babu, D. L. Bhaskari, and C. Satyanarayana, "A comprehensive analysis of spoofing," *International Journal of Advanced Computer Science and Applications*, vol. 1, no. 6, pp. 157–62, 2010.

[55] J. Birch, "Exploring Background Execution Limits on Android Oreo," *Medium*, Jun 2018. [Online]. Available: https://medium.com/exploring-android/exploring-background-execution-limits-on-android-oreo-ab384762a66c

[56] A. P. Felt, H. J. Wang, A. Moshchuk, S. Hanna, and E. Chin, "Permission re-delegation: Attacks and defenses." in *USENIX security symposium*, vol. 30, 2011, p. 88.

[57] "Path Traversal | OWASP," Dec 2020, [Online; accessed 28. Apr. 2021]. [Online]. Available: https://owasp.org/www-community/attacks/Path_Traversal

[58] "Reverse Engineering Network Protocols," Mar 2018, [Online; accessed 3. May 2021]. [Online]. Available: https://jhalon.github.io/reverse-engineering-protocols

[59] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[60] P. Dorfinger, G. Panholzer, and W. John, "Entropy estimation for real-time encrypted traffic identification (short paper)," in *International workshop on traffic monitoring and analysis*. Springer, 2011, pp. 164–171.

[61] "UDP vs. TCP and Which One to Use for Video Streaming," Wowza Media Systems, Dec 2020, [Online; accessed 4. May 2021]. [Online]. Available: https://www.wowza.com/blog/udp-vs-tcp

[62] "L7-Filter | ClearFoundation | Deep Packet Inspection," May 2021, [Online; accessed 3. May 2021]. [Online]. Available: https://l7-filter.clearfoundation.com

[63] K. G. Hartman, "Calculate File Entropy," May 2013, [Online; accessed 3. May 2021]. [Online]. Available: https://kennethghartman.com/calculate-file-entropy

[64] R. Braden, "Rfc1122: Requirements for internet hosts-communication layers," 1989.

[65] J. F. Kurose, *Computer networking: A top-down approach featuring the internet, 3/E*. Pearson Education India, 2005.

[66] "networking:kernel_flow [Wiki]," The Linux Foundation, Sep 2020, [Online; accessed 5. May 2021]. [Online]. Available: https://wiki.linuxfoundation.org/networking/kernel_flow

[67] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 1996.

[68] I. Darwin, *file(1) - Linux man page*.

[69] H. Bidgoli, *Handbook of Information Security, Threats, Vulnerabilities, Prevention, Detection, and Management*. John Wiley & Sons, 2006, vol. 3.

[70] "Coordinated vulnerability disclosure: the guideline," 2018. [Online]. Available: https://english.ncsc.nl/binaries/ncsc-en/documents/publications/2019/juni/01/coordinated-vulnerability-disclosure-the-guideline/WEB_Brochure-NCSC_EN.pdf

[71] "Brottsbalk (1962:700)." [Online]. Available: https://lagen.nu/1962:700

[72] "Lag (2018:558) om företagshemligheter." [Online]. Available: https://lagen.nu/2018:558

[73] A. Ghedini, "Encrypt it or lose it: how encrypted SNI works," *Cloudflare Blog*, Dec 2019. [Online]. Available: https://blog.cloudflare.com/encrypted-sni

[74] "Iana's official list of ports," IANA, 03 2021. [Online]. Available: https://www.iana.org/assignments/port-numbers

[75] *Netify Agent - Packet Capture Files*, Netify, [Online; accessed 3. May. 2021]. [Online]. Available: https://www.netify.ai/developer/netify-agent/packet-capture

Appendix A

Network data

A.1 Xiaomi Mi Home Security Camera 360° 1080P

This section will present network traffic data gathered while doing information gathering on the Xiaomi Mi Home Security Camera 360° 1080P.

A.1.1 Camera communications

This section will present data related to network traffic going to and from the camera during boot and video stream initialisation. Some analysis of the traffic is also done here. The IPs that appear in the log have been matched to domain names. The hosts are grouped into four groups based upon the traffic and the host names.

