Potential security risks in Google Nest Indoor Camera

JACOB KLASMARK
VALTER LUNDEGÅRDH
Potential security risks in Google Nest Indoor Camera

JACOB KLASMARK
VALTER LUNDEGÅRDH

Degree Programme in Information and Communication Technology
Date: June 26, 2020
Supervisor: Pontus Johnson
Examiner: Mathias Ekstedt
School of Electrical Engineering and Computer Science
Swedish title: Potentiella säkerhetsrisker hos Google Nest Indoor Camera
Potential security risks in Google Nest Indoor Camera / Potentiella säkerhetsrisker hos Google Nest Indoor Camera

© 2020 Jacob Klasmark and Valter Lundegårdh
Abstract

Our world is becoming more and more connected every day. Recently a new trend has taken off: smart homes. As people are investing in making their house connected this usually also includes some type of surveillance, more precisely a camera. A camera inside your home does however raise a question: how secure is it?

In this bachelor thesis an investigation of how secure the Google Nest Indoor Camera is was conducted. This camera was chosen because it is relatively cheap compared to other Nest cameras making it more attractive to your average person. In addition, being a camera developed by Google one would expect hefty security. Previous penetration tests did to the contrary however discover several vulnerabilities resulting in media attention across the globe. These factors combined gave us enough reason to conduct our research around this device.

A study around how secure the camera is was done. It began with an initial research phase to identify the current most crucial threats. Together with a threat model of the system and earlier research the threats were then summarised in a threat matrix. These were then ranked with DREAD as to delimit the amount of threats further so only the most potentially harmful ones could be investigated further. Penetration tests were then performed.

From our study we did not find any evidence of there existing vulnerabilities that could allow an adversary to view the camera feed or access other parts of ones home network through it. Even so, other, perhaps dangerous, vulnerabilities and exploits where found. This includes the ability to learn in which mode the camera is in or prohibit a user from connecting to the camera through Bluetooth.

Keywords

Hacking, Security, Internet of Things, Privacy, Camera
ii | Abstract
Sammanfattning


Från vår studie hittade vi inga bevis på befintliga sårbarheter som kan göra det möjligt för en tuff motståndare att se kameraflödet eller komma åt andra delar av ens hemnätverk genom den. Trots det hittades andra, kanske farliga, sårbarheter och utnyttjanden. Detta inkluderar möjligheten att lära sig i vilket läge kameran är i eller hindra en användare från att ansluta till kameran via Bluetooth.

Nyckelord

Hackning, Säkerhet, Sakernas internet, Integritet, Kamera
iv | Sammanfattning
Acknowledgements

This project has been a roller-coaster to say the least. It turns out that attempting to hack a camera from Google is not easy, who would’ve thought right? This is why we would like to thank certain people for their help.

Pontus Johnson and Mathias Ekstedt for their continuous support and guidance throughout this project. Your combined efforts made this thesis possible and we are extremely grateful for this.

Our friends and families as without them life would have seemed so dull. Their laughter and joy brought us happiness outside of the project and encouraged us to work even harder.

Finally ourselves for helping each other through the tougher times of this report. Friendship is truly the greatest of all powers.

Stockholm, June 2020
Jacob Klasmark and Valter Lundegårdh
Contents

1 Introduction 1
1.1 Introduction and background .......................... 1
1.2 Problem .............................................. 1
1.3 Purpose ............................................. 2
1.4 Goals ................................................ 2
1.5 Delimitations ........................................ 2
1.6 Sustainability and ethics ............................. 3
1.7 Structure of the thesis ............................... 3

2 Background 5
2.1 Google Nest Indoor Camera .......................... 5
2.2 Weave ................................................. 5
   2.2.1 Setup phase .................................... 6
   2.2.2 Video streaming phase .......................... 6
2.3 Terminology ........................................... 7
   2.3.1 Vulnerability .................................... 7
   2.3.2 Threat .......................................... 7
   2.3.3 Attack ......................................... 7
2.4 S.T.R.I.D.E ........................................... 7
2.5 DREAD ............................................... 8

3 Methodology 11
3.1 Method used ......................................... 11
3.2 Planning and discovery .............................. 11
   3.2.1 Open Web Application Security project (OWASP) 12
   3.2.2 Pre-study ........................................ 12
   3.2.3 Threat model .................................... 13
3.3 Penetration testing and Reporting ................. 13
# Threat model and Threat matrix

## 4.1 OWASP IoT top 10

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

## 4.2 OWASP top 10 Web Application Security Risks

1. Injection attack
2. Broken Authentication
3. Sensitive Data Exposure

## 4.3 Summary of OWASP

## 4.4 Previously discovered vulnerabilities

1. Bluetooth related vulnerabilities
2. Privacy Vulnerabilities in Google Nest Indoor Camera
3. Weave related vulnerabilities

## 4.5 Vulnerabilities on similar devices

1. D-link DCS-2132L
2. Other findings

## 4.6 Initial delimitations

## 4.7 The system

1. Main components
2. Communication channels

## 4.8 Threat report

## 4.9 Threat Matrix

1. Compilation of threats

# Risk evaluation

## 5.1 Risk assessment

1. TCP SYN Flood attack
2. Fuzzing attack
3. IP spoofing and HTTP requests
4. RST attack
5. ICMP attack
5.1.6 Outdated software .............................................. 45
5.1.7 Software update attack ......................................... 45
5.1.8 Fake certificate attack ......................................... 45
5.1.9 Bluetooth WiFi disassociation ................................. 45
5.1.10 Man-in-the-middle privacy vulnerability .................. 46
5.1.11 Buffer overflow ............................................... 46
5.1.12 Port scanning ................................................. 46
5.1.13 ARP spoofing ............................................... 46
5.1.14 Replay attack over TCP ..................................... 47
5.1.15 TCP DoS attack ........................................... 47
5.1.16 TCP repudiation ............................................ 47
5.1.17 Replay attack over Bluetooth ................................ 47
5.1.18 Bluetooth DoS attack ....................................... 48
5.1.19 DNS spoofing ............................................... 48
5.1.20 Bluetooth collision attack ................................... 48
5.1.21 Bluetooth repudiation ...................................... 48
5.1.22 Man-in-the-middle Bluetooth ............................... 49
5.1.23 Brute-force Bluetooth encryption ........................... 49
5.1.24 Brute-force data encryption over TCP .................... 49
5.2 Decision of attacks to investigate ............................. 51

6 Penetration testing ................................................. 53
6.1 ICMP attack - Ping of death .................................... 53
6.2 ICMP attack - Flood attack ..................................... 54
6.3 Port scanning ..................................................... 55
6.4 ARP-spoofing .................................................... 57
6.5 Man-in-the-middle privacy vulnerability .................... 58
6.6 Bluetooth Man-in-the-middle attack ......................... 61
6.7 Disabling Bluetooth connect-ability .......................... 62
6.8 Bluetooth Buffer overflow ...................................... 64
6.9 Bluetooth WiFi disassociation .................................. 65

7 Researched but not tested ......................................... 69
7.1 RST Attack ....................................................... 69
7.2 Software update attack ......................................... 69
7.3 Replay attack ................................................... 70
7.4 Fake Certificate attack ......................................... 70
7.5 Outdated software ............................................... 70
7.6 Injection .......................................................... 70
7.7 Broken Authentication ............................................. 71
7.8 Sensitive Data Exposure ............................................. 71

8 Discussion and Conclusions ........................................ 73
  8.1 Bluetooth security ................................................. 73
    8.1.1 Hardcoded password ......................................... 73
    8.1.2 PASE ............................................................. 74
    8.1.3 Additional aspects ........................................... 74
  8.2 TCP security ....................................................... 74
    8.2.1 Pairing and certificates .................................... 74
    8.2.2 Man-in-the-middle privacy vulnerability .................. 75
    8.2.3 Port scan and other TCP attacks ........................... 76
  8.3 Remaining attacks .................................................. 76
    8.3.1 ICMP attacks .................................................. 76
    8.3.2 Software update ............................................... 77
    8.3.3 ARP spoofing ................................................ 77
  8.4 Conclusion .......................................................... 77
  8.5 Future work .......................................................... 78
    8.5.1 6-digit pairing code risk .................................... 78
    8.5.2 Medium threats ............................................... 78
    8.5.3 ICMP crash scenario .......................................... 78
    8.5.4 Talos-Cisco TCP DoS attack ................................ 78
    8.5.5 Buffer overflow attacks ..................................... 79
    8.5.6 TLSv1.0 .......................................................... 79
    8.5.7 Delimitations ................................................ 79
  8.6 Reflection ............................................................ 79

References ................................................................. 81

A Threat Model Tool Matrix .......................................... 91
List of Figures

2.1 Threat rating table for DREAD ........................................... 9
3.1 Our Methodology .......................................................... 12
4.1 Stack in Linux ............................................................... 17
4.2 Eavesdropping on wireless channel between Alice and Bob ... 22
4.3 Threat model of the system generated with Microsoft threat modelling tool .................................................. 30
6.1 Bytes/second captured during test 1 ................................. 59
6.2 Bytes/second captured during test 2 ................................. 60
6.3 Bytes/second captured during test 3 ................................. 60
6.4 Camera unable to connect to ............................................ 63
List of Tables

4.1 Configurations tested ........................................ 32
4.2 Threat traceability matrix ................................. 36
5.1 DREAD risk assessment ................................. 50
A.1 Configurations tested .................................... 91
List of acronyms and abbreviations

ACK Acknowledge
ACL Access Control List
AES Advanced Encryption Standard
API Application Programming Interface
App Application
ARP Address Resolution Protocol

CASE Certificate-assisted Session Establishment
CVSS Common Vulnerability Scoring System

DDoS Distributed Denial of Service
DNS Domain Name System
DoS Denial of Service

ESP Encapsulating Security Payload

gatttool Generic Attribute tool
GPS Global Positioning System

hcitool Host Controlen Interface Tool
HTTP Hypertext Transfer Protocol
HTTPS Hypertext Transfer Protocol Secure

I/O Input/Output
ICMP Internet Control Message Protocol
ID Identification
IETF Internet Engineering Task Force
IoT Internet of Things
IP Internet Protocol
IPSec Internet Protocol Security

JSON JavaScript Object Notation

MAC
  Message Authentication Code
  Media Access Control
MITM Man in the Middle

OWASP Open Web Application Security project

PASE Password-Authentication Session Establishment
QR Quick Response
REST Representational state transfer
RFC Request for Comments
RST Reset
SEQ Sequence
SHA Secure Hash Algorithm
SSID Service Set Identifier
SYN Synchronize
TCP Transmission Control Protocol
TLS Transport Layer Security
TLV Tag Length Value
VM Virtual Machine
WiFi wireless fidelity
Chapter 1

Introduction

In this chapter, section 1.1 describes the introduction and background, section 1.3 the problem, section 1.3 the purpose and section 1.4 the goals of the project. After this comes the initial delimitations in 1.5 and sustainability and ethics aspects in 1.6. Finally 1.7 presents the structure of this bachelor thesis.

1.1 Introduction and background

Today many households move more and more towards a so called "smart-home" and it is expected that this market will grow by over 100 billion dollars between 2016 and 2025 [1]. A smart home is a house with gadgets connected to either each other, your phone or some other system. All this to make life more convenient for you. This however comes with a necessity, these systems should be secure so the use and data of these products can not be exploited by ill-meaning people.

This degree project will focus on the Google Nest Indoor Camera and the potential security vulnerabilities within it. We believe the security flaws of this type of product, that can monitor inside the house, need to be solved so the privacy of the user is not jeopardized.

1.2 Problem

The problem our work is based on is which, if there are any, security vulnerabilities the Google Nest Indoor Camera has. This should answer the question: How secure is the camera?
1.3 Purpose

The purpose is to shine light upon any security flaws present in the device in order to increase the integrity of future users. We also aim to provide a report that satisfyingly fulfill all the criteria for this thesis report.

1.4 Goals

The goal of the degree project is to investigate if the camera still has any vulnerabilities that can be exploited. This will be done based on previously found vulnerabilities, other similar products, the most common attacks that are applicable on the camera and its communication channels. If some are found, then an evaluation of the severeness of these vulnerabilities will also be done.

1.5 Delimitations

In this project we will not investigate potential weaknesses in the cloud, web Application Programming Interface (API), Nest Application (App) or communication between the camera and other Nest devices. These delimitations were made as to not break the law and to limit the scope of our work. The same line of thought was used for our decision to not look at hardware threats.

As this is also a thesis project which has to be handed in we must set a time delimitation. Thus we decided to only look at the most dangerous potential vulnerabilities more in depth.

We also decided to not look at the firmware. In the "IoT penetration testing cookbook" five different ways of acquiring firmware are presented [2].

- Downloading from vendor’s website
- Proxying or mirroring traffic during device updates
- Dumping firmware directly from the device
- Googling/researching
- Decompiling associated mobile apps
As the firmware is not available on the internet nor easily accessible during upgrades as the camera uses proper encryption only the dumping and decompiling options are left. These are however two options we believe are too time consuming and thus we will not move forward with these.

1.6 Sustainability and ethics

In this report we have paid sincere attention to the law as not to break it. First and foremost we have bought the camera and the wireless fidelity (WiFi) adapter for ourselves as to not break into someone’s property. All the communication we have intercepted is hence our own. During the project we never attempted to attack the cloud operated by Google Nest nor any other assets we do not own.

As a consequence the law is on our side as it is our right to conduct these experiments according to “observationsrätten”. We only attempted to access data, such as the video stream from the camera, which we already own and thus we have not overstepped any boundaries.

Furthermore our sole purpose was to improve the social sustainability in our society. Our project was carried out in the name of the general interest as we aimed to find and disclose vulnerabilities so that one’s integrity and home security would not be breached in the future.

This thesis project is also inline with the sustainable development goal 16: "Peace, Justice and Strong Institutions" as making sure a camera, for personal use, is secure helps protect a person’s fundamental freedoms and integrity.

Finally, if any vulnerabilities are found the vendor will be contacted. Only after that a patch has been published or a time limit of 90 days has passed, if nothing else is agreed upon, will the vulnerability be published to the public. This is to give the company an opportunity to develop a patch without putting their customers in an immediate danger. [3]

1.7 Structure of the thesis

The report is structured as follows. Chapter 2 specifies the relevant background information for this project. Chapter 3 declares the method used in our study. Chapter 4 presents our threat matrix which was then used in Chapter 5 to create our DREAD model. Chapter 6 then covers our penetrations tests and Chapter 7 the results from the tests not conducted but still researched. Our discussion and conclusions are then done in Chapter 8.
4 | Introduction
Chapter 2

Background

In this chapter the general background knowledge is presented. 2.1 presents information about the camera. 2.2 covers the communication protocol Weave and 2.3 some hacker terminology. Next, section 2.4 provides the reader with facts about the STRIDE model and 2.5 displays the DREAD threat rating model.

2.1 Google Nest Indoor Camera

Google Nest has a rather large selection of different cameras. The indoor camera is one of the cheaper options making it lucrative to the average person [4]. The relative low-cost compared to other products does however mean it is not equipped with the more technical options such as facial recognition. It does not store the video footage on itself but instead uses a WiFi connection to send data to a cloud server where the video is available to users.

The camera, and all its associated parts, uses the Weave protocol for the communication.

2.2 Weave

Weave is a communication protocol released in 2017. It was created with the purpose of providing low-power devices with a scalable, robust, versatile and secure network protocol [5].