IP	Domain name	Comment
47.254.135.53	-	
47.254.151.46	-	
47.254.176.66	-	

Table A.1: Group named *Triplet*.

The three addresses in *Triplet* group (Table A.1) has no associated domain names. These are grouped based upon that the camera send the same message to all three. The camera communicates with these hosts over UDP. (Replicating the messages to three servers could be for redundancy.) In the log three different types of messages to or from this group are found. The first kind is repeated

IP	Domain name	Comment
18.185.150.236	de.business.smartcamera.api.io.mi.com	
35.158.162.71	de.business.smartcamera.api.io.mi.com	Not found in logs, but found in DNS record for domain name.

Table A.2: Group named *Smartcamera API*.

IP	Domain name	Comment
15.164.195.115	gm.iotcplatform.com	
50.7.97.53	gm.iotcplatform.com	
198.16.70.62	gm.iotcplatform.com	
47.112.127.239	cm.iotcplatform.com	
114.67.98.218	cm.iotcplatform.com	

Table A.3: Group named *IoT Cloud*.

IP	Domain name	Comment
18.159.88.239	de.ot.io.mi.com	Camera has regular UDP communication with this host.
18.159.56.174	de.ott.io.mi.com	The camera establishes a TCP connection with this host and immediately closes it.

Table A.4: Group named *OT*.

every fifth second and does not get a response. Every 40th second the camera sends a message that the triplet echoes almost identically. (Not every server respond every time though, maybe due to message loss.) The last kind is initiated by the server and just precedes the camera initialising the video stream connection to the mobile application.

The second group is the *Smartcamera API* (Table A.2). The logs only contain one address belonging to this group, the other one were found through the DNS records for the domain associated with the address in the logs. The camera establishes two TLS connection with this host. The domain name the camera connects to is leaked through Server Name Indication (SNI).[73] Both of these connections are made to port 443, which is the default port for HTTP over TLS (HTTPS). [74] That the traffic actually is TLS and the port number combined makes it very likely that the inner protocol is HTTPS. Only a minor amount of data is sent over these connections. In both instances the camera sends some payload and shortly after receive a reply. This pattern of

communication fits the request and response structure of HTTP, strengthening the hypothesis that traffic is HTTPS. Given that the domain name contain `api` indicates that this host is an HTTPS API that is accessed during the boot sequence.

The third group, the *IoT Cloud* group (Table A.3), is reminiscent of the triplet group in some ways. The camera use UDP to send the same message to all hosts in this group and receives almost identical responses from some but not all hosts. The camera has been observed to send this message once or twice during the boot sequence. The domain name is matched through DNS queries the camera made. The `iotc` in the domain name could be an acronym for just *IoT Cloud*, hence the name of the group.

The fourth and final group, *OT* contains two hosts. The two hosts are grouped together due to similar domain names. The camera does not communicate with these hosts in a similar manner though. With one of the hosts it establishes a TCP connection, which it immediately closes again without sending any data. With the other host the camera has a conversation over UDP. The conversation starts with a message from the camera followed by a message from the host. This pattern then repeats, indicating that some request response protocol is followed. However, at a later point, correlated in time with when the mobile application is opened a message is sent from the host which the camera responds to. Due to the timing, this could have something to do with the application opening, e.g. the application querying the camera for some information.

A.1.2 Application communications

This section will present data related to network traffic going to and from the mobile application when opening it and initialising a video stream. Like when presenting the traffic to and from the camera, the IPs in these logs are presented with matched domain names. The hosts are also grouped based upon the traffic and their matched domain names.

Capturing the traffic specific to the camera companion application were a bit trickier than capturing the traffic to and from the camera itself. When monitoring the traffic to and from the camera, all traffic to and from that device were of interest, but in the case of the mobile application it is only one of many possible sources of traffic. The mobile OS or other application on the phone also generates traffic. To filter out these types of traffic a baseline where the target application was closed was recorded. A capture filter were then designed to filter out this traffic.