The communication to and from the camera can be split up into two sessions, setup and video streaming.
2.2.1 Setup phase

During the setup phase Bluetooth is used to send the camera a WiFi Service Set Identifier (SSID) and password so that the camera can obtain a WiFi connection. This is done by first creating a secure connection to the camera. There are two options here, either scanning the Quick Response (QR) -code printed on the back of the camera or typing in a 6-digit code found next to the QR-code. This is used to create a Password-Authentication Session Establishment (PASE) between the device and client App. This results in a pairing process that cannot be subverted by an adversary, making sure that the WiFi password cannot be captured. [6]

Once this is all completed the camera will start sending data to the cloud server which a user can access through a Web API.

2.2.2 Video streaming phase

After the setup phase the camera goes into the streaming phase. Here the camera contacts the Nestlab’s server and authenticates the connection with the help of hardware certificates that is already coded into the camera during the production. The camera then initiates a Weave tunnel to have a secure connection to the cloud with the Transport Layer Security (TLS).

Transport Layer Security

TLS is used for providing security to the transport layer. It is often implemented on top of Transmission Control Protocol (TCP). TLS makes sure the connection is private with the help of symmetric cryptography. This symmetric key is generated uniquely for every connection with the help of a TLS Handshake Protocol. TLS also makes sure the connection is reliable by checking the integrity of messages with a keyed Message Authentication Code (MAC).

TLS Record Protocol is used to encapsulate higher level protocols. This allows the handshake, with authentication, negotiation of shared secret and integrity, to take place before the application receive its first byte of data. [7]
2.3 Terminology

2.3.1 Vulnerability
A vulnerability in a system could be something as an insecure file.

2.3.2 Threat
Given access to this file an adversary could then take control of the entire system. This is what is known as a threat.

2.3.3 Attack
An attack is an action taken by an adversary as to enact the threat by abusing the vulnerability.

2.4 S.T.R.I.D.E

When creating a secure system there are several approaches. One could for example be reactive and wait until the bug reports arrive or analyze the code after shipping. The problem with these methods is that the focus is not on the most critical thing, that is the system itself. After all, the system is what could be insecure so perhaps one should focus on potential threats earlier and be proactive. [8]

This is why two engineers at Microsoft decided to create S.T.R.I.D.E, which stands for [9].

- **Spoofing of user identity:**
  An adversary could impersonate a legitimate user and be given access to the system.

- **Tampering with data**
  Modifying system and or user data with or without detection.

- **Repudiability**
  The performance of illegal operations, such as attempting to break into an account, by an untrusted user without being detected.
- **Information disclosure**
  The ability to access information that should be protected, or in other words private.

- **Denial of Service (DoS)**
  Making a service or system unavailable or unusable for a period of time.

- **Elevation of privilege**
  A user obtaining a higher privilege level than originally intended and hence being able to make potentially dangerous adjustments to the system.

It is a security threat model and its purpose is to identify possible threats early on in a project. That is, the threats are identified based on the design of the system which is undeniably one of the most reliable factors.

### 2.5 DREAD

The risk of an attack is usually classified as the probability of it happening multiplied with the damage potential [10]. However, rating the probability and damage potential on a scale 1-10 is problematic as it is too simplistic and people might not be able to agree. Due to this, DREAD was created to add some dimensions so that it would be easier to determine which score a vulnerability would get.

- **Damage potential**
  How great is the damage if the vulnerability is exploited?

- **Reproducibility**
  How easy is it to reproduce the attack?

- **Exploitability**
  How easy is it to launch an attack?

- **Affected users**
  As a rough percentage, how many users are affected?

- **Discoverability**
  How easy is it to find the vulnerability?
When rating the different attacks one would then give each category a score based on the figure 2.1. The table was taken from Microsoft’s book "Improving Web Application Security: Threats and Countermeasures" [10].

<table>
<thead>
<tr>
<th>Rating</th>
<th>High (3)</th>
<th>Medium (2)</th>
<th>Low (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D Damage potential</td>
<td>The attacker can subvert the security system; get full trust authorization; run as administrator; upload content.</td>
<td>Leaking sensitive information</td>
<td>Leaking trivial information</td>
</tr>
<tr>
<td>R Reproducibility</td>
<td>The attack can be reproduced every time and does not require a timing window.</td>
<td>The attack can be reproduced, but only with a timing window and a particular race situation.</td>
<td>The attack is very difficult to reproduce, even with knowledge of the security hole.</td>
</tr>
<tr>
<td>E Exploitability</td>
<td>A novice programmer could make the attack in a short time.</td>
<td>A skilled programmer could make the attack, then repeat the steps.</td>
<td>The attack requires an extremely skilled person and in-depth knowledge every time to exploit.</td>
</tr>
<tr>
<td>A Affected users</td>
<td>All users, default configuration, key customers.</td>
<td>Some users, non-default configuration.</td>
<td>Very small percentage of users, obscure feature; affects anonymous users</td>
</tr>
<tr>
<td>D Discoverability</td>
<td>Published information explains the attack. The vulnerability is in a seldom-used part of the product, and only a few users should come across it. It would take some thinking to see malicious use.</td>
<td>The vulnerability is in a seldom-used part of the product, and only a few users should come across it. It would take some thinking to see malicious use.</td>
<td>The bug is obscure, and it is unlikely that users will work out damage potential.</td>
</tr>
</tbody>
</table>

Figure 2.1: Threat rating table for DREAD
Chapter 3

Methodology

In this part of the thesis the method for the project is shown which has been divided into three parts. Section 3.1 presents the method used which 3.2 and 3.3 describes more in depth.

3.1 Method used

In our thesis project we used the methodology presented in Figure 3.1. To guarantee that we did not miss any attacks we first had planning and discover stages. After that we conducted our penetration tests. However, to remain open to new vulnerabilities we also realised that new discoveries might arise after each attack. Continuously reporting, that is writing of this thesis report, was thus done throughout the entirety of the project after the planning phase and each attack.

To make sure errors were minimised each penetration test was conducted several times to make sure the outcome remained the same. If different tests showed different results they were reported and an attempt to understand why this happened was made.

3.2 Planning and discovery

The aim with our planning and discovery sections was to identify the potential threats our system could be susceptible to. To make sure we had not missed any crucial attacks we split up the work into three different parts.
3.2.1 **Open Web Application Security project (OWASP)**

To narrow down the report to the currently most crucial security issues developers should keep in mind when creating a system, the "Open Web Application Security project (OWASP) top 10 IoT vulnerabilities" from 2018 and the most relevant vulnerabilities of the "OWASP top 10 Web Application Security Risks" were used [11][12]. From this template we managed to identify the numerous attacks we moved forward with. Some were however removed due to our delimitations.

3.2.2 **Pre-study**

A pre-study was conducted in order to identify already discovered vulnerabilities. The assessments of the solutions and patches provided for each of these were also considered. We looked at the bug report and then the Github code, if available, where the vulnerability was addressed. We then made a decision to move forward with those threats we believed could still pose a threat to the system.
3.2.3 Threat model

To improve our understanding of the underlying mechanisms of the system, data was collected from documentation of the Weave protocol and real life testing where Wireshark was used to look at the communication between the different parts of the system.

Combined a threat model of the camera’s real network and communication protocols was created. Threats were later generated with the aid of Microsoft threat modelling tool.

3.3 Penetration testing and Reporting

After the threats from the research were discovered, to summarise all of the threats found, a compilation of these were made using a threat trace-ability matrix. They where then evaluated using DREAD and those deemed the most serious, based on the score given and in our general interest, were taken into our penetration testing session.

After each penetration test the results were written down. If future planned attacks were discovered to be harmless, from one of these penetration tests, it was documented. The affected tests were then not conducted. The same methodology was used for potentially new vulnerabilities found during testing. However, these were of course then penetration tested on their own.

Based on the results from the penetration testing an evaluation of the system’s security was done to the best of our abilities.
Chapter 4

Threat model and Threat matrix

This chapters contains 4.1 and 4.2 which covers the OWASP top 10 risks in Internet of Things (IoT) and general devices. These sections are then summarised in 4.3. Section 4.4 then goes into previously discovered vulnerabilities in Google Nest cameras while 4.5 does the same for other devices. Section 4.6 then continues the chapter with some initial delimitations for the thesis project. The system is then described in 4.7 which is followed by the threat report in 4.8. The chapter is then finalised with the threat matrix in 4.9.

4.1 OWASP IoT top 10

To make sure we have covered the most current crucial attacks we decided to use the top 10 OWASP list for IoT things [11]. The ten attacks are listed in the sections below.

4.1.1 Weak Guessable or Hardcoded Passwords

If the device uses these types of passwords there is an increased risk the device can be hacked. For instance the credentials might be publicly available or unchangeable making them easy to find.

Brute-force attack

A brute-force attack could also be used in which the hacker simply tries all possible combinations until the correct one is found. The time to crack a username or password using this method depends on the complexity of data. With a good enough encryption the time to brute-force would be unreasonable and hence the system would be deemed secure. [13]
4.1.2 Insecure Network Services

If the device is exposed to the internet, that is, it allows arbitrary devices to connect to it, the risk of a hacker finding a vulnerability through this gateway rises. Open communication ports is one of these vulnerabilities which the attacker uses as a spring board to later create more damage. One method often used to identify these weaknesses is the port scanning attack.

Port scanning

There are three kinds of port scans. Vertical port scan where you scan several ports on a single Internet Protocol (IP), horizontal port scan where you scan a single port on several IPs and block scan which is a combination of both horizontal and vertical. The most common scan is a TCP Synchronize (SYN) scan which is used to discover open TCP ports. First a connection is initiated with a SYN-flagged packet. If the port is open a SYN or Acknowledge (ACK) packet is sent back and if the port is closed a Reset (RST) packet is sent. [14]

An open port is an attack surface. The daemon listening to the port may have exploitable vulnerabilities or be exposed to other attacks. One way of defending against port scans is to have a firewall block the packets so no response at all is sent back. Another method is to have all unnecessary ports closed and have the daemon, listening to the ports that are open, properly configured. [15]

Often open ports can later lead to other types of attacks taking place [16]:

Fuzzing attack

Fuzzing is a way of testing or attacking a system or program in order to find bugs that may cause problems. This method treats the program as a black box, and the input that is sent to the program is in some sense random. The idea is to either brute force every possible input or at least try edge cases related to the program. One way of approaching this is to have a list of "known-to-be-dangerous values". For example, for inputting numbers this could include zero, very large numbers or negative integers. If the program only expects numbers between 1 and 7 to appear, other values could cause unexpected crashes if they are not handled correctly. [17]
Buffer overflow attack

In a computer a buffer is a sequential piece of memory used to store data or code. Let us assume we have a Linux computer running some C code. When the code starts a function, certain things will be placed on the memory’s stack. The layout will be similar to the one showed in Figure 4.1. The figure is based on the work of Computerphile [18].

It includes several things but the most important ones here are the buffer and the return address. The buffer could for example be an array in the function where a string inputted from the command line is stored, in our case perhaps a SSID key or encrypted password parameter. The return address is a way of telling the processor for where it should return after the function is done.

Now, a buffer overflow occurs when more data than can fit is written to this location causing it to overflow. The data that cannot be placed there but instead has to go somewhere and can overflow, or in other words overwrite, memory in adjacent memory locations outside the buffer’s memory. This is dangerous because as the the Figure 4.1 shows it can alter the following parts of the stack such as the return address. [19]

One obvious danger with this is if the following parts of the stack are corrupted in such a way that they are no longer valid instructions. The program will most likely then be shut down by the processor as it realises it is trying to execute illegal commands. This forces a restart which disables the system for some time which of course rarely is desirable. However even worse things
could occur.

Assume that a hacker instead of a harmless string now writes malicious executable code to the buffer. As it is just a storage for the code nothing will happen by itself. However if the attacker is able to have the buffer overflow and modify the return address to point to the start of the buffer where the malicious code is stored it could be catastrophic. Since the return address points to a perfectly legal position to execute code the processor will not interrupt the program here. Instead, once the function is done it will look at the return address, recognise that it points to the buffer and simply run the code which was stored there. Essentially the hacker is hence able to execute any sort of code which could be used to take over the device itself.

To prevent this a good method is usually to make sure ones buffers cannot be overflowed by limiting the size of incoming data. One could also randomise the address space of the buffer, making it harder for the attacker to know where they should aim their attack as the address space would not be static [20].

**Denial of service attack**

A Denial of Service (DoS) attack’s purpose is to prevent the targeted device from operating at its full capacity. This is done with a single computer, in contrast to Distributed Denial of Service (DDoS) where a network of computers are used, usually by sending a large amount or specifically crafted messages.

It is often categorised in two different ways, buffer overflow or flooding. A buffer overflow DoS is done by causing a targets buffer to fill up from all the messages making it function slower, become sluggish and in some cases even crash. This is a consequence of the device having to process more data than usual.

A flood attack on the other hand is done by sending a flood of packets to the destination, consuming most of the network’s capacity. Thus, the consequence is an impairment of available service for other users on the same network. It does however require that the network has the capacity of receiving a high amount of messages. [21]

Two variants of flood attacks are TCP SYN and Internet Control Message Protocol (ICMP) flood.

**TCP SYN Flood attack**

The SYN flood attack is a DoS attack that tries to soak up resources from the server by establishing a TCP "three-way handshake". The attacker sends
multiple SYN packets to every port on the target’s server which believes these are all legitimate users asking for a connection. The server then responds with SYN-ACK to every connection and then waits for an ACK from the user. The server is hence wasting resources by keeping a connection open that no one will answer to as the ACK will never arrive. When the server has reached its limit of active TCP connections other legitimate users will not be able to connect. [22]

There are many ways to mitigate the power of such an attack. For instance, the time before the server drops the connection could be lowered as to quicker free resources. It is also possible to send specifically crafted packets back to the initiator of the connection to see if their response is correctly adapted based on this [22].

**ICMP attacks**

**ICMP** is an error-reporting tool devices use to send and receive error messages across a network. Most networks allow these to be sent around freely which hackers have exploited to create DoS attacks.

The ICMP flood attack exploits this by sending many ping messages rapidly without waiting for replies. This drains most of the network capacity which creates a DoS condition.

There also exists another **ICMP** attack called the ping of death attack. Here large ICMP packets are sent in fragments which once received by the target, tries to reassemble them. The final product is a packet that is so large that the device simply crashes as it cannot handle it. It also consumes network capacity as there will be several smaller packets sent instead of one larger.

A usual causation is that **ICMP** messages, or fragmented ping blocks, are not blocked on the network. [23, 24]

**RST attack**

By sending a **RST** message to the device, while spoofing, see 4.1.3, the IP address of the server in the active TCP session, the connection will be torn down. If an attacker is able to do this the device will not be able to send information for a period of time. This does require one to have the IP address of the device, IP address of the server, port of the device, and the port of the server. With this information it is possible to create a **RST** segment that stops this connection. The sequence number also has to be within the receive window for it to not be thrown away. This problem exists if the device does not verify that the **RST** segment is sent from the authenticated server. [25]
4.1.3 Insecure Ecosystem Interfaces

If the device uses a cloud service, web API or other external interfaces that lack the necessary security it is said to have insecure ecosystem interfaces. Typical causes for this is insufficient authentication, authorization and encryption for the device itself and the components related to it. A typical attack regarding this could be spoofing.

Spoofing attack

As the name states a spoofing attack is about a perpetrator spoofing or pretending to be someone else. This is done in an effort to gain access to unintended data or networks. There are many different kinds of spoofing aimed at different areas; email, Global Positioning System (GPS), Caller Identification (ID), Website and facial spoofing are a couple of them. [26]

There is also Address Resolution Protocol (ARP) and Domain Name System (DNS) spoofing. The former is about an attacker connecting its own Media Access Control (MAC) address to a legitimate IP-address on a network in order to trick the network into sending the data to them instead of the real user. Some ways of protection against this includes using a static ARP table and using encryption so that the attacker must also create a legitimate encrypted communication channel. In addition, filtering out packets that are from outside your network can help. [27]

The latter is similar to the first as it also attempts to reroute information sent but here the victims computer is infected through DNS poisoning. Different domain names are associated with IP-addresses. If an attacker is able to change the IP-address which a certain domain is paired with then the user will connect to a fake website possible exposing themselves to malware [28]. To prevent this, one could implement active monitoring of the DNS data. The Internet Engineering Task Force (IETF) also developed a DNS Security Extension to address these security threats. [29]

Furthermore, there also exists IP spoofing. Here the attacker pretends to be another system. An effective way of gaining access to a system is by spoofing an authorised IP-address tricking the victim into believing you are a trusted friend. As a result, the attacker can be given access to the host and or capture confidential data. [24]

To prevent IP spoofing an Access Control List (ACL) could be used to stop traffic coming into the network if it is claiming to have a local IP-address. It is also recommended to use cryptographic authentication so that an attacker will not be trusted automatically by a user through trust relationships. This is a
Threat model and Threat matrix | 21

vulnerability if only the IP-addresses are used to verify users [30]. Encrypting one’s traffic is also extremely useful as even if a perpetrator is able to capture the traffic it will be unreadable. An example of this is using an Internet Protocol Security (IPSec) tunnel. [24]

4.1.4 Lack of Secure Update Mechanism

If the device lacks the ability to update itself in a secure manner this could be a vulnerability. If a company sends a software update the device should be able to see that the update has not been tampered with and is from a legitimate host. To mitigate additional risks anti-rollback mechanisms and notifications of security changes are also important.

Software Update attack

A software update attack is when an attacker in some way manages to trick the device into updating itself with files that are not from the trusted source. This kind of attack could be possible if the device does not check the integrity of the update before installing it. If that is the case the attacker can pretend to send an update from the trusted source. This could both lead to malicious code being executed on the device and it can be hard for the device owner to discover where the intrusion happened since they think the updates to the device are real ones. It is important that the integrity of the updates are verified so no malicious activity can be committed. [31]

4.1.5 Use of Insecure or Outdated Components:

A device that uses hardware and or software that is outdated or has been proven to be insecure is a potential vulnerability. This concerns everything from the operative systems in use to physical components.

4.1.6 Insufficient Privacy Protection

An insufficient protection of users’ information within the device’s ecosystem could be a threat. Often the main cause of this is a lack of encryption of messages sent. For example, if a user’s credentials such as username and password are sent in plain text over a wireless communication link a simple Man in the Middle (MITM) attack could capture these.
Man-in-the-middle attack

MITM attacks or eavesdropping attacks are attacks where the adversary inserts itself into the communication between two parties [32]. Once this has happened the attacker can analyse and steal the data. As seen in Figure 4.2, Eve is eavesdropping and collecting data from the transmission between Alice and Bob.

There are two types of eavesdropping attacks [33]:

- **Passive eavesdropping**: The malicious nodes listen to the transmissions in the wireless medium and collects the information.

- **Active eavesdropping**: The malicious nodes disguise themselves as friendly nodes and sends queries to the transmitting units.

Passive eavesdropping is hard to detect because it is passive and does not interact with the users. One way to solve this could be encryption but this might not always be a possibility between IoT units since they are very dependent on battery life and are supposed to be cheap and simple. Adding an encryption will thus drain the energy faster,

4.1.7 Insecure Data Transfer and Storage

Similar to the one above this is also about the security of data but overall. If the system does not take the required precautions to store, send and process
information an attacker might be able to extract sensitive data. Usually this is a result of improper encryption of data. It might also be able to conduct a replay attack.

**Replay attack**

Also known as a playback attack. "A replay attack is a category of network attack in which an attacker detects a data transmission and fraudulently has it delayed or repeated." [34] This means that attackers can record a message sent from a legitimate user and send it again. Even though the message is encrypted the receiver will not notice this message is not from the original sender. For example a bank transaction sent from the user to the bank could be replayed by the attacker which makes the transaction become executed many times.

This can be prevented with the help of Sequence (SEQ) numbers and or timestamps to invalidate the messages that are old or replayed.

### 4.1.8 Lack of Device Management

It is important to manage the IoT devices connected to the network. One insecure device could possibly compromise the entire system. To keep your network components updated and secure is thus an important aspect of all large IoT systems.

### 4.1.9 Insecure Default Settings

Many people that use an IoT device does not bother to change the settings to something else other than the default settings. That means that it is important to have the most secure settings as the default one. It can also be important to have the opportunity to modify the configurations of the system in order to make it more secure. For example it would not be optimal to have the default admin-admin username and password. Potential attacks here are similar to the brute-force attacked described earlier here as they are both targeting weak security settings.

### 4.1.10 Lack of Physical Hardening

A lack in physical hardening basically means that an attacker has the ability to extract some sensitive information from the device that can make it easier for them to later get access to the device remotely. An example of this could be
that a memory card is easily removable from the device and that memory card contains for example passwords and login credentials [35].

### 4.2 OWASP top 10 Web Application Security Risks

Even though the camera is an IoT device some of the items on the OWASP top 10 Web Application Security Risks are also interesting to look at [12]. The remaining seven risks were deemed to not be of interest to this camera device or very closely related to the risks brought up in the IoT top 10.

#### 4.2.1 Injection attack

An attacker can inject untrusted data to the device as part of a command. This data tricks the device to execute unintended code or accessing unauthorized data.

#### 4.2.2 Broken Authentication

Authentication functionality is often implemented incorrectly which leads to the possibility that passwords, session tokens and or keys are compromised which leads to compromised authentication.

#### 4.2.3 Sensitive Data Exposure

It is important to properly protect the data with encryption. If this is not implemented an attacker could be able to extract sensitive and personal data.

### 4.3 Summary of OWASP

As some of these potential vulnerabilities fall under out delimitations, namely that we will not attempt hardware hacking, attacks regarding sections 4.1.5, 4.1.8 and 4.1.10 will not be investigated. Outdated software components from 4.1.5 will however be investigated.
4.4 Previously discovered vulnerabilities

In this section the previous work regarding security vulnerabilities in Google Nest cameras are presented.

4.4.1 Bluetooth related vulnerabilities

In 2017 Jason Doyle disclosed three vulnerabilities in some Google Nest cameras related to how the device communicated over Bluetooth using the Generic Attribute tool (gatttool).

Two of them were about the buffer overflow condition which could be triggered by either setting the SSID or the encrypted password parameter of the camera. By providing the device with an instruction to set one of these followed by a specific value it caused a buffer overflow which in turn forced the device shut down and restart.

This was possible as long as the attacker was within the Bluetooth range of the device during the entire attack. One cause for this was that the Bluetooth on the camera was never shutdown after initialisation phase.

The third attack was used to disable the camera by having it attempt to access a new WiFi. As the camera does not have any local storage the video stream was hence out-of-action during this process. If the provided WiFi SSID was faulty it would then take approximately 60-90 seconds before the camera reconnected to the previous network.

Since these were discovered Doyle published that they had all been patched in the Nest indoor camera. Yet as the open source code was published after this update from Doyle we cannot be sure that they have been patched properly and will thus explore these ourselves. [36]

4.4.2 Privacy Vulnerabilities in Google Nest Indoor Camera

In 2017 three scientist from Princeton performed a privacy and vulnerability test on four IoT devices that are common in a smart home [37]. One of these devices was the Google Nest Indoor Camera. They argue that even though the traffic is encrypted the metadata and traffic patterns can be intrusive for the users. They used a model where the adversary could use an eavesdropping attack and listen to all wide-area network traffic. The packets where all sent with TLS so the only information the adversary could access was the IP and TCP packet headers.
They found that it was possible to divide the data streams from the different devices and connect them to the device they came from by looking at the DNS they sent the packets to. The Google Nest Indoor Camera sent all its packets to a domain called dropcam.com.

Moreover, they also saw that when the Nest Camera was in live-streaming mode the traffic to and from the camera was significantly higher. It also showed a difference in data sent when there was motion in front of the camera while no one was watching the stream. This does provide privacy risk since an adversary could monitor the data stream and see when the motion detector detects movement inside the house or when a user is actively watching the camera.

### 4.4.3 Weave related vulnerabilities

At Cisco Talos two reasearchers, Lilith Wyatt and Claudio Bozzato, discovered eight vulnerabilities in connection to the Weave protocol used in the Google Nest camera. They were all patched before the disclosure in August 2019 and follow from highest to lowest Common Vulnerability Scoring System (CVSS) score down below. [38]

- **Weave PASE pairing brute force vulnerability [39]:**
  
  To connect to the camera a QR-code or a 6-figure code, found on the device, is used to authenticate a user. Even though the failed login attempts are not tracked a raw brute-force strategy here would take too long as the time for each round of authentication is lengthy.

  However, Talos Cisco discovered that the Nest Cam uses a fewer set of possible 6-figure codes, namely only six-digit codes used which are Verhoeff base32 checksum values. As a consequence, brute-forcing all of the sudden becomes more reasonable. Another issue is that this static code never changes as it is printed on the camera. Once the attacker successfully discovers the correct code, and depending on the state of the camera, they can possibly be allowed to send certain messages or even worse connect the camera to their own Nest account.

  This was patched by decreasing the possible amount of access attempts and not allowing more than 3 failures each 15 seconds [40].

- **DecodeMessageWithLength information disclosure vulnerability [41]:**
  
  It was possible to trigger an integer overflow that would make the data in the assigned buffer accessible.
This was solved by making the integer 32 bits instead of 16 so an overflow could not take place. The buffers were also cleared before they were handed out again to a new message. [42]

- **Weave Key Error denial of service vulnerability [43]:**
  When the camera received an error message, that was used to break up the connection to the cloud storage, there was a major flaw in the Weave protocol. The camera did not look at who the sender of the error message was. It was therefore possible for an attacker to exploit this by either sending Bluetooth or UDP messages to respectively break up TCP or 802.15.14 connections.
  This seems to have been fixed by Nest adding code which makes sure the messageID of an error message is the same as the current one. Also, the connection over which the received message was transported must be equivalent to the one used at the moment. A Bluetooth message could hence not break up a TCP connection.
  In addition, they also randomized the messageID counter from the start. Consequently, even if an attacker could find the key ID and source node ID, required to send one of these error messages, they would now also need the messageID. Thus, if they established a connection to the camera they would have to brute force the 32 bit messageID which would take a long time. This ID is also not static as it is always changing with new sent messages. [44]

- **Weave CASE Engine::DecodeCertificateInfo denial of service vulnerability [45]:**
  It was possible to set certain values in a Certificate-assisted Session Establishment (CASE) request to trigger an integer overflow condition causing the camera to crash. This was easily fixed by defining the variable, which was overflown, to 32 bits instead of 16 bits, so it could not happen [46].

- **OpenWeave Weave tool Print-TLV code execution vulnerability [47]:**
  When writing out the Tag Length Value (TLV) it was possible to cause a stack buffer overflow by having a device execute the print TLV command. By simply asking a device to execute this command on a file provided by the adversary unwanted behaviour could occur.
Intelligently Google Nest patched this by instead of having a fixed size buffer they now enable arbitrary lengths to the TLV depth, which a causation of this vulnerability [48].

- **ASN1Writer PutValue code execution vulnerability [49]:**
  It was possible to cause a heap buffer and integer overflow during a change of certificate used for the encoding of messages. However, this was fixed by implementing checks to avoid the integer overflow and changing the variable, causing the heap buffer overflow, from a 16 bit unsigned int to a 32 bit one. Given this, checks could no longer be bypassed and therefore the vulnerability was nullified. [50].

- **Weave legacy pairing information disclosure vulnerability [51]:**
  There existed a possibility to craft Weave packets that can cause out of bounds reads potentially resulting in information disclosure. By not checking the size of the nonce Cisco Talos were able to brute force information byte by byte from the Packet-buffer. Nest fixed this by earlier checking that the nonce size actually was the size of the nonce [52].

- **Weave TCP connection denial of service vulnerability [53]:**
  The Weave protocol used in the camera only allowed for up to 10 TCP connections at the same time. An attacker could trivially set up several different TCP connections, thus taking up all the available slots. Furthermore, there was no timeout for a TCP connection so once connected a link was never torn down.
  
  To solve this, Google Nest made a commit to change the idle timeout from infinity to 15000 ms [54]. They also seem to have increased the number of allowed TCP slots while only allowing two connections from the same IP-address [55]. However it is still questionable if it is still possible to achieve a DoS attack if one is able to connect the 51 slots and consecutively send messages to keep the connection up. This would potentially be doable with a network of computers.

### 4.5 Vulnerabilities on similar devices

In this section will we go through vulnerabilities on other cameras that is not the Google Nest Camera and hopefully find some vulnerabilities that can be applied to the Google Nest Camera as well.
4.5.1 D-link DCS-2132L

The D-link camera is another camera with almost the same functions as the Google Nest Camera. It is a web camera that sends data to a cloud which is then viewed from the "mydlink Lite App" [56]. It also has a tunnel between the camera and the cloud, and the cloud to the application [57].

One vulnerability found in this camera is that all Hypertext Transfer Protocol (HTTP) requests from 127.0.0.1 are elevated to admin level. This gives the attacker access to the entire device and the possibility to compromise the system. [57]

4.5.2 Other findings

Moreover, the Nest camera differs a lot from other cameras’ previously discovered vulnerabilities. This means that there are many vulnerabilities, according to us, that are not of interest to this thesis as Nest by default has already solved these.

For example, in Kim Jonatan Wessel Bjørneset’s master thesis he goes over a couple of cameras that have weakness in their communication as they are using HTTP for sending data [58]. As Nest encrypts its data, assuming the algorithm for this is implemented correctly, these attacks can be neglected.

Nest also uses Weave so weaknesses in others cameras communication protocols can not directly be applied to it. We did however look at other Nest devices. It did not yield any meaningful results however as the vulnerability was regarding the USB cable in the Nest thermostat, something that our camera does not even use [59].

In addition, to view the camera feed a user does not connect to the camera directly but instead a Nest server through their own account. This adds another layer to the security but also a lot of infrastructure that could be at risk of hacking. Though, as we in our delimitations stated, see 4.6, we do not plan to attack the Nest App, Web API nor cloud storage, and thus many attacks related to these areas were not investigated.

4.6 Initial delimitations

From the previous work regarding this area we have seen that most of the discovered vulnerabilities have been patched properly by examining the source code. Therefore we have decided that moving forward we will, of the previous attacks, only further investigate:
4.7 The system

In this section the main components and communication in the system are described.

4.7.1 Main components

In this section the main components in the Google Nest system used for our threat model are described.
- **Google Nest Indoor Camera**: The video recording is done by a camera connected to the local WiFi.

- **Cloud storage**: According to Google, the camera does not use a memory card to store the video. Instead it uploads the video continuously to the cloud [60].

- **Nest App**: The Nest App is used by the human user to access the video feed of the camera through the web API.

- **Web API**: To simplify the access to camera video feed, either stored in and or sent to the cloud storage, a web API is used [61].

### 4.7.2 Communication channels

Below follows a description of the different communication channels between the components in the Google Nest camera system. These communication channels are based on documentation of the camera and our findings while looking at the data transfer with Wireshark.

- **Bluetooth**: There is Bluetooth communication between the Nest App and the camera during the set up and authentication phase [62].

- **Hypertext Transfer Protocol Secure (HTTPS)**: The data sent between a user application and the web API is over HTTP using TLS. It also seems as if they use JavaScript Object Notation (JSON) and Representational state transfer (REST) according to slide 88 from a presentation held in 2018. [63]

- **IPSec**: The traffic from the camera to the cloud server is sent over a channel using TCP/TLS and in the Weave protocol they mention the Weave tunnel is using Encapsulating Security Payload (ESP) protocol of IPSec [64].