IP	Domain name	Comment
15.207.92.143	act.account.xiaomi.com	HTTPS
161.117.94.168	account.xiaomi.com	HTTPS
161.117.94.141	api.account.xiaomi.com	HTTPS

Table A.5: Group named *Account*.

IP	Domain name	Comment
183.84.6.228	home.mi.com	HTTP
58.83.177.231	home.mi.com	HTTPS
18.192.232.218	de.home.mi.com	TCP connection attempted but reset by the camera.
18.159.192.64	de.home.mi.com	HTTPS

Table A.6: Group named *Home*.

IP	Domain name	Comment
47.74.174.54	data.mistat.xiaomi.com	HTTP
161.117.204.141	data.mistat.intl.xiaomi.com	HTTP

Table A.7: Group named *Statistics*.

IP	Domain name	Comment
52.29.172.150	fr.app.chat.global.xiaomi.net	XMPP

Table A.8: Group named *XMPP*.

IP	Domain name	Comment
120.133.33.3	userprofile.mina.mi.com	HTTPS
183.84.6.210	userprofile.mina.mi.com	A connection attempt is made by the camera to this host at the same time as with the one above, but is abandoned.

Table A.9: Group named *Profile*.

The mobile application is very talkative in comparison to the camera. The application talks to more than double the number of individual IP addresses and more than thrice as many domain names. It is notable that the absolute majority of these conversations either can be confirmed to have been HTTP or suspected (due to TLS on port 443) to be HTTPS conversations. This traffic

IP	Domain name	Comment
124.251.58.59	hd.mina.mi.com	HTTPS
183.84.6.203	hd.mina.mi.com	A connection attempt is made by the camera to this host at the same time as with the one above, but is abandoned.

Table A.10: Group named *HD*.

IP	Domain name	Comment
163.171.134.120	cdn.alsgp0.fds.api.mi.img.com	HTTPS

Table A.11: Group named *Image*.

IP	Domain name	Comment
120.133.33.1	api2.mina.mi.com	HTTPS
183.84.5.179	api2.mina.mi.com	A connection attempt is made by the camera to this host at the same time as with the one above, but is abandoned.
161.117.95.80	api.miwifi.mi.com	HTTPS
18.195.211.50	de.api.io.mi.com	HTTPS
183.84.6.21	api.io.mi.com	HTTPS
124.251.58.87	api.io.mi.com	The first message of a TCP handshake is sent but no message is ever received. No retransmission is done either.
120.92.65.5	api.io.mi.com	The first message of a TCP handshake is sent and response is received. Camera resets connection.

Table A.12: Group named *API*.

IP	Domain name	Comment
69.171.250.15	graph.facebook.com	HTTPS

Table A.13: Group named *Analytics*.

hit domain names which indicate the traffic were related to querying APIs, account information, profile information as well as logging some statistics and analytics. Of note is that the requests to the statistics endpoints and one ”home” were over unencrypted HTTP. The only UDP messages that were sent when opening the application were DNS queries. When the video stream is initialised some additional UDP conversation were held with the triplet and IoT cloud

groups. The video stream itself from the camera were over UDP as well. A TCP connection were dedicated to XMPP (maybe for sending server generated events such as notification).

Appendix B

Scripts

Listing B.1: Script for running a comprehensive port scan against a host.

```
#!/usr/local/bin/fish

# Perform TCP SYN scan
echo "#### Commencing TCP SYN scan ####"
nmap -vv -Pn -sS -T3 -p0-65535 $argv[1]
echo

# Perform SCTP INIT scan
echo "#### Commencing SCTP INIT scan ####"
nmap -vv -Pn -sY -T3 -p0-65535 $argv[1]
echo

# Perform UDP scan
echo "#### Commencing UDP scan ####"
nmap -vv -Pn -sU -T3 -p0-65535 $argv[1]
```

Appendix C

Use Cases

Diagram over use cases 01-04 from section section 7.2 Architecture overview. Use cases 05-08 considers a different user than the four depicted, but are otherwise the same. Therefore the diagrams over use case $k \in 5, 6, 7, 8$ are the same as over use cases $k - 4$. Thus only one copy is displayed.