### 4.8 Threat report

In this sections follows a shorter version of threat report generated by Microsoft threat modelling tool based on the system shown in figure 4.3. It uses the STRIDE model presented in subsection 2.2. The full version can be seen in Appendix A.
The threats that were removed were deemed to be out of the scope of our thesis as they were in conflict with our delimitations as they were threats related to the Nest App, cloud server or Web API.

Table 4.1: Configurations tested

<table>
<thead>
<tr>
<th>ID</th>
<th>Threat</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Collision Attacks</td>
<td>Tampering</td>
<td>Attackers who can send a series of packets or messages may be able to overlap data. For example, packet 1 may be 100 bytes starting at offset 0. Packet 2 may be 100 bytes starting at offset 25. Packet 2 will overwrite 75 bytes of packet 1. Ensure you reassemble data before filtering it, and ensure you explicitly handle these sorts of cases.</td>
</tr>
<tr>
<td>2</td>
<td>Replay Attacks</td>
<td>Tampering</td>
<td>Packets or messages without sequence numbers or timestamps can be captured and replayed in a wide variety of ways. Implement or utilize an existing communication protocol that supports anti-replay techniques (investigate sequence numbers before timers) and strong integrity.</td>
</tr>
<tr>
<td>4</td>
<td>Weak Authentication Scheme</td>
<td>Information Disclosure</td>
<td>Custom authentication schemes are susceptible to common weaknesses such as weak credential change management, credential equivalence, easily guessable credentials, null credentials, downgrade authentication or a weak credential change management system. Consider the impact and potential mitigations for your custom authentication scheme.</td>
</tr>
<tr>
<td>8</td>
<td>Data Flow Bluetooth Is Potentially Interrupted</td>
<td>Denial Of Service</td>
<td>An external agent interrupts data flowing across a trust boundary in either direction.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>ID</th>
<th>Threat</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Weak Credential Transit</td>
<td>Information Disclosure</td>
<td>Credentials on the wire are often subject to sniffing by an attacker. Are the credentials re-usable/re-playable? Are credentials included in a message? For example, sending a zip file with the password in the email. Use strong cryptography for the transmission of credentials. Use the OS libraries if at all possible, and consider cryptographic algorithm agility, rather than hard-coding a choice.</td>
</tr>
<tr>
<td>14</td>
<td>Elevation by Changing the Execution Flow in Camera</td>
<td>Elevation Of Privilege</td>
<td>An attacker may pass data into Camera in order to change the flow of program execution within Camera to the attacker’s choosing.</td>
</tr>
<tr>
<td>16</td>
<td>Data Flow Bluetooth Is Potentially Interrupted</td>
<td>Denial Of Service</td>
<td>An external agent interrupts data flowing across a trust boundary in either direction.</td>
</tr>
<tr>
<td>17</td>
<td>Potential Process Crash or Stop for Camera</td>
<td>Denial Of Service</td>
<td>Camera crashes, halts, stops or runs slowly; in all cases violating an availability metric.</td>
</tr>
<tr>
<td>18</td>
<td>Weak Credential Transit</td>
<td>Information Disclosure</td>
<td>Credentials on the wire are often subject to sniffing by an attacker. Are the credentials re-usable/re-playable? Are credentials included in a message? For example, sending a zip file with the password in the email. Use strong cryptography for the transmission of credentials. Use the OS libraries if at all possible, and consider cryptographic algorithm agility, rather than hard-coding a choice.</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 4.1 – continued from previous page

<table>
<thead>
<tr>
<th>ID</th>
<th>Threat</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>Potential Data Repudiation by Camera</td>
<td>Repudiation</td>
<td>Camera claims that it did not receive data from a source outside the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.</td>
</tr>
<tr>
<td>20</td>
<td>Weak Authentication Scheme</td>
<td>Information Disclosure</td>
<td>Custom authentication schemes are susceptible to common weaknesses such as weak credential change management, credential equivalence, easily guessable credentials, null credentials, downgrade authentication or a weak credential change management system. Consider the impact and potential mitigations for your custom authentication scheme.</td>
</tr>
<tr>
<td>22</td>
<td>Collision Attacks</td>
<td>Tampering</td>
<td>Attackers who can send a series of packets or messages may be able to overlap data. For example, packet 1 may be 100 bytes starting at offset 0. Packet 2 may be 100 bytes starting at offset 25. Packet 2 will overwrite 75 bytes of packet 1. Ensure you reassemble data before filtering it, and ensure you explicitly handle these sorts of cases.</td>
</tr>
<tr>
<td>39</td>
<td>Elevation by Changing the Execution Flow in Camera</td>
<td>Elevation Of Privilege</td>
<td>An attacker may pass data into Camera in order to change the flow of program execution within Camera to the attacker’s choosing.</td>
</tr>
<tr>
<td>41</td>
<td>Spoofing of Source Data Store Cloud Storage</td>
<td>Spoofing</td>
<td>Cloud Storage may be spoofed by an attacker and this may lead to incorrect data delivered to Camera. Consider using a standard authentication mechanism to identify the source data store.</td>
</tr>
<tr>
<td>43</td>
<td>Data Store Inaccessible</td>
<td>Denial Of Service</td>
<td>An external agent prevents access to a data store on the other side of the trust boundary.</td>
</tr>
</tbody>
</table>

*Continued on next page*
<table>
<thead>
<tr>
<th>ID</th>
<th>Threat</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>44</td>
<td>Data Flow <strong>IPSec Is Potentially Interrupted</strong></td>
<td>Denial Of Service</td>
<td>An external agent interrupts data flowing across a trust boundary in either direction.</td>
</tr>
<tr>
<td>45</td>
<td>Potential Process Crash or Stop for Camera</td>
<td>Denial Of Service</td>
<td>Camera crashes, halts, stops or runs slowly; in all cases violating an availability metric.</td>
</tr>
<tr>
<td>46</td>
<td>Potential Data Repudiation by Camera</td>
<td>Repudiation</td>
<td>Camera claims that it did not receive data from a source outside the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.</td>
</tr>
<tr>
<td>47</td>
<td>Potential Excessive Resource Consumption for Camera or Cloud Storage</td>
<td>Denial Of Service</td>
<td>Does Camera or Cloud Storage take explicit steps to control resource consumption? Resource consumption attacks can be hard to deal with, and there are times that it makes sense to let the OS do the job. Be careful that your resource requests do not deadlock, and that they do timeout.</td>
</tr>
<tr>
<td>49</td>
<td>Spoofing of Destination Data Store Cloud Storage</td>
<td>Spoofing</td>
<td>Cloud Storage may be spoofed by an attacker and this may lead to data being written to the attacker’s target instead of Cloud Storage. Consider using a standard authentication mechanism to identify the destination data store.</td>
</tr>
<tr>
<td>50</td>
<td>Data Store Inaccessible</td>
<td>Denial Of Service</td>
<td>An external agent prevents access to a data store on the other side of the trust boundary.</td>
</tr>
<tr>
<td>51</td>
<td>Data Flow <strong>IPSec Is Potentially Interrupted</strong></td>
<td>Denial Of Service</td>
<td>An external agent interrupts data flowing across a trust boundary in either direction.</td>
</tr>
</tbody>
</table>
4.9 Threat Matrix

From all the attacks presented above we will now summarise those that we will continue with.

4.9.1 Compilation of threats

Below is the summarisation of threats presented in 4.1, 4.4 and 4.8 that were not removed due to our delimitations and or already patched vulnerabilities. To avoid superfluous material we put some threats under the same attack. The numbers, from 4.8, and or a label symbolising an attack described in 4.6, of those attacks are shown at the end of the attack box in each row.

If the attack requires that the threat agent was within Bluetooth range or on the network it was said that the agent was an unauthenticated internal agent. If not required the agent was marked as an unauthorised external agent.

Table 4.2: Threat traceability matrix

<table>
<thead>
<tr>
<th>Vector</th>
<th>Affected asset</th>
<th>Threat agent</th>
<th>Attack</th>
<th>Attack surface</th>
<th>Attack goal</th>
<th>Attack impact</th>
<th>Security controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bluetooth</td>
<td>Camera data</td>
<td>Unauth. internal user</td>
<td>Brute-force: Bluetooth encryption</td>
<td>Bluetooth link between assets</td>
<td>Break encryption</td>
<td>Access to confidential messages</td>
<td>Encrypt traffic</td>
</tr>
<tr>
<td>Software</td>
<td>Camera functionality</td>
<td>Unauth. external user</td>
<td>Exploration of outdated software</td>
<td>Camera’s software</td>
<td>Find weakness in software</td>
<td>Exploit the found weakness</td>
<td>Continuous updating of software</td>
</tr>
<tr>
<td>Bluetooth</td>
<td>Camera data</td>
<td>Unauth. internal user</td>
<td>Man-in-the-middle attack (10, 18)</td>
<td>Bluetooth links between assets</td>
<td>Intercept data</td>
<td>Access to confidential messages</td>
<td>Encrypt traffic</td>
</tr>
<tr>
<td>TCP/TLS</td>
<td>Camera data</td>
<td>Unauth. external user</td>
<td>Brute-force data encryption key</td>
<td>Data communication link</td>
<td>View camera feed</td>
<td>Sensitive data is compromised</td>
<td>Good enough encryption of the data sent</td>
</tr>
</tbody>
</table>

Continued on next page
Table 4.2 – continued from previous page

<table>
<thead>
<tr>
<th>Vector</th>
<th>Affected asset</th>
<th>Threat agent</th>
<th>Attack</th>
<th>Attack surface</th>
<th>Attack goal</th>
<th>Attack impact</th>
<th>Security controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network ports</td>
<td>Network ports services</td>
<td>Unauth. interal user</td>
<td>Port scanning</td>
<td>Network ports on camera</td>
<td>Find access point to camera</td>
<td>Extract useful information for a later directed attack</td>
<td>Closing unnecessary ports. Have listening daemon configured. Firewalls filtering out the SYN messages</td>
</tr>
<tr>
<td>Network ports</td>
<td>Camera’s functionality</td>
<td>Unauth. interal user</td>
<td>Fuzzing attack</td>
<td>Network ports on camera</td>
<td>Enable other attacks to take place</td>
<td>Disrupt camera functionality</td>
<td>Remove edge cases and handle all possible inputs</td>
</tr>
<tr>
<td>Weave</td>
<td>Camera’s functionality</td>
<td>Unauth. interal user</td>
<td>Buffer overflow</td>
<td>Buffers in the camera</td>
<td>Buffer overflow condition</td>
<td>Execute own code on the camera</td>
<td>Make sure inputted data cannot cause overflow and randomise address space of buffer</td>
</tr>
<tr>
<td>Network ports</td>
<td>Camera’s functionality</td>
<td>Unauth. interal user</td>
<td>TCP SYN Flood</td>
<td>Network ports on camera</td>
<td>Disrupt camera functionality</td>
<td>Lower timeout and or send back special packets</td>
<td>Try to verify that there is a real user</td>
</tr>
<tr>
<td>Network ports</td>
<td>Camera’s functionality</td>
<td>Unauth. interal user</td>
<td>ICMP attacks</td>
<td>Network ports on camera</td>
<td>DoS the camera</td>
<td>Disrupt camera functionality</td>
<td>Block large or fragmented ICMP packets</td>
</tr>
</tbody>
</table>

*Continued on next page*
<table>
<thead>
<tr>
<th>Vector</th>
<th>Affected asset</th>
<th>Threat agent</th>
<th>Attack</th>
<th>Attack surface</th>
<th>Attack goal</th>
<th>Attack impact</th>
<th>Security controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>Camera data</td>
<td>Unauth. external user</td>
<td>ARP Spoofing</td>
<td>Communication links between assets</td>
<td>Intercept data</td>
<td>Access to confidential messages</td>
<td>Static ARP table, encrypted com. link so the attacker must fake that too or filtering out packets from other networks</td>
</tr>
<tr>
<td>TCP</td>
<td>Camera data</td>
<td>Unauth. external user</td>
<td>DNS spoofing</td>
<td>Communication links between assets</td>
<td>Intercept data</td>
<td>Access to confidential messages</td>
<td>Active monitoring of DNS data and implement DNS security extension.</td>
</tr>
<tr>
<td>TCP</td>
<td>Access to camera and or data</td>
<td>Unauth. external user</td>
<td>IP Spoofing and HTTP requests (41, 49, D-link)</td>
<td>Communication links between assets</td>
<td>Trick camera into trusting you and enable other attacks</td>
<td>Access to confidential messages and or have camera execute commands</td>
<td>Encrypt traffic with an IPSec tunnel and or authenticate users</td>
</tr>
<tr>
<td>TCP</td>
<td>Camera data</td>
<td>Unauth. external user</td>
<td>Man-in-the-middle attack (Princeton)</td>
<td>Communication links between assets</td>
<td>Intercept data</td>
<td>Access to confidential messages</td>
<td>Encrypt traffic</td>
</tr>
<tr>
<td>TCP</td>
<td>Camera’s function-ality</td>
<td>Unauth. external user</td>
<td>Replay attack</td>
<td>Communication links between assets</td>
<td>Intercept data and send messages back</td>
<td>Change flow of messages and create unwanted behaviour</td>
<td>Add sequence number and or timestamps</td>
</tr>
</tbody>
</table>

Table 4.2 – continued from previous page
<table>
<thead>
<tr>
<th>Vector</th>
<th>Affected asset</th>
<th>Threat agent</th>
<th>Attack</th>
<th>Attack surface</th>
<th>Attack goal</th>
<th>Attack impact</th>
<th>Security controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bluetooth</td>
<td>Camera’s WiFi connection</td>
<td>Unauth. internal user</td>
<td>DoS attack (Doyle, 14)</td>
<td>Open Bluetooth connection</td>
<td>Change WiFi connection of the camera</td>
<td>Disable the camera</td>
<td>Turn off Bluetooth after setup</td>
</tr>
<tr>
<td>TCP</td>
<td>Camera’s connectivity</td>
<td>Unauth. internal user</td>
<td>TCP DoS attack (Cisco-Talos, 43, 44, 45, 47, 50, 51)</td>
<td>Camera’s TCP ports</td>
<td>Connect to all available TCP slots</td>
<td>Hinder connections to the camera and perhaps stop the connection to the cloud storage</td>
<td>Increase number of open slots and reduce number of connections from the same IP. The time until an idle connection is dropped can also be reduced.</td>
</tr>
<tr>
<td>Bluetooth</td>
<td>Camera’s connectivity</td>
<td>Unauth. internal user</td>
<td>Bluetooth DoS attack (8, 16, 17)</td>
<td>Bluetooth connection between Nest App and camera</td>
<td>Disturb Bluetooth connection during setup phase</td>
<td>Unable to setup the camera properly</td>
<td>It is hard to defend against [65]</td>
</tr>
<tr>
<td>Bluetooth</td>
<td>Camera’s functionality</td>
<td>Unauth. internal user</td>
<td>Repudiation (19)</td>
<td>Bluetooth connection between Nest App and camera</td>
<td>Manipulate data in camera</td>
<td>Camera starts believing data is not received properly</td>
<td>Logging, time-stamps and auditing of data</td>
</tr>
<tr>
<td>Bluetooth</td>
<td>Camera’s functionality</td>
<td>Unauth. internal user</td>
<td>Collision attack (1, 22)</td>
<td>Bluetooth connection between Nest App and camera</td>
<td>Make messages overlap</td>
<td>Cause the camera to behave in an unwanted way</td>
<td>Reassemble data before filtering and handle edge cases</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Vector</th>
<th>Affected asset</th>
<th>Threat agent</th>
<th>Attack</th>
<th>Attack surface</th>
<th>Attack goal</th>
<th>Attack impact</th>
<th>Security controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bluetooth</td>
<td>Communication integrity</td>
<td>Unauth. internal user</td>
<td>Replay</td>
<td>Communication links between assets</td>
<td>Intercept data and send messages back</td>
<td>Change flow of messages and create unwanted behaviour</td>
<td>Add sequence number and or timestamps</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>attack (2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP</td>
<td>Camera’s functionality</td>
<td>Unauth. internal user</td>
<td>Improper</td>
<td>Channel auth.</td>
<td>Compromise authentication scheme</td>
<td>Enable spoofing of data</td>
<td>Proper certificate’s chain of trust validation</td>
</tr>
<tr>
<td>certificates</td>
<td></td>
<td></td>
<td>validation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>of certificates</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP</td>
<td>Camera data transfer</td>
<td>Unauth. internal user</td>
<td>Repudiation</td>
<td>TCP connection between Nest App and camera</td>
<td>Manipulate data in camera</td>
<td>Camera starts believing data is not received properly</td>
<td>Logging, time-stamps and auditing of data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(46)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP</td>
<td></td>
<td></td>
<td>RST attack</td>
<td>TCP connection to the camera</td>
<td>DoS the camera</td>
<td>Send a RST segment and tear down the TCP connection and deny the camera the ability to send data.</td>
<td>If a RST message is received inside the correct window, send an ACK to the sender and make them re-send the RST message with the correct SEQ-number</td>
</tr>
</tbody>
</table>

Continued on next page
Table 4.2 – continued from previous page

<table>
<thead>
<tr>
<th>Vector</th>
<th>Affected asset</th>
<th>Threat agent</th>
<th>Attack</th>
<th>Attack surface</th>
<th>Attack goal</th>
<th>Attack impact</th>
<th>Security controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>Camera software</td>
<td>Unauth. internal user</td>
<td>Software update attack</td>
<td>Camera software</td>
<td>Change software in camera</td>
<td>A software update to the camera that does not contain the correct code can be dangerous for all camera functionality and its data</td>
<td>It is important to verify the integrity of the software update in order to secure that it has not been modified</td>
</tr>
</tbody>
</table>
42 | Threat model and Threat matrix
Chapter 5

Risk evaluation

This chapter expands on the threat matrix from 4.9. Section 5.1 provides a DREAD score for each attack and 5.2 then marks which attacks that were chosen to be investigated further based on this score.

5.1 Risk assessment

All of these attacks below assumes that the adversary can have access to the target’s network if required.

By convention the discoverability was set to a 3 for all threats [66]. The affected users was also set to 3 for all as of the threats found targeted the average camera owner.

According to Microsoft a score between 5-7 is considered Low risk, 8-11 is Medium risk, and 12-15 is High risk [10].

5.1.1 TCP SYN Flood attack

- **Damage (3):** Can potentially cause a crash in the camera.

- **Reproducibility (3):** It can be done whenever after the setup process has completed.

- **Exploitability (2):** As the attack is systematic a skilled programmer could follow a step by step guide.

5.1.2 Fuzzing attack

- **Damage (3):** Can potentially cause a crash in the camera.
- **Reproducibility (3):** Easily done by sending a variety of messages to the target.

- **Exploit-ability (2):** Requires open vulnerable ports to be open and that an attacker can connect to them.

### 5.1.3 IP spoofing and HTTP requests

- **Damage (3):** Previous work related to the D-link camera showed there were vulnerabilities in HTTP requests from specific IP-addresses. Exploiting this could lead to admin privileges on the camera. 4.5.1

- **Reproducibility (3):** The attack is executed after the camera has been setup correctly and started to send data.

- **Exploit-ability (2):** As with the previous two spoofing related attacks this is also a known one and hence guides exists on the internet.

### 5.1.4 RST attack

- **Damage (3):** By sending a RST message to the camera while pretending to be the cloud storage the possibility of bypassing security measures exists.

- **Reproducibility (3):** With all the requirements below it is simple to send a RST message to the camera.

- **Exploit-ability (2):** First you need to find the IP/MAC-address of the camera. Secondly the port used for the communication between the camera and the cloud storage is required which can be found using Wireshark if on the same network. Once done a RST message can be sent.

### 5.1.5 ICMP attack

- **Damage (3):** There is a possibility to overflow the camera and cause it to reboot.

- **Reproducibility (3):** After the TCP connection from the camera to the cloud server is completed this attack can be executed whenever.

- **Exploit-ability (2):** We believe someone with some programming expertise and a proper guide can go through with this attack.
5.1.6 Outdated software

- **Damage (3):** In the worst case scenario one of these vulnerabilities can enable the attacker to subvert the security system.

- **Reproducibility (2):** Difficult to give a score as it depends on the exploit itself.

- **Exploit-ability (2):** Same argument as above.

5.1.7 Software update attack

- **Damage (3):** If an attacker can compromise or upload their own code to the device it is serious.

- **Reproducibility (3):** A software update can arrive whenever to the camera so no timing window is required here.

- **Exploit-ability (1):** It takes a skilled hacker to create a software update that is able to alter the functionality of the camera.

5.1.8 Fake certificate attack

- **Damage (3):** Bypass the authentication

- **Reproducibility (2):** As the camera only exchanges certificates with the cloud storage during the beginning of their transmission there is a race situation.

- **Exploit-ability (2):** The attacker needs to be more than novice as they are required to create and send their own certificate to the targeted host.

5.1.9 Bluetooth WiFi disassociation

- **Damage (3):** Possible to through an arbitrary Bluetooth connection send harmful data to the camera. See 4.4.1.

- **Reproducibility (3):** If the Bluetooth connection is open all of the time then no timing window is required.

- **Exploit-ability (1):** The attacker must be within Bluetooth range of the device and most likely a tool such as gatttool is required as used in Jason Doyle’s work 4.4.1.
5.1.10 Man-in-the-middle privacy vulnerability

- **Damage (2):** Although the security of the camera is not compromised by this exploit the intended function of the camera, that is keeping your house safe, is still at risk. By extracting information of which state the camera is in, the attacker can make a more educated decision on how to proceed.

- **Reproducibility (3):** Just listen to the communication from the camera.

- **Exploit-ability (2):** Requires the attacker to be able to listen to the communication sent from the camera. This means either physically close or on the same network. Then a simple tool such as Wireshark is all that is necessary.

5.1.11 Buffer overflow

- **Damage (3):** With a buffer overflow condition it is possible to execute code as described in 4.1.2.

- **Reproducibility (2):** As the buffer overflow could be dependent on which state the camera is in a score of 2 was given.

- **Exploit-ability (2):** If a vulnerability is found then it would most likely be reproducible by following a guide.

5.1.12 Port scanning

- **Damage (1):** The damage caused by a port scan is not significant. It may lead to other attacks that are.

- **Reproducibility (3):** Easily reproduced using already existing tools.

- **Exploit-ability (2):** It is not hard to perform a port scan on a network.

5.1.13 ARP spoofing

- **Damage (1):** A successful attack will only redirect the traffic but as it is encrypted not much information would be available.

- **Reproducibility (3):** The attack does not require a timing window.

- **Exploit-ability (2):** As it is a fairly know attack there are guides online one could follow.
5.1.14 Replay attack over TCP

- **Damage (1):** Not much damage can be done as a replay attack would simply replay a video sequence, potentially showing the same frame twice.

- **Reproducibility (3):** No timing window required.

- **Exploit-ability (2):** As you must be able to record a message and then send it back it most definitely requires more than a novice programmer to complete this attack. Some listening program such as Wireshark will also be necessary.

5.1.15 TCP DoS attack

- **Damage (2):** The damage is not threatening the security but potentially other devices can be refused connection and the connection from the camera to the cloud storage can be severed.

- **Reproducibility (3):** The attack can be executed whenever.

- **Exploit-ability (1):** Most likely a network of computers would be required as to send several TCP SYN requests. Maintaining of these connections are then also necessary.

5.1.16 TCP repudiation

- **Damage (1):** This attack itself will not extract any useful information but can instead be used to cover an attackers track.

- **Reproducibility (3):** This attack would take place after the camera has been setup and thus we assume there is no timing window involved.

- **Exploit-ability (1):** This demands deep knowledge in how the camera works.

5.1.17 Replay attack over Bluetooth

- **Damage (2):** Replaying a sensitive message could result in sensitive information being leaked.

- **Reproducibility (2):** To resend a message sent over the Bluetooth link one must first capture one during the setup process.
• **Exploit-ability (1):** The attacker would have to be within Bluetooth range and have equipment to capture Bluetooth messages. It would also require skill in how to send Bluetooth messages.

### 5.1.18 Bluetooth DoS attack

- **Damage (2):** Could deny the pairing process between the Nest App and camera.

- **Reproducibility (2):** Can only be done during the Bluetooth exchange between Nest App and camera.

- **Exploit-ability (1):** Requires some equipment such as the Ubertooth One and some instructions. However the biggest requirements is that the attacker is within range of the camera during this startup stage.

### 5.1.19 DNS spoofing

- **Damage (1):** A successful DNS spoofing would likely be the first step in retrieving data. Alone however it will not be enough as the camera requires certificates and encryption keys before it starts sending data.

- **Reproducibility (2):** As the camera is only requesting the IP-address of the necessary servers during the setup process.

- **Exploit-ability (2):** As it is a public general attack we expect there to be guides one could follow to successfully use this attack.

### 5.1.20 Bluetooth collision attack

- **Damage (1):** Potentially overlapping data will by itself not do much.

- **Reproducibility (2):** Can only be done during Bluetooth setup and hence requires a timing window.

- **Exploit-ability (1):** This demands an extremely skilled programmer since it demands knowledge in creating packets and bit manipulation.

### 5.1.21 Bluetooth repudiation

- **Damage (1):** Can be used to hide evidence of other attacks but does not do much damage by itself.
• **Reproducibility (2):** Conducting the attack in a timing window is needed as the Bluetooth transmissions are only done during the setup phase.

• **Exploit-ability (1):** This demands deep knowledge in how the camera and Bluetooth works.

### 5.1.22 Man-in-the-middle Bluetooth

- **Damage (2):** If the attacker gains access to the credentials for the WiFi, by listening to the traffic, there is an obvious security flaw.

- **Reproducibility (1):** A clear timing window as the data is sent only during the setup. A normal user will also not have to do the pairing process more than once.

- **Exploit-ability (1):** You will most likely need a device such as Ubertooth One to make this attack possible. It is also necessary to be physically close to the device so that you can listen to the Bluetooth communication.

### 5.1.23 Brute-force Bluetooth encryption

- **Damage (2):** Leaking of sensitive information sent over Bluetooth.

- **Reproducibility (1):** A clear timing window as the data is sent only during the setup. A normal user will also not have to do the pairing process more than once.

- **Exploit-ability (1):** You will most likely need a device such as Ubertooth to make this attack possible. It is also necessary to be physically close to the device so that you can listen to the Bluetooth communication. In addition, you will have to break an encryption which is not a trivial task.

### 5.1.24 Brute-force data encryption over TCP

- **Damage (2):** If successful one could read the data sent over the link until a new key is negotiated.

- **Reproducibility (1):** It is generally hard to brute-force an encryption key [67].

- **Exploit-ability (1):** Extremely skilled to hack an encryption.
<table>
<thead>
<tr>
<th>Attack</th>
<th>Damage</th>
<th>Reproducibility</th>
<th>Exploitability</th>
<th>Affected users</th>
<th>Discoverability</th>
<th>Score</th>
<th>Risk level</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP SYN Flood attack</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>14</td>
<td>High</td>
</tr>
<tr>
<td>Fuzzing attack</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>14</td>
<td>High</td>
</tr>
<tr>
<td>IP spoofing and HTTP requests</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>14</td>
<td>High</td>
</tr>
<tr>
<td>RST attack</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>14</td>
<td>High</td>
</tr>
<tr>
<td>ICMP attack</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>14</td>
<td>High</td>
</tr>
<tr>
<td>Outdated software</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>13</td>
<td>High</td>
</tr>
<tr>
<td>Software update attack</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>13</td>
<td>High</td>
</tr>
<tr>
<td>Fake certificate attack</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>13</td>
<td>High</td>
</tr>
<tr>
<td>Bluetooth WiFi disassociation</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>13</td>
<td>High</td>
</tr>
<tr>
<td>Man-in-the-middle attack privacy vulnerability</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>13</td>
<td>High</td>
</tr>
<tr>
<td>Buffer overflow</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>13</td>
<td>High</td>
</tr>
<tr>
<td>Port scanning</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>12</td>
<td>High</td>
</tr>
<tr>
<td>ARP spoofing</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>12</td>
<td>High</td>
</tr>
<tr>
<td>Replay attack over TCP</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>12</td>
<td>High</td>
</tr>
<tr>
<td>TCP DoS attack</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>12</td>
<td>High</td>
</tr>
<tr>
<td>TCP repudiation</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>11</td>
<td>Medium</td>
</tr>
<tr>
<td>Replay attack over Bluetooth</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>11</td>
<td>Medium</td>
</tr>
</tbody>
</table>

Continued on next page
### 5.2 Decision of attacks to investigate

From the DREAD table above we decided to move forward with all the attacks that has a risk level of High as those are the most critical ones. The others were excluded except for the Bluetooth Man-in-the-middle attack. This was decided important as we wanted to learn more about how the communication was done and if there were any obvious security threats related to it.
Risk evaluation
Chapter 6

Penetration testing

In this chapter the penetration tests that were made are described.

6.1 ICMP attack - Ping of death

- INTRODUCTION: The camera was penetration tested for the ICMP attack ping of death.

- BACKGROUND: To execute this test the program Scapy was used on a Kali Linux Virtual Machine (VM) and a Windows 10 operating system [68]. In the Request for Comments (RFC) for ICMP messages it is stated that the maximum length of a packet is 65535 bytes [69].

- METHOD: Before this experiment was conducted the Windows computer, camera and Kali Linux VM, with a Mac as supervisor were all connected to the same network.

On the Kali Linux machine we attempted the attack in two different ways. The first was to send packets of a size larger than 65535 bytes and the second to send packets between 65,493 and 66,592 bytes. That range had worked previously according to a computer science project at the University of California Santa Barbara [70]. They reasoned that sending larger packets than this might just cause the network to discard them and hence we did not try to send packets outside of this range. The experiment was done using the following code snippets:

Test 1:

```python
from scapy.all import *
send(fragment(IP(dst="192.168.1.81")/ICMP()/("X"*66000)))
```
Test 2:

```python
from scapy.all import *
import time
for i in range(1099):
    data = 65493 + i
    send(fragment(IP(dst="192.168.1.81")/ICMP()/("X"*data)))
    time.sleep(1)
```

A sleep function was added as to try to make sure the camera had enough time to respond to the ICMP package so that it would not discard it if another package arrived. For the windows computer we conducted the same two tests.

- **RESULTS:**
  
  When we ran test 1 on the Kali Linux computer the first time the camera did actually reboot and we were unable to view the camera feed during that time period. However, the following 10 attempts were unable to reproduce this behaviour from the camera.

  Test 2 and both tests on the windows computer did not show any results as the camera was able to operate as usual while being pinged.

- **DISCUSSION:** There could be a weakness in the camera regarding the ICMP attack as it did in fact reboot when pinged the first time. Yet, as we were unable to replicate it we cannot prove that there exists a definite vulnerability. There are after all many plausible scenarios which can force the camera to reboot as a simple software update or setting change can force the same behavior.

### 6.2 ICMP attack - Flood attack

- **INTRODUCTION:** We attempted to perform a flooding attack using the resources we had.

- **BACKGROUND:** A flooding attack is usually done with access to hundreds or thousands of computers. As all we had was two computers we tried to make them perform at a high capacity as to simulate a potential flooding attack.