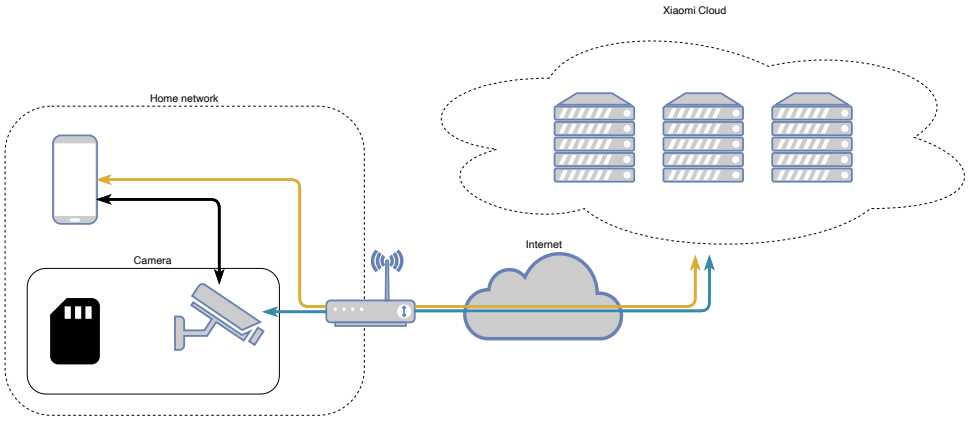


Figure C.1: Use case 01 & 05: User access real time stream on mobile device from same network as camera.

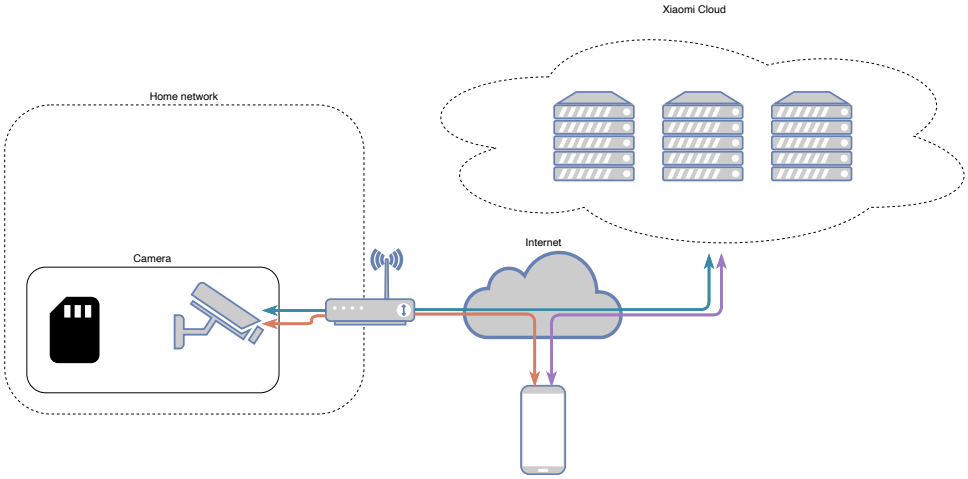


Figure C.2: Use case 02 & 06: User access real time stream on mobile device from different network than camera.