- **METHOD:** The two computers and camera were connected to the same network.
On the Windows computer 100 terminal windows were opened and they all ran the same command:

```
ping 192.168.1.81 -t -l 65500
```

192.168.1.81 was the IP-address of the camera and 65500 the maximum allowed length of ping messages from a Windows.

We then attempted to ping the camera from a Mac with just one terminal window.

- **RESULTS:** The result found was that the camera sent its video feed as usual but the ping messages from the Mac computer, which usually were answered quickly, now received a response after several seconds or a timeout.

- **DISCUSSION:** It seems as if the camera, when receiving too many ping messages starts dropping them and thus has some sort of resource handler in place. If this was not the case we should have seen the camera crash or reboot once flooded with this large amount of packages.

### 6.3 Port scanning

- **INTRODUCTION:** In this penetration test we did a vertical port scan on the camera.

- **BACKGROUND:** We used the nmap program to conduct our tests. The documentation for it can be found on the official nmap website [71].

- **METHOD:** To begin the test we created a Kali Linux VM on a NAT:ed network. The hypervisor, in this case a Mac, was on the same network as the camera. In the terminal we then used the command in which 192.168.101.112 is the IP-address of the camera.

```
$ nmap -sS 192.168.101.112 -p 0-65535
```

The -sS flag is to tell nmap that we want to do a TCP SYN scan. The -p 0-65535 notifies the program that we want it to scan all of the ports on the entire camera.

Later the same code was executed on a Windows computer on the same network as the camera.

These tests were done with the email notifications turned off and on.
A Wireshark was also running to capture the traffic from and to the camera.

- **RESULTS:** The port scan from the Kali Linux VM gave us the following result with email notifications turned off and on:

```
<table>
<thead>
<tr>
<th>PORT</th>
<th>STATE</th>
<th>SERVICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>80/tcp</td>
<td>open</td>
<td>http</td>
</tr>
<tr>
<td>110/tcp</td>
<td>open</td>
<td>pop3</td>
</tr>
<tr>
<td>143/tcp</td>
<td>open</td>
<td>imap</td>
</tr>
<tr>
<td>993/tcp</td>
<td>open</td>
<td>imaps</td>
</tr>
<tr>
<td>995/tcp</td>
<td>open</td>
<td>pop3s</td>
</tr>
</tbody>
</table>
```

The answer from the Windows computer did to the contrary state that all 65535 ports on the camera were closed.

In both cases when the camera received a TCP SYN message from nmap it also said that it sent a RST message back.

- **DISCUSSION:** Initially after running the port scan from the Kali Linux operating system we believed the ports were open. This made us think that we would be able to conduct some of the other penetration tests we had planned. Furthermore, after running the port scan on the Windows computer and attempting to connect to the camera on port 80 unsuccessfully we realised that the ports were indeed closed. The RST message sent back from the camera supports this theory. This meant that the penetration tests below were scrapped as they all required one to be able to connect to the camera.

We had also hoped that with the email notifications enabled, on the Nest App, some necessary ports would open up. The results however show that this is not the case and therefore we can only assume the messages are instead sent through the same Weave tunnel as everything else.

**Fuzzing attack**

**TCP SYN Flood attack**

**IP Spoofing HTTP requests**

**TCP DoS attack**

**TCP Buffer overflow**
6.4 ARP-spoofing

- **INTRODUCTION:** An ARP-spoofing attack aims to change the MAC-address associated with an IP-address so that communication is sent through the hackers computer.

- **BACKGROUND:** This penetration test was done using the program ettercap [72]. With this program it is possible to execute an ARP-poisoning attack which is another name for an ARP-spoofing attack. In a second trial another program called bettercap was also used [73].

- **METHOD:**
  
  First attempt: At setup the camera and a Kali Linux (VM) were set to run on the same network. The VM was set to bridge mode. To then make sure packages were forwarded correctly the two following lines were typed into a terminal window in the root folder:

  ```
  sysctl w met_ipv4.ip_forward = 1
  echo 0 > /proc/sys/net/ipv4/conf/eth0/use_tempaddr
  ```

  Ettercap, the non-graphical version, was then opened, at sniffing on startup and the eth0 interface selected followed by clicking the check box. The IP-address of the camera was added using the host scan function. Then the router’s IP and camera’s IP were added to target 1 and target 2 respectively. To launch the attack the ARP-poisoning option was then selected under the MITM-menu.

  Second attempt: In the second attempt bettercap was used on a Mac. The Mac was connected to the same WiFi as the camera. The following command were used to initiate the attack:

  ```
  set arp_spoof.targets 192.168.1.106, (cont.) 192.168.1.1; arp_spoof on
  ```

  192.168.1.106 was the camera’s IP and 192.168.1.1 the router’s IP.

- **RESULTS:**

  First attempt: We were unable to extract any data being sent from the camera to the router. Instead the camera feed was interrupted after the death of the camera.

  Second attempt: Bettercap did not show any results as there were no
intercepted transmissions captured in Wireshark after the command was executed.

- **DISCUSSION:** We are not certain but we believe the results from the first attempt was due to the VM using improper IP-forwarding. Without a stable connection to the Weave server the camera could simply decide to terminate its processes as it has no destination to send its data to. The results from the second attempt are also strange as according to their bettercap’s website the command should be enough to successfully start an ARP-spoofing attack [74]. Plausible reasons for this outcome could be that the program itself is faulty or that some external parameters were not setup properly.

Nonetheless, even if our ARP-spoof attempts were unsuccessful we believe the attack does not pose any real threat. This assessment was made after discussions with our supervisor and other students as they did not believe this was dangerous. A correctly executed attack would alone simply result in similar results as 6.5, shown directly below.

### 6.5 Man-in-the-middle privacy vulnerability

- **INTRODUCTION:** The attack vector to be explored is a type of man-in-the-middle attack. By listening to the traffic sent and received by the camera an adversary could according to prior findings perhaps learn in which state the camera is, see 4.4.2.

- **BACKGROUND:** To intercept the traffic from and to the camera the program Wireshark was used [75].

- **METHOD:** Before the penetration test was conducted the devices were set up properly.

  Firstly, the camera was placed so close to the computer screen the video stream from it only showed the screen. Secondly, the sample video was paused at 00:00 showing a pitch black picture [76]. Thirdly, the Mac was configured so that it was listening to the wireless communication through monitor mode in Wireshark using its WiFi interface. Fourthly, using the Nest App, the settings on the camera were configured accordingly to the test conducted. In test 1 the microphone on the camera was turned off. In test 2 it was turned on and in test 3 the sound notifications were also enabled. Lastly, the Google Nest App, without Nest aware,
a subscription option within the App, on an iPhone was running in the background.

All the tests were conducted in one session without the camera being moved and with the same sound levels on the equipment.

The experiment began with 30 seconds of silence to record the camera in its initial state. At the 30 second mark the App was opened so that the video stream could be viewed on the iPhone. After another 30 seconds had passed the play button on the video was pressed. Following this the video was paused, set back to 00:00 and the App minimised, in that order. Another 30 seconds went by and then the video was started once again and played for 30 seconds. After this the video was paused, showing a still image, and after 30 seconds the Wireshark transmission was stopped and downloaded.

To produce the graphs of the data the build in Input/Output (I/O) graph tool in Wireshark was used.

- RESULTS:
  - In Figure 6.1 the results from test 1 is presented.
  - In Figure 6.2 the results from test 2 is presented.
  - In Figure 6.3 the results from test 3 is presented.

![Figure 6.1: Bytes/second captured during test 1](image)
• DISCUSSION: We believe there are 4 distinct levels of transmission. Level 1 where there is basically no activity. This can be seen in all tests from 0-30 seconds, 115-120 and 155-180. During these phases the camera sends minimal data only to make sure it is still connected to the Nestlabs server. Level 2 is between 120-155. During this period there was movement in front of the camera but the video was not being streamed, only an increased amount of notification messages. Level 3 is
between 30-60 and 90-110. Here we have increased data transmission due to the user watching the video stream on the application but the picture is not changing. Level 4 is between 60-90. Here the stream is both being viewed by the user and the content on the stream is changing. Level 1, 2 and 3 mirrors the results from the previous study done by Princeton, see 4.4.2.

An important aspect is that there can be some variability between when the video is played or paused and the camera feed opened or closed. Although the human interaction could cause damage to the reliability of the test we strongly believe there is no such danger here. Even if the camera feed between the tests were opened 1 or 2 milliseconds earlier or later than intended, the difference in data bytes sent is still clearly visible. We can therefore with great certainty say that this did not affect the result.

It is easy to establish in what mode the device and user is by just looking at the amount of data that is being sent over the network. We would say this is a privacy concern for the user that someone outside your house monitoring your WiFi can establish if you are awake and moving around inside your house, or maybe visiting some special room the attacker know you have the camera in.

The fact that an attacker could know when a user is actively viewing the camera could be sensitive if for example an intruder is waiting for the user to stop monitoring the camera before breaking into the house. Level 4, that shows if movement is detected while the user is monitoring the stream, is also interesting for a physical intruder since they can know if they have been detected or not by the camera. This is something Princeton did not explore in their work and thus new findings [37].

### 6.6 Bluetooth Man-in-the-middle attack

- **INTRODUCTION:** We explore if it is possible to extract any useful information by listening to the Bluetooth communication.

- **BACKGROUND:** We used the Ubertooth One device for this attack [77]. It is a wireless platform that can be used to sniff communication sent over Bluetooth. In addition Wireshark was used in combination with this to easier visualise the output.
• **METHOD:** The Ubertooth One device was connected to the MacBook laptop. A first in first out pipe /tmp/pipe was created so Wireshark knew where to listen to the traffic. To capture traffic with Ubertooth One following command was used:

\[ \text{ubertooth-btle -f -t <MAC:address of camera> -c /tmp/pipe} \]

The traffic was then analysed in Wireshark.

• **RESULTS:** We were able to intercept messages and see when a transaction of WiFi information, used to connect the camera, took place. It was possible to see to which network it was connecting to but the password for that WiFi was encrypted. Moreover, the camera was, once connected to the WiFi, sending broadcast messages.

• **DISCUSSION:** Even though no directly potential harmful information was extracted it is still important to notice a couple of things. If an attacker is present during the pairing phase they can learn to which WiFi the camera is connecting. It is in addition also possible for an attacker to see if a Nest camera is within Bluetooth range as it keeps sending these broadcasts messages even after the setup.

### 6.7 Disabling Bluetooth connect-ability

• **INTRODUCTION:** Our goal was to mimic the third vulnerability presented by Jason Doyle in 4.4.1. However due to a lack in equipment, at the time of the test, we could not get gatttool, a necessary program for the attack to work. Instead we then downloaded the application Bluetility as to attempt to connect to the camera and stumbled upon unexpected results.

• **BACKGROUND:** Bluetility is a Bluetooth Low-Energy utility for Mac OS X. It can among many things be used to connect to devices [78].

• **METHOD:** The penetration test was done on the camera when it was in two different stages, before paired and after paired. The computer was also in Bluetooth range of the camera.

The attempt was done through a Mac and was done by opening the Bluetility application. This was followed by the refresh button being pressed so that the App could find all the nearby devices. Once the Nest
Once this had been completed a Nest App user then attempted to connect to the camera by scanning the QR-code.

• RESULTS: This resulted in the Bluetooth connection, available on the Nest camera, becoming occupied by Bluetility. It was possible to connect to Bluetooth on the camera before the user during the setup phase. This resulted in the user being unable to connect and provide the camera with access to the WiFi. Figure 6.4 shows the screen an App user was presented with after attempting to change the WiFi connection. If the setup phase had already been completed an adversary could still connect to the camera’s Bluetooth with the help of Bluetility. As long as they then remained connected to the camera the user was unable to
change the WiFi the camera is connected to.

- **DISCUSSION**: This is in agreement with what we found in the OpenWeave github code [79]. It seems that the process is halted waiting for input from a user, and thus preventing others from using the connection. Due to this weakness the attacker has the ability to prevent a user from having full control over their device. If the attacker is present during the setup phase it has the ability to prevent any usage of the camera. This nullifies the entire functionality of the camera. Furthermore, it could give the attacker extra time break into the area the camera is supposed to monitor.

If the attack is conducted after the camera has been setup it can still prevent some functionality by hindering the user from changing the WiFi the camera is connected to. Say that a particular WiFi has some weakness, then the attacker can force the camera to stay on that WiFi and maybe take advantage of said weakness.

In conclusion, our DREAD score for the Bluetooth DoS attack was obviously an underestimate. We did not expect it would be possible to deny a connection from a legitimate user after the pairing had been done but this penetration test shows this is the case.

### 6.8 Bluetooth Buffer overflow

- **INTRODUCTION**: This penetration test was done to explore if the buffer overflow vulnerabilities Jason Doyle found had been patched or not. More information about these can be found in 4.4.1.

- **BACKGROUND**: To conduct these experiments we used a virtual Ubuntu Linux machine running on a Windows 10 computer. The Delock USB Bluetooth 4.0 USB-Adapter was used for communicating with the camera. The programs `gatttool` and Host Control Interface Tool (`hcitool`) were also used to find the camera’s Bluetooth address, connect to it and send it data.

- **METHOD**: To first find the camera the following command was used:

  ```
  $ hcitool lescan
  LE Scan ...
  18:B4:30:DF:1E:30 Nest Cam
  
  Once the address was acquired gatttool was used to connect to the camera.
  ```
Once this was completed the write requests were sent to the camera. During this the live stream on the camera was also investigated to see if the camera would crash and reboot.

Test 1:

```
[18:B4:30:5D:00:B8][LE]> char-write-req 0xfffd 3a03120545534d31a10160335efa15e4b2a9868f22f122382eabb20022800323610de21ecaeb47b7da4afec1eb8bc34ce663a203750b9637dde3ad0ec6e895f7fa60f42ae559f4d
```

Test 2:

```
[18:B4:30:5D:00:B8][LE]> char-write-req 0xfffd 3a03120545534d31a10160335efa15e4b2a9868f22f122382eabb20022800323610de21ecaeb47b7da4afec1eb8bc34ce663a203750b9637dde3ad0ec6e895f7fa60f42ae559f4d
```

- **RESULTS:** The result from both tests were that the camera did not crash and reboot but instead remained online the entire time.

- **DISCUSSION:** The results seem to be inline with the expected outcome if one trusted Jason Doyle’s reported patching.

### 6.9 Bluetooth WiFi disassociation

- **INTRODUCTION:** This penetration test was aimed to check if the WiFi disassociation vulnerability Jason Doyle found had been patched or not.

- **BACKGROUND:** See the background presented in 6.8 for information regarding hcitool and gatttool. The program Bluetility was used to write data to the camera. Ubertooth One was also used to view the transactions of data and collect necessary data.

- **METHOD:** To begin this penetration test a real WiFi transmission was collected using Ubertooth One. The hexcode for this transmission was then used to see if the camera would connect to a legitimate WiFi. The hexcode acquired was:

  3403120c54656c69612d4545364535331a10160335efa15e4b2a9868f22f122382eabb20022800323610de21ecaeb47b7da4afec1eb8bc34ce663a203750b9637dde3ad0ec6e895f7fa60f42ae559f4d
Furthermore, Jason Doyle’s own hexcode was also used which was used to see if the camera would crash given a false WiFi SSID and password:

3a03120b0a6574536d6172742d356e1a20232323233a03120b0a6574536d6172742d356e1a202323233a03120b0a6574536d6172742d356e1a20232323
3becb824ba437c13233ac2ff78b1776456e47a01
3ca5787d2f5e53f394a512200228003210bc9253
3d48cad7a0d921d57b2d26ae89c3a04DEADBEEF
3e

With the first part completed it was then time to provide the camera with a fake and a real SSID and password to see if it would cause the camera to crash or swap network. The test proceeded in two different ways.

The first write attempt was with the aid of Bluetility, which has an added write functionality. The second attempt was with gatttool which Jason Doyle’s used himself for the proof of concept attack. After each write the camera’s behaviour was observed using the App and Wireshark.

Test 1: After using Bluetility to connect to the camera, using the same steps as in 6.7, Jason Doyle’s hexcode and a legitimate WiFi hexcode was sent to the camera through Bluetility’s write function. Here the entire hexcode was sent at once as Bluetility supported the transmissions of several messages at once.

Test 2: To begin hcitool and gatttool was used to find and connect to the camera.

```
$ hcitool lescan
LE Scan ...
18:B4:30:DF:1E:30 Nest Cam
```

```
$ gatttool -b 18:B4:30:DF:1E:30 -t random -I
[18:B4:30:DF:1E:30][LE]> connect
Attempting to connect to 18:B4:30:DF:1E:30
Connection successful
```

This was then followed by write requests, one with the fake SSID and one with a real SSID.

```
[18:B4:30:5D:00:88][LE]> char-write-req 0xfffd <hexcode>
```

The <hexcode> section was then replaced with a line of hexcode. The same command was then used until all hexcode lines had been written for either the fake or the real SSID.
• RESULTS: Out of the four total write requests the camera did not crash or reconnect to a different WiFi.

• DISCUSSION: The result is in agreement with the statements from Jason Doyle and thus it seems as if the vulnerabilities have been patched properly.
Penetration testing
Chapter 7

Researched but not tested

In this chapter the results of the attacks that were not penetration tested, but still researched, are shown.

7.1 RST Attack

According to RFC793 (TCP) "In all states except SYN-SENT, all RST segments are validated by checking their SEQ fields. A reset is valid if its sequence number is in the window." [80]. This is the reason for the feasible RST attack. Since the window size is large it is "easy" to find a SEQ-number that matches that range.

Google Nest has already solved this by violating RFC793 and implement a check that requires the sender to re-send the RST packet with matching sequence number even if it is inside the window [81]. This forces the attacker to guess the exact sequence number instead of just the one inside the window.

7.2 Software update attack

Google Nest makes sure that the camera is updated automatically whenever there is a new update available [82]. To avoid the possibility of a software update attack every software update for the camera has its integrity checked. Thanks to this the camera knows the update has not been tampered with and the camera can trust that the update is from a trusted source. The integrity check is done by comparing the Secure Hash Algorithm (SHA) 160, SHA256 or SHA512 hash of the update, with the integrity value sent in the update message. If these match, the camera is updated. [83]
7.3 Replay attack

To avoid replay attacks it is important for messages to be unique so the software does not get fooled by an older message. Google Nest has implemented a new pseudo random initial sequence number for each new TCP connection that is initiated [84]. This makes each TCP segment unique and if an attacker replays a message that message will be discarded.

7.4 Fake Certificate attack

The pairing process between the Nest App and the Google Nest Indoor Camera starts over Bluetooth. Weave uses the QR-code printed on the camera itself as a way to authenticate that user is actually in possession of the camera and initiate something Weave calls PASE [6]. During this PASE stage, between the Nest App and the Camera, a session-key is initiated which is later used for secure communication between the devices, see method GenerateInitiationStep1, [85]. The Nest App contacts a Nest server to obtain the pairing token, which is transmitted to the camera through PASE [6]. The camera uses CASE to contact the Nest server and uses the pairing token to be associated with the users account [6]. During this process the signature of the certificate is verified, see method VerifySignature, where the signature of the certificates are verified with the help of the built in hardware certificates in the device [86].

7.5 Outdated software

When doing the man-in-the-middle privacy vulnerability test an interesting observation was made. Wireshark, which was used to intercept the packages, noted that the packages sent from the camera was done with TLSv1.0. Packages received by user from the cloud however was done with TLSv1.2. This is thought-provoking as it shows a discrepancy between different communication links. IETF also went out in 2018 and wrote that TLSv1.0 is to not be used as it lacks in some security aspects and is less secure than the TLS versions that succeeded it [87].

7.6 Injection

Injecting anything into the camera is not easy. The connection to the cloud is with TLS which makes it nearly impossible to inject packets that is not from
the trusted source. The camera only receive commands during the initial setup phase with the cloud which as said previously is using certificates to verify the information is trusted.

The only commands sent over Bluetooth is the pairing phase, PASE phase mention in 2.2. During this session establishment none of the information is executed and only verified with different certificates et cetera [85].

7.7 Broken Authentication

The connection between the camera and the cloud is authenticated with the help of certificate verification, as mentioned in 7.4. The 6-digit code is used for authentication of the Bluetooth pairing process. All this would lead to that all traffic is properly authenticated.

7.8 Sensitive Data Exposure

The data sent over IPSec is encrypted in the Weave tunnel with Advanced Encryption Standard (AES)-128 which would make a sufficient protection of all the data. The Bluetooth traffic is also encrypted with AES-128 as seen both by looking at the messages and reviewing the code of the PASE process [85].
72 | Researched but not tested
Chapter 8

Discussion and Conclusions

In this chapter we discuss and draw conclusions from our findings. Section 8.1 describes the overall security over the Bluetooth communication link. Section 8.2 does the same but for the TCP link. The attacks that are not directly connected to Bluetooth or TCP are covered in 8.3. The future work is discussed in 8.5 which is followed by the overall conclusion in 8.4.

8.1 Bluetooth security

8.1.1 Hardcoded password

Google Nest Indoor Camera uses a fixed 6 digit pairing code for the pairing phase. This falls under OWASP’s top 10 hardcoded passwords vulnerability. Brute forcing a 6 digit code is not impossible but as mentioned in 4.4.3 one is only able to guess 3 different pairing codes every 15 seconds. This would on average take:

\[ 3^5 = 243 \]

which is the number of possible combinations of the pairing code. The time in seconds to try all combinations would thus be

\[ \frac{3^5 \times 3}{15} = 167772160 \]

Which on average would take

\[ \frac{3^5 \times 3 \times 15}{2} = 83886080 \]

seconds or 2.66 years.
This sort of brute force time we would argue is good enough for this type of camera. It is rather unlikely one would be able to stay within range of this camera for that long without being discovered. Yet, as the code cannot be changed there still exists potential vulnerabilities. Once found, either through brute-force or by physically looking at the device, it can hence be used for the entire life-time of that product.

The conclusion for this step would be that the brute force-ability of the pairing code is secure enough but it would still be an improvement for the overall security of the device if this code was changeable by the user.

8.1.2 PASE

In the pairing process PASE is used to exchange a session key between the camera and Nest App. This code is used to prove that it is indeed the owner of the camera that is authenticating themselves with the camera. As long as this code has not been obtained by an adversary the authentication process of the user is valid. Due to this we conclude that the user is authenticated with the camera and that an attacker without the proper 6-digit code cannot authenticate with the camera and get full access.

8.1.3 Additional aspects

Since the user and the camera establish a session key and encrypt the traffic, the MITM becomes negated since the attacker will only see random data. The only potentially valuable data that we were able to extract was the WiFi SSID the camera connected to. This does nonetheless not open up any serious risks by itself.

The more interesting discovery we made in 6.7 was that it was possible to deny someone from connecting to the camera completely. This is certainly a vulnerability an owner of a camera should be informed about even if it does not directly give the adversary access to, for example, the camera stream.

8.2 TCP security

8.2.1 Pairing and certificates

Continuing on the topic of pairing from 8.1.2 the camera contacts the Nest server and provides the pairing token to the camera received from the Nest App, see 7.4. The connection between the camera and cloud uses the built in
certificates provided to the camera during manufacturing. This means that the camera can verify the signature provided by the Nest cloud and be convinced that it is communicating with the server and not an attacker. Our conclusion is that it would not be possible to intercept the connection from the camera and trick it to believe that the certificate provided by the attacker is the valid one. The camera and cloud server negotiates a session key which all the traffic is encrypted with. All this would protect the camera from both fake certificate attacks, and if the attacker is trying a MITM attack it will only see random data, which means the data is protected.

8.2.2 Man-in-the-middle privacy vulnerability

However we have discovered, based on the work from Princeton, that there is some information leakage when monitoring the traffic. Even when only looking at the encrypted data it is still possible to establish which mode the camera is in and if there is any activity in front of the camera. Based on our results we have discovered that the same information leakage described earlier by Princeton still exists. An attacker could exploit this information leakage to establish an understanding of the user’s behaviour.

For example, it is possible to know when there is movement in front of the camera. If a camera is monitoring a room in the house and motion is detected by it, a burglar catching the traffic can learn that someone or something is in the house. They could also see for how long and often a homeowner is watching their video stream. All of this together enables them to make a more educated decision whether to break into the house or not.

Furthermore, since the amount of data transmitted increases a lot when the user is watching the video stream the attacker can establish how long it takes for the user to start watching the stream after movement has been noticed by the camera, and for how long the user normally watches the stream.

What our research found that was not mentioned by Princeton previously was that it was possible to see a fourth level of data transfer. It was a distinct difference between the amount of data sent when someone was watching the video stream with and without movement in front of the lens. This means that an attacker can at all times establish when someone is watching and the camera has captured any motion.

The number of different ways this can be exploited is only limited by the creativity of the attacker. We believe that this is not the most severe vulnerability but still something the user should be aware of.
8.2.3 Port scan and other TCP attacks

It was a bit worrying that we got contradicting results when we did the port scan on the MacBook and on the Windows computer. But with our multiple scans with different programs and monitoring packets in Wireshark we are convinced that all ports on the camera are closed.

With all the ports being closed the camera only send out RST packets as a response to all attempts of establishing a connection. Our conclusion of this is that the camera is not vulnerable to fuzzing attacks since it does not matter what is sent as we still only get RST back. The TCP SYN flood attack is also not a threat as without an established connection the camera does not consume much processing power to send a RST packet. This also means that the camera has protection against the IP spoof attack as well as it does not matter where the connection comes from as the ports will still only send a RST back. The TCP DoS attack also becomes a non-threat as every connection request is replied with by a RST message.

The RST attack was fixed by a modification of the TCP protocol and we classify this implementation as secure against that sort of attack.

8.3 Remaining attacks

In this section we present the additional security results that are not connected to the Bluetooth or TCP communication.

8.3.1 ICMP attacks

We have looked at two attacks that use ICMP. The Ping of Death attack did not do much against the Google Nest Indoor Camera. This did not really come as a surprise since the Ping of Death attack seems to be an older attack that is often solved in newer and updated systems. In conclusion, the Google Nest Indoor Camera was able to handle IP fragmentation without any errors occurring. The flood attack was successfully in one way. Sending several pings from one computer made pings from other computers take longer than usual to get answered. Still, this did not impact the functionality of the camera so we concluded that the camera has a functioning resource management handler and is hence safe from this attack.
8.3.2 Software update

Due to the fact that Google Nest Weave protocol has implemented an integrity check for all updates means that an update that has been modified or not is not from Google will not be implemented by the camera. This protects it from all the different software update attacks. Thus we would say that this vulnerability can not be exploited.

8.3.3 ARP spoofing

The ARP spoofing we attempted resulted in the data from the camera ending up at our computer instead of being forwarded to the router. This lead to the camera loosing contact with the Nest server and the user was not able to watch the stream. We would say this happened because the ARP table in the camera was changed due to our ARP reply tricking the camera.

To conclude, although this threat was found through a non-successful ARP spoofing attack it still shows it is possible to bring down the camera effectively. If the attacker is able to gain access to the network the possibility of this exists and is thus something an owner of a camera should be aware of according to us.

One mitigation option we have thought of would be to have a static ARP entry to the router that would be initiated during the setup phase. Since most people do not change IP address to their router often it should not lead to too many problems and the worst case would be that they would have to redo the pairing process with the network. The static ARP entry would make the camera ignore other ARP replies that pretends to be the router.

8.4 Conclusion

So to complete our thesis report we want to answer our initial question: How secure is the camera?

Clearly it is not a perfect system. From the MITM privacy vulnerability to the outdated TLS version improvements can definitely be made to the camera’s security. However the existing vulnerabilities allow the owner to monitor their own inside of their house without having to be worried that the camera feed will be leaked. We have also not found any evidence that shows that the camera can be used to access other parts of the network.

Nevertheless, the vulnerabilities that have been discovered should not be taken lightly. What on the surface could be seen as a harmless bug might turn
out to be the gateway to further exploitation of the system.

8.5 Future work

Due to the breadth of the problem and our delimitations we have not been able to cover all aspects of security in this thesis report. In this section we describe what has yet to be done.

8.5.1 6-digit pairing code risk

As stated in 8.1.1 the pairing code was deemed to be safe from a brute-force attack. Yet, if the code was to be found Cisco-Talos wrote that an attacker that authenticates with the camera might be able to send certain messages to the camera and thus possibly take control of it [39]. It could be of interest to look at what the possibilities are once one has discovered this code.

8.5.2 Medium threats

Due to our time limitation we did not have time to investigate all the possible vulnerabilities we found. In the future we would hence like to look at the threats that by DREAD got a medium score, apart from the Bluetooth MITM attack which we looked into.

8.5.3 ICMP crash scenario

During our ICMP Ping of Death attack, shown in 6.1, the camera did crash during the first attempt. We could not make this happen consistently and thus this is something we would love to see someone else investigate. Was it dumb luck or did we actually find something?

8.5.4 Talos-Cisco TCP DoS attack

Talos-Cisco discovered among many vulnerabilities that an adversary could connect to all TCP ports and thus hindering others from connecting to the camera. The patch for this did solve this in some ways but as can be read in 4.4.3 we did not believe it was completely safe. When we then wanted to test this it turned out we were unable to connect to any of the TCP ports on the camera as they were closed. We therefore wonder if this was a late response to address this vulnerability or if we have missed something regarding the
ports. Future work could therefore look into this further to see if it is still a vulnerability.

8.5.5 Buffer overflow attacks

Although we did not look at the buffer overflow attacks there could without a doubt be some out there. This would most likely require a deep dive in the code but if someone has the energy to do so perhaps a new vulnerability can be discovered.

8.5.6 TLSv1.0

From 7.5 we learned that the camera is using an outdated version of TLS when sending data from itself to the cloud server. We did however not pursue this lead but instead leave it for future work.

8.5.7 Delimitations

Lastly we would of course encourage future work to cover areas such as the cloud, web API, the connection to other Nest devices, and the hardware which were things we decided to not look into. One thing could be to attempt to extract firmware as that is perhaps a place where you would like to start in an attempt to better understand the camera.

8.6 Reflection

Our methodology used in this thesis could perhaps have been improved by introducing a more standardised error handling. This was something we realised after our paper was peer-reviewed.