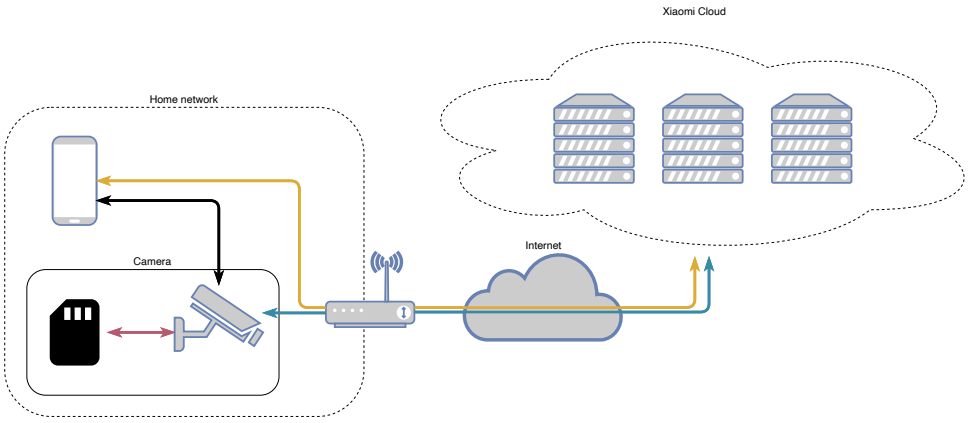


Figure C.3: Use case 03 & 07: User access recorded stream on mobile device from same network as camera.

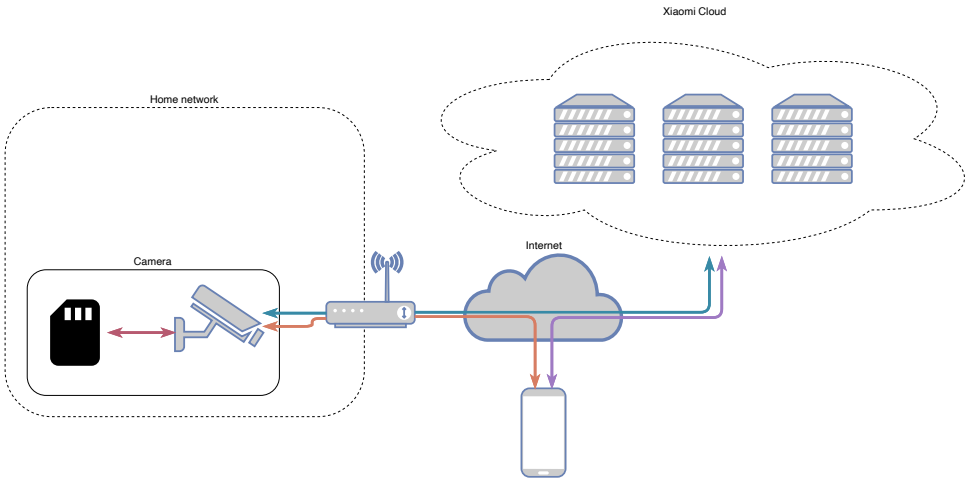


Figure C.4: Use case 04 & 08: User access recorded stream on mobile device from different network than camera.

Appendix D

Netify usage example

An usage example of netify from it's documentation. [75] Here we can see that Netify have identified a Network Time Protocol (NTP) connection and three HTTPS connections.

```
> netifyd -d -v -t -r -I lo,/tmp/netify.pcap

... preamble ...

lo: [i4----] NTP 91.189.89.199:123 <-- 192.168.4.189:52895
lo: [i4-g--] HTTPS.206.netify.cloudflare 162.159.135.234:443 --> 192.168.4.189:40328
lo: [i4----] HTTPS.10033.netify.netify 35.182.46.62:443 <-- 192.168.4.189:48540
        SSL C: sink.eg.netify.ai
lo: [i4----] HTTPS.10091.netify.amazon-aws 52.216.110.189:443 <-- 192.168.4.189:52974
        SSL C: s3.amazonaws.com
Caught signal: [35] Real-time signal 1: Update

Cumulative Packet Totals [Uptime: 0d 00:00:15]:
        Wire:      3.11 KP          ETH:      3.11 KP          VLAN:         0
          IP:      3.11 KP         IPv4:      3.11 KP         IPv6:         0
   ICMP/IGMP:         0             UDP:      2.69 KP          TCP:       423
        MPLS:         0            PPPoE:         0
       Frags:         0        Discarded:         1         Largest:      1.8 KiB

Cumulative Byte Totals:
        Wire:      2.15 MiB
          IP:      2.08 MiB         IPv4:      2.08 MiB         IPv6:         0
                              Discarded:       1.5 KiB         Flows:       22 (+22)
```

Appendix E

UDP Garbage Generator

Source can be found here.
(https://gits-15.sys.kth.se/jespel-exjobb-2020/udp_noise_generator)

```
use markov::Chain;
use pnet::packet::ip::IpNextHeaderProtocols;
use pnet::packet::MutablePacket;
use pnet::transport::transport_channel;
use pnet::transport::TransportChannelType::Layer3;
use serde::Deserialize;
use std::error::Error;
use std::fs::File;
use std::io::stdin;
```

```rust
use std::net::{IpAddr, SocketAddrV4};
use std::path::PathBuf;
use std::thread::JoinHandle;
use structopt::StructOpt;

fn main() -> Result<(), Box<dyn Error>> {
    // Load program options
    let opt = Opt::from_args();
    if opt.verbose {
        println!("{:?}", opt)
    }

    // Load payloads
    println!("Loading payloads file...");
    let payloads_file = File::open(opt.payloads_path)?;
    let payloads: Vec<Payload> = bincode::deserialize_from(payloads_file)?;
    println!("Loaded {} payloads", payloads.len());

    // Parsing sockets
    println!("Parsing sockets...");
    let source_socket: SocketAddrV4 = opt.source_socket.parse()?;
    let target_socket: SocketAddrV4 = opt.target_socket.parse()?;

    // Setup workers
```

```
    let worker = Worker {
        payloads: payloads.clone(),
        source_socket: source_socket.clone(),
        target_socket: target_socket.clone(),
        wait: opt.wait,
        verbose: opt.verbose,
        counter: opt.counter,
        max: opt.max,
    };

    let workers = vec![worker; opt.number_of_threads].into_iter();

    let join_handles = workers.map(Worker::start);

    for jh in join_handles {
        if let Err(_) = jh.join() {
            eprintln!("Thread panicked");
        }
    }

    Ok(())
}

#[derive(Clone)]
```

```
struct Worker {
    payloads: Vec<Payload>,
    source_socket: SocketAddrV4,
    target_socket: SocketAddrV4,
    wait: bool,
    verbose: bool,
    counter: Option<u32>,
    max: Option<u32>,
}

impl Worker {
    fn start(self) -> JoinHandle<()> {
        std::thread::spawn(move || {
            if let Err(e) = self.run() {
                eprintln!("{}", e)
            }
        })
    }

    fn run(self) -> Result<(), Box<dyn Error>> {
        let Worker {
            payloads,
            source_socket,
            target_socket,
```

```
    wait,
    verbose,
    counter,
    max,
} = self;

// Train markov chain
println!("Training␣markov␣chain...");
let mut payload_bytes_chain: Chain<u8> = Chain::of_order(3);
for payload in payloads.iter() {
    payload_bytes_chain.feed(&payload.bytes);
}
println!("Training␣complete");

// Prepare channel
println!("Network␣handle␣is␣being␣prepared");
let (mut sender, _) = transport_channel(
    10_000,
    Layer3(IpNextHeaderProtocols::Udp)
)?;

// Wait for user prompt
if wait {
    println!("Hit␣enter␣to␣start␣sending␣noise");
```

```
    let mut buffer = String::new();
    stdin().read_line(&mut buffer)?;
}

// Send noise
println!("Starting to send noise...");
let mut count = 0;
for noise_data in payload_bytes_chain.iter() {
    const IP_HEADER_SIZE: u8 = 20;
    const UDP_HEADER_SIZE: u8 = 20;

    let ip_total_length =
        IP_HEADER_SIZE as usize + UDP_HEADER_SIZE as usize + noise_data.len();
    let packet = vec![0u8; ip_total_length];
    let mut ip_packet =
        pnet::packet::ipv4::MutableIpv4Packet::owned(packet).unwrap();
    ip_packet.set_version(4);
    ip_packet.set_header_length(5);
    ip_packet.set_total_length(ip_total_length as u16);
    ip_packet.set_identification(count as u16);
    ip_packet.set_ttl(64);
    ip_packet.set_next_level_protocol(IpNextHeaderProtocols::Udp);
    ip_packet.set_source(source_socket.ip().clone());
    ip_packet.set_destination(target_socket.ip().clone());
```

```
ip_packet.set_checksum(
    pnet::packet::ipv4::checksum(&ip_packet.to_immutable())
);

let mut udp_packet =
    pnet::packet::udp::MutableUdpPacket::new(ip_packet.payload_mut())
    .unwrap();
udp_packet.set_source(source_socket.port());
udp_packet.set_destination(target_socket.port());
udp_packet.set_length(UDP_HEADER_SIZE as u16 + noise_data.len() as u16);
udp_packet.set_payload(&noise_data);
udp_packet.set_checksum(pnet::packet::udp::ipv4_checksum(
    &udp_packet.to_immutable(),
    source_socket.ip(),
    target_socket.ip(),
));

let n = sender.send_to(ip_packet, IpAddr::V4(target_socket.ip().clone()));
let n = match n {
    Ok(n) => n,
    Err(e) => {
        eprintln!("{}", e);
        continue;
    }
```

```
            };

            count += 1;
            if verbose {
                println!("Sent packet with {} bytes of noise", n);
            }
            if let Some(counter) = counter {
                if count % counter == 0 {
                    println!("{} packets of noise sent", count);
                }
            }
            if let Some(max) = max {
                if max <= count {
                    println!("{} packets of noise sent", count);
                    break;
                }
            }
        }

        Ok(())
    }
}

#[derive(Deserialize, Debug, Clone)]
```

```rust
struct Payload {
    bytes: Vec<u8>,
}

#[derive(Debug, StructOpt, Clone)]
#[structopt(name = "udp_noise_generator")]
struct Opt {
    /// Path to processed payloads
    #[structopt(short, long, parse(from_os_str))]
    payloads_path: PathBuf,

    /// Target socket
    #[structopt(short, long)]
    target_socket: String,

    /// Source socket
    #[structopt(short, long)]
    source_socket: String,

    /// Run in verbose mode
    #[structopt(short, long)]
    verbose: bool,

    /// Wait for user to hit enter before starting to send noise
```

```
    #[structopt(short, long)]
    wait: bool,

    /// Print sent packets every counter packet sent
    #[structopt(short, long)]
    counter: Option<u32>,

    /// Maximum amount of packets to send
    #[structopt(short, long)]
    max: Option<u32>,

    /// Number of threads to generate packets with
    #[structopt(short, long)]
    number_of_threads: usize,
}
```