As stated in section 1.6 this project has been beneficial to the Sustainable and development goal 16 as we help protect fundamental freedoms. However, with our conclusion there exists a potential risk that we might affect some on these goals negatively as people might be inclined to purchase this camera. With more purchases of cameras there is a danger that the goals 7, 12 and 13 might be harder to reach due to the demand of more materials and energy.
Discussion and Conclusions
References


Available: https://github.com/openweave/openweave-core/commit/c5d1aebo8e76bf070403b21305c8b976df0f685


## Appendix A

### Threat Model Tool Matrix

Table A.1: Configurations tested

<table>
<thead>
<tr>
<th>ID</th>
<th>Threat</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Collision Attacks</td>
<td>Tampering</td>
<td>Attackers who can send a series of packets or messages may be able to overlap data. For example, packet 1 may be 100 bytes starting at offset 0. Packet 2 may be 100 bytes starting at offset 25. Packet 2 will overwrite 75 bytes of packet 1. Ensure you reassemble data before filtering it, and ensure you explicitly handle these sorts of cases.</td>
</tr>
<tr>
<td>2</td>
<td>Replay Attacks</td>
<td>Tampering</td>
<td>Packets or messages without sequence numbers or timestamps can be captured and replayed in a wide variety of ways. Implement or utilize an existing communication protocol that supports anti-replay techniques (investigate sequence numbers before timers) and strong integrity.</td>
</tr>
</tbody>
</table>

Continued on next page
Table A.1 – continued from previous page

<table>
<thead>
<tr>
<th>ID</th>
<th>Threat</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Camera Process Memory Tampered</td>
<td>Tampering</td>
<td>If Camera is given access to memory, such as shared memory or pointers, or is given the ability to control what Nest App executes (for example, passing back a function pointer.), then Camera can tamper with Nest App. Consider if the function could work with less access to memory, such as passing data rather than pointers. Copy in data provided, and then validate it.</td>
</tr>
<tr>
<td>4</td>
<td>Weak Authentication Scheme</td>
<td>Information Disclosure</td>
<td>Custom authentication schemes are susceptible to common weaknesses such as weak credential change management, credential equivalence, easily guessable credentials, null credentials, downgrade authentication or a weak credential change management system. Consider the impact and potential mitigations for your custom authentication scheme.</td>
</tr>
<tr>
<td>5</td>
<td>Elevation Using Impersonation</td>
<td>Elevation Of Privilege</td>
<td>Nest App may be able to impersonate the context of Camera in order to gain additional privilege.</td>
</tr>
<tr>
<td>6</td>
<td>Elevation by Changing the Execution Flow in Nest App</td>
<td>Elevation Of Privilege</td>
<td>An attacker may pass data into Nest App in order to change the flow of program execution within Nest App to the attacker’s choosing.</td>
</tr>
<tr>
<td>7</td>
<td>Nest App May be Subject to Elevation of Privilege Using Remote Code Execution</td>
<td>Elevation Of Privilege</td>
<td>Camera may be able to remotely execute code for Nest App.</td>
</tr>
<tr>
<td>ID</td>
<td>Threat</td>
<td>Category</td>
<td>Description</td>
</tr>
<tr>
<td>----</td>
<td>--------------------------------------------</td>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>8</td>
<td>Data Flow Bluetooth Is Potentially Interrupted</td>
<td>Denial Of Service</td>
<td>An external agent interrupts data flowing across a trust boundary in either direction.</td>
</tr>
<tr>
<td>9</td>
<td>Potential Process Crash or Stop for Nest App</td>
<td>Denial Of Service</td>
<td>Nest App crashes, halts, stops or runs slowly; in all cases violating an availability metric.</td>
</tr>
<tr>
<td>10</td>
<td>Weak Credential Transit</td>
<td>Information Disclosure</td>
<td>Credentials on the wire are often subject to sniffing by an attacker. Are the credentials re-usable/re-playable? Are credentials included in a message? For example, sending a zip file with the password in the email. Use strong cryptography for the transmission of credentials. Use the OS libraries if at all possible, and consider cryptographic algorithm agility, rather than hard-coding a choice.</td>
</tr>
<tr>
<td>11</td>
<td>Potential Data Repudiation by Nest App</td>
<td>Repudiation</td>
<td>Nest App claims that it did not receive data from a source outside the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.</td>
</tr>
<tr>
<td>12</td>
<td>Elevation Using Impersonation</td>
<td>Elevation Of Privilege</td>
<td>Camera may be able to impersonate the context of Nest App in order to gain additional privilege.</td>
</tr>
<tr>
<td>ID</td>
<td>Threat</td>
<td>Category</td>
<td>Description</td>
</tr>
<tr>
<td>----</td>
<td>------------------------------------------------------------------------</td>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>13</td>
<td>Thick Client Process Memory Tampered</td>
<td>Tampering</td>
<td>If Nest App is given access to memory, such as shared memory or pointers, or is given the ability to control what Camera executes (for example, passing back a function pointer.), then Nest App can tamper with Camera. Consider if the function could work with less access to memory, such as passing data rather than pointers. Copy in data provided, and then validate it.</td>
</tr>
<tr>
<td>14</td>
<td>Elevation by Changing the Execution Flow in Camera</td>
<td>Elevation Of Privilege</td>
<td>An attacker may pass data into Camera in order to change the flow of program execution within Camera to the attacker’s choosing.</td>
</tr>
<tr>
<td>15</td>
<td>Camera May be Subject to Elevation of Privilege Using Remote Code Execution</td>
<td>Elevation Of Privilege</td>
<td>Nest App may be able to remotely execute code for Camera.</td>
</tr>
<tr>
<td>16</td>
<td>Data Flow Bluetooth Is Potentially Interrupted</td>
<td>Denial Of Service</td>
<td>An external agent interrupts data flowing across a trust boundary in either direction.</td>
</tr>
<tr>
<td>17</td>
<td>Potential Process Crash or Stop for Camera</td>
<td>Denial Of Service</td>
<td>Camera crashes, halts, stops or runs slowly; in all cases violating an availability metric.</td>
</tr>
<tr>
<td>ID</td>
<td>Threat</td>
<td>Category</td>
<td>Description</td>
</tr>
<tr>
<td>----</td>
<td>-----------------------------</td>
<td>------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>18</td>
<td>Weak Credential Transit</td>
<td>Information Disclosure</td>
<td>Credentials on the wire are often subject to sniffing by an attacker. Are the credentials re-usable/re-playable? Are credentials included in a message? For example, sending a zip file with the password in the email. Use strong cryptography for the transmission of credentials. Use the OS libraries if at all possible, and consider cryptographic algorithm agility, rather than hard-coding a choice.</td>
</tr>
<tr>
<td>19</td>
<td>Potential Data Repudiation by Camera</td>
<td>Repudiation</td>
<td>Camera claims that it did not receive data from a source outside the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.</td>
</tr>
<tr>
<td>20</td>
<td>Weak Authentication Scheme</td>
<td>Information Disclosure</td>
<td>Custom authentication schemes are susceptible to common weaknesses such as weak credential change management, credential equivalence, easily guessable credentials, null credentials, downgrade authentication or a weak credential change management system. Consider the impact and potential mitigations for your custom authentication scheme.</td>
</tr>
<tr>
<td>21</td>
<td>Replay Attacks</td>
<td>Tampering</td>
<td>Packets or messages without sequence numbers or timestamps can be captured and replayed in a wide variety of ways. Implement or utilize an existing communication protocol that supports anti-replay techniques (investigate sequence numbers before timers) and strong integrity.</td>
</tr>
</tbody>
</table>

*Continued on next page*
<table>
<thead>
<tr>
<th>ID</th>
<th>Threat</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Collision Attacks</td>
<td>Tampering</td>
<td>Attackers who can send a series of packets or messages may be able to overlap data. For example, packet 1 may be 100 bytes starting at offset 0. Packet 2 may be 100 bytes starting at offset 25. Packet 2 will overwrite 75 bytes of packet 1. Ensure you reassemble data before filtering it, and ensure you explicitly handle these sorts of cases.</td>
</tr>
<tr>
<td>23</td>
<td>Spoofing of Source Data Store Cloud Storage</td>
<td>Spoofing</td>
<td>Cloud Storage may be spoofed by an attacker and this may lead to incorrect data delivered to Weave Web API. Consider using a standard authentication mechanism to identify the source data store.</td>
</tr>
<tr>
<td>24</td>
<td>Weak Access Control for a Resource</td>
<td>Information Disclosure</td>
<td>Improper data protection of Cloud Storage can allow an attacker to read information not intended for disclosure. Review authorization settings.</td>
</tr>
<tr>
<td>25</td>
<td>Spoofing of Destination Data Store Cloud Storage</td>
<td>Spoofing</td>
<td>Cloud Storage may be spoofed by an attacker and this may lead to data being written to the attacker’s target instead of Cloud Storage. Consider using a standard authentication mechanism to identify the destination data store.</td>
</tr>
<tr>
<td>26</td>
<td>Spoofing the Weave Web API External Entity</td>
<td>Spoofing</td>
<td>Weave Web API may be spoofed by an attacker and this may lead to unauthorized access to Nest App. Consider using a standard authentication mechanism to identify the external entity.</td>
</tr>
<tr>
<td>27</td>
<td>Elevation Using Impersonation</td>
<td>Elevation Of Privilege</td>
<td>Nest App may be able to impersonate the context of Weave Web API in order to gain additional privilege.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>ID</th>
<th>Threat</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>JavaScript Object Notation Processing</td>
<td>Tampering</td>
<td>If a dataflow contains JSON, JSON processing and hijacking threats may be exploited.</td>
</tr>
<tr>
<td>29</td>
<td>Potential Data Repudiation by Nest App</td>
<td>Repudiation</td>
<td>Nest App claims that it did not receive data from a source outside the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.</td>
</tr>
<tr>
<td>30</td>
<td>Potential Process Crash or Stop for Nest App</td>
<td>Denial Of Service</td>
<td>Nest App crashes, halts, stops or runs slowly; in all cases violating an availability metric.</td>
</tr>
<tr>
<td>31</td>
<td>Data Flow HTTPS Is Potentially Interrupted</td>
<td>Denial Of Service</td>
<td>An external agent interrupts data flowing across a trust boundary in either direction.</td>
</tr>
<tr>
<td>32</td>
<td>Nest App May be Subject to Elevation of Privilege Using Remote Code Execution</td>
<td>Elevation Of Privilege</td>
<td>Weave Web API may be able to remotely execute code for Nest App.</td>
</tr>
<tr>
<td>33</td>
<td>Elevation by Changing the Execution Flow in Nest App</td>
<td>Elevation Of Privilege</td>
<td>An attacker may pass data into Nest App in order to change the flow of program execution within Nest App to the attacker’s choosing.</td>
</tr>
<tr>
<td>34</td>
<td>Spoofing of the Weave Web API External Destination Entity</td>
<td>Spoofing</td>
<td>Weave Web API may be spoofed by an attacker and this may lead to data being sent to the attacker’s target instead of Weave Web API. Consider using a standard authentication mechanism to identify the external entity.</td>
</tr>
</tbody>
</table>

Continued on next page
Table A.1 – continued from previous page

<table>
<thead>
<tr>
<th>ID</th>
<th>Threat</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>External Entity Weave Web API Potentially Denies Receiving Data</td>
<td>Repudiation</td>
<td>Weave Web API claims that it did not receive data from a process on the other side of the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.</td>
</tr>
<tr>
<td>36</td>
<td>Data Flow HTTPS Is Potentially Interrupted</td>
<td>Denial Of Service</td>
<td>An external agent interrupts data flowing across a trust boundary in either direction.</td>
</tr>
<tr>
<td>37</td>
<td>Spoofing the Human User External Entity</td>
<td>Spoofing</td>
<td>Human User may be spoofed by an attacker and this may lead to unauthorized access to Nest App. Consider using a standard authentication mechanism to identify the external entity.</td>
</tr>
<tr>
<td>38</td>
<td>Elevation Using Impersonation</td>
<td>Elevation Of Privilege</td>
<td>Nest App may be able to impersonate the context of Human User in order to gain additional privilege.</td>
</tr>
<tr>
<td>39</td>
<td>Elevation by Changing the Execution Flow in Camera</td>
<td>Elevation Of Privilege</td>
<td>An attacker may pass data into Camera in order to change the flow of program execution within Camera to the attacker’s choosing.</td>
</tr>
<tr>
<td>40</td>
<td>Camera May be Subject to Elevation of Privilege Using Remote Code Execution</td>
<td>Elevation Of Privilege</td>
<td>Cloud Storage may be able to remotely execute code for Camera.</td>
</tr>
<tr>
<td>41</td>
<td>Spoofing of Source Data Store Cloud Storage</td>
<td>Spoofing</td>
<td>Cloud Storage may be spoofed by an attacker and this may lead to incorrect data delivered to Camera. Consider using a standard authentication mechanism to identify the source data store.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>ID</th>
<th>Threat</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>Weak Access Control for a Resource</td>
<td>Information Disclosure</td>
<td>Improper data protection of Cloud Storage can allow an attacker to read information not intended for disclosure. Review authorization settings.</td>
</tr>
<tr>
<td>43</td>
<td>Data Store Inaccessible</td>
<td>Denial Of Service</td>
<td>An external agent prevents access to a data store on the other side of the trust boundary.</td>
</tr>
<tr>
<td>44</td>
<td>Data Flow IPSec Is Potentially Interrupted</td>
<td>Denial Of Service</td>
<td>An external agent interrupts data flowing across a trust boundary in either direction.</td>
</tr>
<tr>
<td>45</td>
<td>Potential Process Crash or Stop for Camera</td>
<td>Denial Of Service</td>
<td>Camera crashes, halts, stops or runs slowly; in all cases violating an availability metric.</td>
</tr>
<tr>
<td>46</td>
<td>Potential Data Repudiation by Camera</td>
<td>Repudiation</td>
<td>Camera claims that it did not receive data from a source outside the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.</td>
</tr>
<tr>
<td>47</td>
<td>Potential Excessive Resource Consumption for Camera or Cloud Storage</td>
<td>Denial Of Service</td>
<td>Does Camera or Cloud Storage take explicit steps to control resource consumption? Resource consumption attacks can be hard to deal with, and there are times that it makes sense to let the OS do the job. Be careful that your resource requests don’t deadlock, and that they do timeout.</td>
</tr>
<tr>
<td>48</td>
<td>Authorization Bypass</td>
<td>Information Disclosure</td>
<td>Can you access Cloud Storage and bypass the permissions for the object? For example by editing the files directly with a hex editor, or reaching it via filesharing? Ensure that your program is the only one that can access the data, and that all other subjects have to use your interface.</td>
</tr>
<tr>
<td>ID</td>
<td>Threat</td>
<td>Category</td>
<td>Description</td>
</tr>
<tr>
<td>----</td>
<td>--------------------------------------------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>49</td>
<td>Spoofing of Destination Data Store Cloud Storage</td>
<td>Spoofing</td>
<td>Cloud Storage may be spoofed by an attacker and this may lead to data being written to the attacker’s target instead of Cloud Storage. Consider using a standard authentication mechanism to identify the destination data store.</td>
</tr>
<tr>
<td>50</td>
<td>Data Store Inaccessible</td>
<td>Denial Of Service</td>
<td>An external agent prevents access to a data store on the other side of the trust boundary.</td>
</tr>
<tr>
<td>51</td>
<td>Data Flow Is Potentially Interrupted</td>
<td>Denial Of Service</td>
<td>An external agent interrupts data flowing across a trust boundary in either direction.</td>
</tr>
<tr>
<td>52</td>
<td>Data Store Denies Cloud Storage Potentially Writing Data</td>
<td>Repudiation</td>
<td>Cloud Storage claims that it did not write data received from an entity on the other side of the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.</td>
</tr>
<tr>
<td>53</td>
<td>The Cloud Storage Data Store Could Be Corrupted</td>
<td>Tampering</td>
<td>Data flowing across IPSec may be tampered with by an attacker. This may lead to corruption of Cloud Storage. Ensure the integrity of the data flow to the data store.</td>
</tr>
</tbody>
</table>
For DIVA

{
  "Author1": { "name": "Jacob Klasmark" },
  "Author2": { "name": "Valter Lundegårdh" },
  "Degree": { "Educational program": "Degree Programme in Information and Communication Technology" },
  "Title": {
    "Main title": "Potential security risks in Google Nest Indoor Camera",
    "Language": "eng"
  },
  "Alternative title": {
    "Main title": "Potentiella säkerhetsrisiker hos Google Nest Indoor Camera",
    "Language": "swe"
  },
  "Supervisor1": { "name": "Pontus Johnson" },
  "Examiner": {
    "name": "Mathias Ekstedt",
    "organisation": "L1: School of Electrical Engineering and Computer Science"
  },
  "Other information": {
    "Year": "2020",
    "Number of pages": "xvi, 100"
  }
}