For DIVA


```
{
"Author1": {
            "Last name": "Larsson",
            "First name": "Jesper A.",
            "Local User Id": "u12s5wlj",
            "E-mail": "jespel@kth.se",
            "organisation": {"L1": "School of Electrical Engineering and Computer Science ",
                             }
          },
"Degree": {"Educational program": "Degree Programme in Computer Science and Engineering"},
"Title": {
            "Main title": "Are modern smart cameras vulnerable to yesterday's vulnerabilities?",
            "Subtitle": "A security evaluation of a smart home camera",
            "Language": "eng" },
"Alternative title": {
            "Main title": "Undviker dagens smarta kameror gårdagens sårbaraheter?",
            "Subtitle": "Utvärdering av säkerheten hos en smart hem kamera",
            "Language": "swe"
},
"Supervisor1": {
                "Last name": "Johnson",
                "First name": "Pontus",
                "Local User Id": "u153b33i",
                "E-mail": "pontusj@kth.se",
                "organisation": {"L1": "School of Electrical Engineering and Computer Science ",
                                 "L2": "Computer Science" }
              },
"Examiner1": {
                "Last name": "Ekstedt",
                "First name": "Mathias",
                "Local User Id": "u18mctxd",
                "E-mail": "mekstedt@kth.se",
                "organisation": {"L1": "School of Electrical Engineering and Computer Science ",
                                 "L2": "Computer Science" }
              },
"Other information": {
"Year": "2021", "Number of pages": "xiii,114"}
}
```

TRITA -EECS-EX